Automotive Math and Motor Control Library Set for NXP S32K3xx, MWCT2DxxS, and MWCT20xxS devices

User's Guide

Rev. 13 — 15 September 2023 User guide
S32K3XXMCLUG

Revision History

Revision	Date	Description
1.0	30-September-2020	Initial version.
2.0	31-December-2020	The Service Release v1.1.23. See release notes for list of changes.
3.0	31-March-2021	The Service Release v1.1.24. See release notes for list of changes.
4.0	30-June-2021	The Service Release v1.1.25. See release notes for list of changes.
5.0	30-September-2021	The Service Release v1.1.26. See release notes for list of changes.
6.0	31-December-2021	The Service Release v1.1.27. See release notes for list of changes.
7.0	31-March-2022	The Service Release v1.1.28. See release notes for list of changes.
8.0	30-June-2022	The Service Release v1.1.29. See release notes for list of changes.
9.0	30-September-2022	The Service Release v1.1.30. See release notes for list of changes.
10.0	31-December-2022	The Service Release v1.1.31. See release notes for list of changes.
11.0	31-March-2023	The Service Release v1.1.32. See release notes for list of changes.
12.0	30-June-2023	The Service Release v1.1.33. See release notes for list of changes.
13.0	30-September-2023	The Service Release v1.1.34. See release notes for list of changes.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

1 Introduction

This document describes the Automotive Math and Motor Control Library Set for NXPS32K3xx (S32K311, S32K312, S32K314, S32K322, S32K324, S32K328, S32K338, S32K341, S32K342, S32K344, S32K348, S32K358, S32K374, S32K376, S32K394, S32K396), MWCT2DxxS, and MWCT20xxS devices (AMMCLib). It lists all components of the library, the API, and steps needed to integrate the library into the customer project.

Note: *MWCT part numbers contain NXP confidential IP for Qi Wireless Power.*

1.1 Architecture Overview

The AMMCLib consists of several sub-libraries, functionally connected as depicted in Figure 1 .

Figure 1. AMMCLIB components structure

The AMMCLib sub-libraries are as follows:

- **Mathematical Function Library (MLIB)** - comprising basic mathematical operations such as addition, multiplication, etc.
- **General Function Library (GFLIB)** - comprising basic trigonometric and general math functions such as sine, cosine, tan, hysteresis, limit, etc.
- **General Digital Filters Library (GDFLIB)** - comprising digital IIR and FIR filters designed to be used in a motor control application
- **General Motor Control Library (GMCLIB)** - comprising standard algorithms used for motor control such as Clarke/Park transformations, Space Vector Modulation, etc.
- **Advanced Motor Control Function Library (AMCLIB)** - comprising advanced algorithms used for motor control purposes

As can be seen in Figure 1 , the AMMCLib libraries form a layered architecture where all higher libraries utilize functions from the MLIB library. MLIB functions are inlined and

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

assembly-optimized for maximum performance. Library functions facilitate abstraction of the computational hardware to the application software.

1.2 General Information

The AMMCLib was developed to support these major implementations:

- Fixed-point 32-bit fractional
- Fixed-point 16-bit fractional
- Single precision floating point

With exception of those functions where the mathematical principle limits the input or output values, these values are considered to be in the following limits:

- **Fixed-point 32-bit fractional**: <-1; $1-2^{-31}$> in Q1.31 format and with minimum positive normalized value 2^{-31}.
- **Fixed-point 16-bit fractional**: <-1; $1-2^{-15}$> in Q1.15 format and with minimum positive normalized value 2^{-15}.
- **Single precision floating point**: <-2^{128}; 2^{128}> and with minimum positive normalized value 2^{-128}.

Functions which are not relevant for a particular implementation, e.g. saturated arithmetic for floating point implementation, are not delivered with this package. For detailed information about available functions, refer to Section 2.1 .

The fixed-point 32-bit fractional and fixed-point 16-bit fractional functions implementions are based on the unity model. This means that the input and output physical values (volts, amperes, radians) must be normalized to fit within the <-1; $1-2^{-31}$> or <-1; $1-2^{-15}$> range representable in the Q1.31 or Q1.15 format.

All functions in the AMMCLib were tested in the target device with all supported compilers to ensure that they achieve the declared accuracy of results in the defined input range.

1.3 Installation

The AMMCLib is delivered as a single executable file.

Note: *The AMMCLib is available as a plugin for the* NXP S32 Design Studio IDE*, therefore the manual installation step can be skipped when using this development tool.*

The following steps will install the AMMCLib on a Windows-based computer:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

1. On welcome page select the Next to start the installation

Figure 2. AMMCLib installation - step 1. Highlighted "MCU Platform" and "ReleaseID" identifies the actual release, which is the S32K3xx_AMMCLIB_v1.1.34

2. Accept the license agreement

Figure 3. AMMCLib installation - step 2

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

3. Select the destination directory

Figure 4. AMMCLib installation - step 3. Highlighted "Platform" and "ReleaseID" identifies the actual release installation path, which is the S32K3xx_AMMCLIB_v1.1.34

4. Check the destination directory and confirm the installation

Figure 5. AMMCLib installation - step 4. Highlighted "Platform" and "ReleaseID" identifies the actual release installation path, which is the S32K3xx_AMMCLIB_v1.1.34

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5. After installation carefully read the Release notes with important additional information about the release

Figure 6. AMMCLib installation - step 5. Highlighted "Platform" and "ReleaseID" identifies the actual release, which is the S32K3xx_AMMCLIB_v1.1.34

6. Select Finish to end the installation

Figure 7. MCLib installation - step 6

1.4 Library File Structure

After a successful installation, the AMMCLib is added by default into the "*C:\NXP \AMMCLIB\S32K3xx_AMMCLIB_v1.1.34* " subfolder. This folder will contain other nested subfolders and files required by the AMMCLib, as shown in Figure 8 .

© NXP B.V. 2023. All rights reserved.

Figure 8. AMMCLIB directory structure

A list of the installed directories/files, and their brief description, is given below:

- **bam** - contains Bit Accurate Models of all the functions for Matlab/Simulink
- **doc** - contains the User Manual
- **include** - contains all the header files, including the master header files of each library to be included in the user application
- **lib** - contains the compiled library file to be included in the user application
- **src** - contains all source files (only available in the source code licensed package)

The header files structure of the AMMCLib is depicted in Figure 9 .

Figure 9. AMMCLIB header file structure

1.5 Multiple Implementation Support

Most library functions are provided in several different implementations, each based on a different primary data type. The accuracy of results and performance may vary between different implementations. Selecting the right implementation for the given processor ensures optimal use of available resources. The preferred implementation can be selected with one of the available call mechanisms:

- API postfix option
- Additional parameter option

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

- Global configuration option

1.5.1 API Postfix Option

The API postfix option calls the selected variant of the function directly, e.g. *GFLIB_Sin_F32(x, GFLIB_SIN_DEFAULT_F32)*. The following two options use overloading macros to switch between the implementation variants.

1.5.2 Additional Parameter Option

The desired implementation is selected by an additional function parameter. The implementation postfix is removed from the function name, e.g. *GFLIB_Sin(x, GFLIB_SIN_DEFAULT_F32, F32)*. The following parameter choices can be used:

- **F32** for 32-bit fixed-point implementation
- **F16** for 16-bit fixed-point implementation
- **FLT** for single precision floating point implementation

1.5.3 Global Configuration Option

This option allows selecting the preferred implementation globally for all functions by setting the *SWLIBS_DEFAULT_IMPLEMENTATION* macro in the file SWLIBS_Config.h. The implementation postfix is removed from the function name so the function call looks the same regardless of the selected implementation. This allows the simplest API call, e.g. *GFLIB_Sin(x)*.

The following definitions are allowed for *SWLIBS_DEFAULT_IMPLEMENTATION*:

- **SWLIBS_DEFAULT_IMPLEMENTATION_F32** for 32-bit fixed-point implementation
- **SWLIBS_DEFAULT_IMPLEMENTATION_F16** for 16-bit fixed-point implementation
- **SWLIBS_DEFAULT_IMPLEMENTATION_FLT** for single precision floating point implementation

Note: *It is recommended to copy SWLIBS_Config.h file to your local project structure and refer the configuration to this local copy. This approach will prevent incorrect setup of default configuration option in case multiple projects with different default configuration exist.*

1.6 Supported Compilers

The AMMCLib is written in ANSI-C99 ISO/IEC 9899:1999 standard language with critical parts implemented in assembly. The library was built and tested using the following compilers:

1. Green Hills MULTI v2021.1.4
2. IAR Embedded Workbench for ARM 8.50.10.290
3. S32 Design Studio for ARM based MCUs 3.5 (GCC compiler version 10.2.0 20200723)

The precompiled library module " *S32K3xx_AMMCLIB.a*" for the Green Hills compiler is located in "*C:\NXP\AMMCLIB\S32K3xx_AMMCLIB_v1.1.34\lib\ghs*" folder (considering the default installation path).

The precompiled library module " *S32K3xx_AMMCLIB.a*" for the IAR compiler is located in "*C:\NXP\AMMCLIB\S32K3xx_AMMCLIB_v1.1.34\lib\iar*" folder (considering the default installation path).

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The precompiled library module " *S32K3xx_AMMCLIB.a*" for the S32 Design Studio IDE for ARM based MCUs is located in "*C:\NXP\AMMCLIB\S32K3xx_AMMCLIB_v1.1.34\lib \s32ds_arm32*" folder (considering the default standalone installation path). The library is also pre-installed with the S32 Design Studio IDE for ARM based MCUs is located in "*<S32DS installation directory>\S32DS\S32K3xx_AMMCLIB_v1.1.34\lib\s32ds_arm32*" folder.

Together with the pre-compiled library modules, these are all the necessary header files. The interfaces to the algorithms included in this library have been combined into a single public interface header file for each respective sub-library, i.e. *mlib.h*, *gflib.h*, *gdflib.h* and *gmclib.h*. This was done to simplify the number of files required for inclusion by application programs. Refer to the specific algorithm sections of this document for details on the software Application Programming Interface (API), definitions and functionality provided for the individual algorithms.

1.7 Integration Assumption

In order to successfully integrate the AMMCLib to the customer application, the following assumptions need to be considered:

1. The C-99 language has to be enabled in the user application (please refer to User manual of your compiler to enable this feature).
2. In case the floating point unit is available on target MCU, the single precision floating point HW support has to be switched on (please refer to User manual of your compiler to enable this feature).

1.8 Library Integration into a Green Hills Multi Development Environment

The AMMCLib is added into a new Green Hills Multi project using the following steps:

1. Open a new empty C project in the Green Hills Multi IDE. See the Green Hills Multi user manual for instructions.
2. Once you have successfully created and opened a new C project, right click on the project file *.gpj in the GHS Multi Project Manager. Select <*Set Build Options...*> from the pop-up menu, as shown in Figure 10

Figure 10. Project build options

3. In the <*Basic Options*> tab of the <*Build Options*> window, expand section <Project> and double click on the item <*Include Directories (-I)*>. Here, the directory where the

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

builder should look for all the project header files, including the library header files, shall be specified as shown in Figure 11 . Considering default settings, the following path shall be added: "*C:\NXP\AMMCLIB\S32K3xx_AMMCLIB_v1.1.34\include.* "

Figure 11. Adding a path to the library header files

4. While still in the <*Project*> section, double click on <*Libraries (-l)*> and enter a path and a pre-compiled library object file into the dialogue box, as shown in Figure 12 .

Figure 12. Adding a path to the library object files

Considering default settings, you should add *"C:\NXP\AMMCLIB \S32K3xx_AMMCLIB_v1.1.34\lib\ghs\S32K3xx_AMMCLIB.a"*

5. In order to use the library functions, the library master header files must be included into the application source code. This is done using the pre-processor directive *#include "<libID>.h"*, where <libID> can be *amclib, gdflib, gflib, gmclib, mlib,* depending on which library is to be employed.

The master header files contain several additional header files that are needed for the AMMCLib integration into any user application. They include the "*SWLIBS_Typedefs.h*" header file which contains all general purpose data type definitions, the "*mlib.h*" header file containing all general math functions, the "*SWLIBS_Defines.h*" file containing common macro definitions and the "*SWLIBS_MacroDisp.h*" allowing the use of simplified API calls.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 13. Opening the C editor for editing the application source code

Figure 14. Using the pre-processor directive to include library master header files

1.9 Library Integration into a IAR Project

The AMMCLib is added into a IAR project using the following steps:

1. Open a new empty C project in the IAR Embedded Workbench IDE. See the IAR Embedded Workbench Getting Started manual for instructions.
2. Once having the new project successfully created and opened, the AMMCLib files needs to be added to the project.
3. In order to integrate the AMMCLib in the IAR project, it is necessary to provide the IAR Embedded Workbench IDE with the access paths to the AMMCLib files. Two access paths needs to be added:
   a. Access path to the AMMCLib header files.
   b. Access path to the AMMCLib binary file.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

4. To add the necessary access paths, select the project from the workspace list (**AMMCLib_Example_IAR** in the example shown in Figure 15) and from the main menu select <*Project*> - <*Options*> item.

Figure 15. Selecting the project properties

5. To add the AMMCLib header files, in the <*Options for node*> window select the <*C/C ++ Compiler*> category. Here, choose the <*Preprocessor*> tab and on the right side of <*Additional include directories:*> click on the dot symbols, as shown in Figure 16 .

Figure 16. Adding the include directory

© NXP B.V. 2023. All rights reserved.

6. In the <*Edit Include Directories*> click on the <*Click to add*> as shown in Figure 17

Figure 17. Adding the include directory to the list

7. Considering the default paths, the <*Edit Include Directories*> should look as shown in Figure 18

Figure 18. Adding the default include path

8. While still in the <*Options for node*> window, add the AMMCLib binary file. In the <*Options for node*> window select the <*Linker*> category. Here, choose the <*Library*> tab and on the right side of <*Additional libraries:*> click on the dot symbols, as shown in Figure 19 .

Figure 19. Adding the binary directory

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

9. In the <*Edit Additional Libraries*> click on the <*Click to add*> as shown in Figure 20

Figure 20. Adding the binary directory to the list

10. Considering the default paths, the <*Edit Additional Libraries*> should look as shown in Figure 21

Figure 21. Adding the default binary path

11. In order to use the library functions, the library master header files must be included into the application source code. This is done using the pre-processor directive *#include "<libID>.h"*, where <libID> can be *amclib, gdflib, gflib, gmclib, mlib,* depending on which library is to be employed.

The master header files contain several additional header files that are needed for the AMMCLib integration into any user application. They include the "*SWLIBS_Typedefs.h*" header file which contains all general purpose data type definitions, the "*mlib.h*" header file containing all general math functions, the "*SWLIBS_Defines.h*" file containing common macro definitions and the "*SWLIBS_MacroDisp.h*" allowing the use of simplified API calls.

1.10 Library Integration into a S32 Design Studio for S32 Platform as a RTD plugin

The AMMCLib is provided as an RTD plugin for the S32 Design Studio IDE. To install the library plugin, go to *Help*, select *S32DS Extensions and Updates* and install the AMMCLib plugin. Make sure to select the latest version of the library. The library can be installed only if the required version of RTD had already been installed.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 22. Installation of the AMMCLib plugin

To add the library to an existing RTD-based project, open the S32 Configuration Tools, select Peripherals, and add the AMMCLib from the Libraries section of the Components tab. Make sure to click the Update Code once the library is added.

Figure 23. Adding the AMMCLib plugin to the project

The library configuration panel allows selecting the default implementation and enabling/disabling selected implementations. Refer to the chapter Multiple Implementation Support for more details.

Figure 24. AMMCLib plugin configuration

Make sure that the S32 Design Studio contains a supported compiler version. The list of supported toolchains is provided in the AMMCLib Release Notes (can be accessed by going to the *Help/Getting Started/EXTENSIONS/AMMCLib/More* menu).

In order to use the AMMCLib functions, the library master header files must be included into the application source code. This is done using the pre-processor directive *#include "<libID>.h"*, where <libID> can be *amclib, gdflib, gflib, gmclib, mlib,* depending on which library is to be employed.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The master header files contain several additional header files which are needed for the AMMCLib integration into the user application. They include the "*SWLIBS_Typedefs.h*" header file which contains all general purpose data type definitions, the "mlib.h" header file containing all general math functions, the "*SWLIBS_Defines.h*" file containing common macro definitions and the "*SWLIBS_MacroDisp.h*" allowing the use of simplified API calls.

1.11 Library Integration into a S32 Design Studio for S32 Platform as a standalone package

In order to use the AMMCLib within the S32 Design Studio GUI environment and it is not desired to use the library as a RTD plugin, it is necessary to set the access paths for the AMMCLib files.

The following files need to be added to the project:

- Library binary files located in the directory *"C:\NXP\AMMCLIB \S32K3xx_AMMCLIB_v1.1.34\lib\s32ds_arm32"* (note: this is the default location and may be modified during library installation). Note that a colon must be added as a prefix of the library name in the S32 Design Studio project settings.
- Header files located in the directory *"C:\NXP\AMMCLIB \S32K3xx_AMMCLIB_v1.1.34\include"* (note: this is the default location and may be modified during library installation)

When creating a new S32 Design Studio project, make sure the supported version of a compiler is selected. In order to use the library functions, the library master header files must be included into the application source code. This is done using the pre-processor directive *\#include "<libID>.h"*, where <libID> can be *amclib, gdflib, gflib, gmclib mlib,* depending on which library is to be employed.

1.12 MATLAB Integration

AMMCLib provides Bit Accurate Models (BAM) for all library functions. These models can be used in MATLAB Simulink to simulate the behavior of each function in real implementation as well as generate target code using the Embedded Coder®. The BAMs come in a common library file *ammclib_bam_S32K3xx.slx* located in the "bam" folder in the library installation path. In order to get the BAMs working in Simulink, it is necessary to add the "bam" folder and all of its subfolders into the MATLAB Search Path.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 25. Matlab integration - set search path

The Bit Accurate Models were compiled and tested using MATLAB R2019a and MinGW 6.3 C compiler. The files are saved in format compatible with Matlab R2016a for backwards compatibility. Only C language is supported for model-based code generation. The Bit Accurate Models were tested with the following versions of Matlab: R2019a, R2019b, R2020a, R2020b, R2021a, R2021b, R2022a, both for simulation and code generation.

The first thing to do after opening a new Simulink schematic is to call the script **defaultCodegenCfg_S32K3xx** to configure the default Model Configuration Parameters of the currently selected Simulink schematic. The script is located in the "bam" folder in the library installation path. Note that the script sets only the most basic properties to avoid common errors. The user should carefully examine the Model Configuration Parameters and set all preferences according to the actual needs.

BAMs of functions with internal state can be configured to make the state variables accessible as input/output signals for debugging purposes. Note that in this configuration the performance is limited and the user must manually connect all feedback loops, i.e. add a unit delay block between a state signal input (e.g. *f32Acc*) and the corresponding state signal output (e.g. *f32Acc_Out*). The internal state is automatically reset during model initialization. Some BAMs allow setting the internal state to a predefined nonzero value during the reset. This feature can be used for seamless transition from an open loop control to a closed loop control in applications such as the sensorless field-oriented control (FOC). Resettable blocks provide rising edge-triggered reset input (marked with "R").

BAMs of matrix/vector functions expect native Simulink matrix/vector signals in column-major arrangement on their signal inputs. The matrix/vector size defined for these signals in Simulink is automatically propagated in the underlying C function as an API parameter.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Functions with complex-valued inputs/outputs use complex signal type in the Simulink interface. The underlying C code recasts the complex signals into an array of interleaved real/imaginary pairs.

Several BAMs use nonvirtual buses for efficient representation of struct signal types. The definitions of the data types required by Simulink are automatically loaded from the data dictionary *ammclib_types.sldd* located in the "bam" folder. A nonvirtual bus can be created from individual signals using the bus creator block. Note that it is necessary to make the following changes to the default properties of the bus creator block:

- Set the *Output data type* to a Bus type corresponding to the struct type of the connected AMMCLIB function port.
- Disable the option *Override bus signal names from inputs*.
- Enable the option *Output as nonvirtual bus*.

Figure 26. Configuring the Simulink Bus Creator block

Individual signals can be extracted from a nonvirtual bus using the bus selector block. Once the bus selector is connected to the bus, it is necessary to manually select the bus signals, see the following figure.

© NXP B.V. 2023. All rights reserved.

Figure 27. Configuring the Simulink Bus Selector block

The BAMs contain sources auto-generated by MATLAB®. © 1984 - 2020 The MathWorks, Inc.

1.13 Library Testing

In order to validate the implementation of the AMMCLib, the comparison of results from the Matlab Reference Model and outputs from the tested library function is used. To ensure the AMMCLib precision, two test methods are used:

- Matlab Simulink Toolbox based testing (refer to Section 1.13.1 for more details).
- Target-in-loop based testing (refer to Section 1.13.2 for detailed information).

The Figure 28 shows the testing principle:

- **Input vector** represents the test vector which enters simultaneously into the Reference Model and into the Unit Under Test (UUT).
- **Reference Model (RM)** implements the real model of the UUT. For simple functions, the models are a part of the Matlab Simulink Toolbox. Advanced functions such as filters or controllers had been designed separately.
- By the type of test method used, the **Unit Under Test (UUT)** may be:
  - **Bit Accurate Model (BAM)** - the "C" implementation of the tested function compiled in the Matlab environment. The compilation result, called the binary MEX-file, is a dynamically-linked subroutine that the Matlab interpreter can load and execute.
  - **SFIO Model** represents the tested function running directly on the target MCU.

Results from the UUT and Reference Model are saved in the final report, together with the calculated error which is simply the difference between the output value from the Reference Model and the output value from the UUT, recalculated to an equivalent precision.

The output value of the function is determined by the following expressions:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

- **32-bit fixed point:** <PRECISE_VALUE> +/- e*2^-15
- **16-bit fixed point:** <PRECISE_VALUE> +/- e*2^-15

where e is the allowed approximation error (see Section 1.14 for specific values).

For single precision floating point implementations, the output accuracy is measured in ULP (units in the last place) and is specified in a separate document S32K3XXMCFLTACC.pdf.

Figure 28. Principle of AMMCLIB testing

In order to test the UUT under all conditions, three types of test vector sets are used:

- Deterministic vectors - a specifically defined set of input values over the entire input range.
- Stochastic vectors - a pseudo-randomly generated set of values (non-deterministic values fully covering the input range).
- Boundary vectors - a set of input values for which the potential weaknesses of the tested function are expected. This test is performed only on functions where these limit conditions might occur.

Each function is considered tested if the required accuracy during deterministic, stochastic and boundary tests has been achieved. The following two subchapters describe the differences between AMMCLIB testing based on BAM models and target-in-loop testing based on SFIO models.

1.13.1 AMMCLIB Testing based on the Matlab Simulink Toolbox

An example of the testing principle based on the BAM is depicted in the Clark transformation function (Figure 29). The Bit Accurate Model contains the binary MEX-file built from the GMCLIB_Clark function using the Matlab compiler. This file is called inside the BAM model, see Figure 30 . The Reference Model of the Clark transformation is not included in the Matlab Simulink Toolbox and hence its mathematical representation had to be created. A detailed scheme of the Clark RM is in Figure 31 .

Figure 29. Testing of the GMCLIB_Clark function based on the Matlab Simulink Toolbox

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 30. Bit Accurate Model parameters of the Clark transformation

Figure 31. Reference Model of the GMCLIB_Clark function

1.13.2 AMMCLIB target-in-loop Testing based on the SFIO Toolbox

The testing method in Figure 32 is the similar to that described in the previous chapter with exception that the BAM model is replaced by the SFIO model. The SFIO Toolbox realizes the bridge between Matlab and the Embedded target. During testing, the function GMCLIB_Clark is called directly from the application running on the target MCU. Unlike testing based on Matlab, the target-in-loop method verifies that the implementation of the AMMCLib functions works correctly on the target MCU. Moreover, the SFIO application running on the processor is used to measure performance of the functions.

The SFIO block Set-up allows the setting of communication parameters which are common to the whole scheme.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 32. Target-in-loop testing example based on the SFIO Toolbox

1.14 Functions Accuracy

The maximum allowed error of the library functions vary based on the implementation, and based on the character of the function. The output error is calculated as the difference between the implemented function and a double-precision reference model, as described in the Section 1.13 section. The actual output error of the function for current input values is smaller or equal to the maximum allowed error described in the table below.

The output error of the 32-bit and 16-bit fixed-point implementations is calculated as an absolute error.

Accuracy of the floating-point functions is described in a separate document S32K3XXMCFLTACC.pdf.

The following table describes the maximum absolute error of the fixed-point functions measured in the 16-bit/32-bit LSB multiples (LSB16/LSB32):

Table 1. Maximum absolute error

Function name	F32 variant	F16 variant
GDFLIB_FilterFIR	u32Order+1 LSB32	u16Order+1 LSB16
GDFLIB_FilterIIR1	1 LSB16	1 LSB16
GDFLIB_FilterIIR2	1 LSB16	1 LSB16
GDFLIB_FilterMA	1 LSB16	1 LSB16
GFLIB_Acos	3 LSB16	3 LSB16
GFLIB_Asin	3 LSB16	3 LSB16
GFLIB_Atan	3 LSB16	3 LSB16
GFLIB_AtanYX	3 LSB16	3 LSB16
GFLIB_AtanYXShifted	3 LSB16	3 LSB16
GFLIB_ControllerPIp	3 LSB16	3 LSB16
GFLIB_ControllerPIpAW	3 LSB16	3 LSB16
GFLIB_ControllerPIr	3 LSB16	3 LSB16

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 1. Maximum absolute error*...continued*

Function name	F32 variant	F16 variant
GFLIB_ControllerPIrAW	3 LSB16	3 LSB16
GFLIB_Cos	3 LSB16	3 LSB16
GFLIB_Hyst	1 LSB16	1 LSB16
GFLIB_IntegratorTR	3 LSB16	3 LSB16
GFLIB_Limit	1 LSB16	1 LSB16
GFLIB_LowerLimit	1 LSB16	1 LSB16
GFLIB_Lut1D	3 LSB16	3 LSB16
GFLIB_Lut2D	3 LSB16	3 LSB16
GFLIB_Ramp	1 LSB16	1 LSB16
GFLIB_Sign	1 LSB16	1 LSB16
GFLIB_Sin	3 LSB16	3 LSB16
GFLIB_SinCos	3 LSB16	3 LSB16
GFLIB_Sqrt	1 LSB16	3 LSB16
GFLIB_Tan	3 LSB16	3 LSB16
GFLIB_UpperLimit	1 LSB16	1 LSB16
GFLIB_VectorLimit	3 LSB16	3 LSB16
GMCLIB_BetaProjection	5 LSB32	4 LSB16
GMCLIB_BetaProjection3Ph	5 LSB32	4 LSB16
GMCLIB_Clark	1 LSB16	3 LSB16
GMCLIB_ClarkInv	1 LSB16	3 LSB16
GMCLIB_DecouplingPMSM	1 LSB16	3 LSB16
GMCLIB_DoubleSwitching	N/A	3 LSB16
GMCLIB_ElimDcBusRip	3 LSB16	3 LSB16
GMCLIB_Park	2 LSB16	2 LSB16
GMCLIB_ParkInv	1 LSB16	1 LSB16
GMCLIB_PwmIct	1 LSB16	3 LSB16
GMCLIB_SvmStd	1 LSB16	3 LSB16
GMCLIB_SvmU0n	2 LSB32	2 LSB16
MLIB_Abs	1 LSB16	1 LSB16
MLIB_AbsSat	1 LSB16	1 LSB16
MLIB_Add	1 LSB16	1 LSB16
MLIB_AddSat	1 LSB16	1 LSB16
MLIB_Convert_F16F32	N/A	1 LSB16
MLIB_Convert_F16FLT	N/A	1 LSB16
MLIB_Convert_F32F16	1 LSB16	N/A
MLIB_Convert_F32FLT	1 LSB16	N/A
MLIB_Convert_FLTF16	N/A	N/A
MLIB_Convert_FLTF32	N/A	N/A
MLIB_ConvertPU_F16F32	N/A	1 LSB16
MLIB_ConvertPU_F16FLT	N/A	1 LSB16
MLIB_ConvertPU_F32F16	1 LSB16	N/A
MLIB_ConvertPU_F32FLT	1 LSB16	N/A
MLIB_ConvertPU_FLTF16	N/A	N/A
MLIB_ConvertPU_FLTF32	N/A	N/A
MLIB_Div	1 LSB16	1 LSB16
MLIB_DivSat	1 LSB16	1 LSB16

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 1. Maximum absolute error*...continued*

Function name	F32 variant	F16 variant
MLIB_DivSatAccurate	1 LSB32	N/A
MLIB_Mac	1 LSB16	1 LSB16
MLIB_MacSat	1 LSB16	1 LSB16
MLIB_Mnac	1 LSB16	1 LSB16
MLIB_Msu	1 LSB16	1 LSB16
MLIB_Mul	1 LSB16	1 LSB16
MLIB_MulSat	1 LSB16	1 LSB16
MLIB_Neg	1 LSB16	1 LSB16
MLIB_NegSat	1 LSB16	1 LSB16
MLIB_Norm	1 LSB16	1 LSB16
MLIB_RndSat	1 LSB16	1 LSB16
MLIB_Round	1 LSB16	1 LSB16
MLIB_ShBi	1 LSB16	1 LSB16
MLIB_ShBiSat	1 LSB16	1 LSB16
MLIB_ShL	1 LSB16	1 LSB16
MLIB_ShLSat	1 LSB16	1 LSB16
MLIB_ShR	1 LSB16	1 LSB16
MLIB_Sub	1 LSB16	1 LSB16
MLIB_SubSat	1 LSB16	1 LSB16
MLIB_VMac	1 LSB16	2 LSB16
AMCLIB_BemfObsrvDQ	24 LSB16	30 LSB16
AMCLIB_TrackObsrv	3 LSB16	3 LSB16
AMCLIB_CurrentLoop	94 LSB16	232 LSB16
AMCLIB_FW	29 LSB16	28 LSB16
AMCLIB_FWSpeedLoop	29 LSB16	28 LSB16
AMCLIB_SpeedLoop	29 LSB16	28 LSB16
AMCLIB_Windmilling	9 LSB16	23 LSB16

2 Functions

This section describes the Application Interface and implementation details for all functions available in AMMCLib.

2.1 Function index

Table 2. Quick function reference

Type	Name	Arguments
void	AMCLIB_BemfObsrvDQInit_F16	AMCLIB_BEMF_OBSRV_DQ_T_F16 *const pCtrl
void	AMCLIB_BemfObsrvDQInit_F32	AMCLIB_BEMF_OBSRV_DQ_T_F32 *const pCtrl
void	AMCLIB_BemfObsrvDQInit_FLT	AMCLIB_BEMF_OBSRV_DQ_T_FLT *const pCtrl

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
void	AMCLIB_BemfObsrvDQSetState_F16	const SWLIBS_2Syst_F16 *const pIAB const SWLIBS_2Syst_F16 *const pUAB tFrac16 f16Velocity tFrac16 f16Phase AMCLIB_BEMF_OBSRV_DQ_T_F16 *const pCtrl
void	AMCLIB_BemfObsrvDQSetState_F32	const SWLIBS_2Syst_F32 *const pIAB const SWLIBS_2Syst_F32 *const pUAB tFrac32 f32Velocity tFrac32 f32Phase AMCLIB_BEMF_OBSRV_DQ_T_F32 *const pCtrl
void	AMCLIB_BemfObsrvDQSetState_FLT	const SWLIBS_2Syst_FLT *const pIAB const SWLIBS_2Syst_FLT *const pUAB tFloat fltVelocity tFloat fltPhase AMCLIB_BEMF_OBSRV_DQ_T_FLT *const pCtrl
tFrac16	AMCLIB_BemfObsrvDQ_F16	const SWLIBS_2Syst_F16 *const pIAB const SWLIBS_2Syst_F16 *const pUAB tFrac16 f16Velocity tFrac16 f16Phase AMCLIB_BEMF_OBSRV_DQ_T_F16 *const pCtrl
tFrac32	AMCLIB_BemfObsrvDQ_F32	const SWLIBS_2Syst_F32 *const pIAB const SWLIBS_2Syst_F32 *const pUAB tFrac32 f32Velocity tFrac32 f32Phase AMCLIB_BEMF_OBSRV_DQ_T_F32 *const pCtrl
tFloat	AMCLIB_BemfObsrvDQ_FLT	const SWLIBS_2Syst_FLT *const pIAB const SWLIBS_2Syst_FLT *const pUAB tFloat fltVelocity tFloat fltPhase AMCLIB_BEMF_OBSRV_DQ_T_FLT *const pCtrl
void	AMCLIB_CurrentLoopInit_F16	AMCLIB_CURRENT_LOOP_T_F16 *const pCtrl
void	AMCLIB_CurrentLoopInit_F32	AMCLIB_CURRENT_LOOP_T_F32 *const pCtrl
void	AMCLIB_CurrentLoopInit_FLT	AMCLIB_CURRENT_LOOP_T_FLT *const pCtrl
void	AMCLIB_CurrentLoopPIpInit_F16	AMCLIB_CURRENT_LOOP_PIP_T_F16 *const pCtrl
void	AMCLIB_CurrentLoopPIpInit_F32	AMCLIB_CURRENT_LOOP_PIP_T_F32 *const pCtrl

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
void	AMCLIB_CurrentLoopPIpInit_FLT	AMCLIB_CURRENT_LOOP_PIP_T_FLT *const pCtrl
void	AMCLIB_CurrentLoopPIpSetState_F16	f32ControllerPIpAWDOut f32ControllerPIpAWQOut AMCLIB_CURRENT_LOOP_PIP_T_F16 * pCtrl
void	AMCLIB_CurrentLoopPIpSetState_F32	tFrac32 f32ControllerPIpAWDOut tFrac32 f32ControllerPIpAWQOut AMCLIB_CURRENT_LOOP_PIP_T_F32 * pCtrl
void	AMCLIB_CurrentLoopPIpSetState_FLT	tFloat fltControllerPIpAWDOut tFloat fltControllerPIpAWQOut AMCLIB_CURRENT_LOOP_PIP_T_FLT * pCtrl
void	AMCLIB_CurrentLoopPIp_F16	tFrac16 f16UDcBus SWLIBS_2Syst_F16 *const pUDQReq AMCLIB_CURRENT_LOOP_PIP_T_F16 * pCtrl
void	AMCLIB_CurrentLoopPIp_F32	tFrac32 f32UDcBus SWLIBS_2Syst_F32 *const pUDQReq AMCLIB_CURRENT_LOOP_PIP_T_F32 * pCtrl
void	AMCLIB_CurrentLoopPIp_FLT	tFloat fltUDcBus SWLIBS_2Syst_FLT *const pUDQReq AMCLIB_CURRENT_LOOP_PIP_T_FLT * pCtrl
void	AMCLIB_CurrentLoopSetState_F16	tFrac16 f16ControllerPIrAWDOut tFrac16 f16ControllerPIrAWQOut AMCLIB_CURRENT_LOOP_T_F16 * pCtrl
void	AMCLIB_CurrentLoopSetState_F32	tFrac32 f32ControllerPIrAWDOut tFrac32 f32ControllerPIrAWQOut AMCLIB_CURRENT_LOOP_T_F32 * pCtrl
void	AMCLIB_CurrentLoopSetState_FLT	tFloat fltControllerPIrAWDOut tFloat fltControllerPIrAWQOut AMCLIB_CURRENT_LOOP_T_FLT * pCtrl
void	AMCLIB_CurrentLoop_F16	tFrac16 f16UDcBus SWLIBS_2Syst_F16 *const pUDQReq AMCLIB_CURRENT_LOOP_T_F16 * pCtrl
void	AMCLIB_CurrentLoop_F32	tFrac32 f32UDcBus SWLIBS_2Syst_F32 *const pUDQReq AMCLIB_CURRENT_LOOP_T_F32 * pCtrl

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
void	AMCLIB_CurrentLoop_FLT	tFloat fltUDcBus SWLIBS_2Syst_FLT *const pUDQReq AMCLIB_CURRENT_LOOP_T_FLT * pCtrl
AMCLIB_ESTIMBJINIT_RET_T	AMCLIB_EstimBJInit_FLT	AMCLIB_ESTIMBJ_T_FLT *const pCtrl AMCLIB_ESTIMBJ_INIT_T_FLT pParam
AMCLIB_ESTIMBJ_RET_T	AMCLIB_EstimBJ_FLT	SWLIBS_2Syst_FLT *const pUAlBeReq const SWLIBS_2Syst_FLT *const pIAlBeFbck tFloat fltUDcBus AMCLIB_ESTIMBJ_T_FLT *const pCtrl
AMCLIB_ESTIMRLINIT_RET_T	AMCLIB_EstimRLInit_FLT	tU32 u32SamplingFreq AMCLIB_ESTIMRL_T_FLT *const pCtrl AMCLIB_ESTIMRL_INIT_T_FLT pParam
AMCLIB_ESTIMRL_RET_T	AMCLIB_EstimRL_FLT	SWLIBS_2Syst_FLT *const pUAlBeReq const SWLIBS_2Syst_FLT *const pIAlBeFbck tFloat fltUDcBus AMCLIB_ESTIMRL_T_FLT *const pCtrl AMCLIB_ESTIMRL_RUN_T_FLT pParam
void	AMCLIB_FWDebug_F16	tFrac16 f16IDQReqAmp tFrac16 f16VelocityFbck SWLIBS_2Syst_F16 *const pIDQReq AMCLIB_FW_DEBUG_T_F16 * pCtrl
void	AMCLIB_FWDebug_F32	tFrac32 f32IDQReqAmp tFrac32 f32VelocityFbck SWLIBS_2Syst_F32 *const pIDQReq AMCLIB_FW_DEBUG_T_F32 * pCtrl
void	AMCLIB_FWDebug_FLT	tFloat fltIDQReqAmp tFloat fltVelocityFbck SWLIBS_2Syst_FLT *const pIDQReq AMCLIB_FW_DEBUG_T_FLT * pCtrl
void	AMCLIB_FWInit_F16	AMCLIB_FW_T_F16 *const pCtrl
void	AMCLIB_FWInit_F32	AMCLIB_FW_T_F32 *const pCtrl
void	AMCLIB_FWInit_FLT	AMCLIB_FW_T_FLT *const pCtrl
void	AMCLIB_FWSetState_F16	tFrac16 f16FilterMAFWOut tFrac16 f16ControllerPIpAWFWOut AMCLIB_FW_T_F16 * pCtrl
void	AMCLIB_FWSetState_F32	tFrac32 f32FilterMAFWOut tFrac32 f32ControllerPIpAWFWOut AMCLIB_FW_T_F32 * pCtrl
void	AMCLIB_FWSetState_FLT	tFloat fltFilterMAFWOut tFloat fltControllerPIpAWFWOut AMCLIB_FW_T_FLT * pCtrl

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
void	AMCLIB_FWSpeedLoopDebug_F16	tFrac16 f16VelocityReq tFrac16 f16VelocityFbck SWLIBS_2Syst_F16 *const pIDQReq AMCLIB_FW_SPEED_LOOP_DEBUG_T_F16 * pCtrl
void	AMCLIB_FWSpeedLoopDebug_F32	tFrac32 f32VelocityReq tFrac32 f32VelocityFbck SWLIBS_2Syst_F32 *const pIDQReq AMCLIB_FW_SPEED_LOOP_DEBUG_T_F32 * pCtrl
void	AMCLIB_FWSpeedLoopDebug_FLT	tFloat fltVelocityReq tFloat fltVelocityFbck SWLIBS_2Syst_FLT *const pIDQReq AMCLIB_FW_SPEED_LOOP_DEBUG_T_FLT * pCtrl
void	AMCLIB_FWSpeedLoopInit_F16	AMCLIB_FW_SPEED_LOOP_T_F16 *const pCtrl
void	AMCLIB_FWSpeedLoopInit_F32	AMCLIB_FW_SPEED_LOOP_T_F32 *const pCtrl
void	AMCLIB_FWSpeedLoopInit_FLT	AMCLIB_FW_SPEED_LOOP_T_FLT *const pCtrl
void	AMCLIB_FWSpeedLoopSetState_F16	tFrac16 f16FilterMAWOut tFrac16 f16FilterMAFWOut tFrac16 f16ControllerPIpAWQOut tFrac16 f16ControllerPIpAWFWOut tFrac32 f32RampOut AMCLIB_FW_SPEED_LOOP_T_F16 * pCtrl
void	AMCLIB_FWSpeedLoopSetState_F32	tFrac32 f32FilterMAWOut tFrac32 f32FilterMAFWOut tFrac32 f32ControllerPIpAWQOut tFrac32 f32ControllerPIpAWFWOut tFrac32 f32RampOut AMCLIB_FW_SPEED_LOOP_T_F32 * pCtrl
void	AMCLIB_FWSpeedLoopSetState_FLT	tFloat fltFilterMAWOut tFloat fltFilterMAFWOut tFloat fltControllerPIpAWQOut tFloat fltControllerPIpAWFWOut tFloat fltRampOut AMCLIB_FW_SPEED_LOOP_T_FLT * pCtrl
void	AMCLIB_FWSpeedLoop_F16	tFrac16 f16VelocityReq tFrac16 f16VelocityFbck SWLIBS_2Syst_F16 *const pIDQReq AMCLIB_FW_SPEED_LOOP_T_F16 * pCtrl

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
void	AMCLIB_FWSpeedLoop_F32	tFrac32 f32VelocityReq tFrac32 f32VelocityFbck SWLIBS_2Syst_F32 *const pIDQReq AMCLIB_FW_SPEED_LOOP_T_F32 * pCtrl
void	AMCLIB_FWSpeedLoop_FLT	tFloat fltVelocityReq tFloat fltVelocityFbck SWLIBS_2Syst_FLT *const pIDQReq AMCLIB_FW_SPEED_LOOP_T_FLT * pCtrl
void	AMCLIB_FW_F16	tFrac16 f16IDQReqAmp tFrac16 f16VelocityFbck SWLIBS_2Syst_F16 *const pIDQReq AMCLIB_FW_T_F16 * pCtrl
void	AMCLIB_FW_F32	tFrac32 f32IDQReqAmp tFrac32 f32VelocityFbck SWLIBS_2Syst_F32 *const pIDQReq AMCLIB_FW_T_F32 * pCtrl
void	AMCLIB_FW_FLT	tFloat fltIDQReqAmp tFloat fltVelocityFbck SWLIBS_2Syst_FLT *const pIDQReq AMCLIB_FW_T_FLT * pCtrl
tFrac16	AMCLIB_FluxEstimAB_F16	tFrac16 f16Velocity const SWLIBS_2Syst_F16 *const pIAB const SWLIBS_2Syst_F16 *const pUAB const AMCLIB_FLUXESTIMAB_T_F16 *const pParam
tFrac32	AMCLIB_FluxEstimAB_F32	tFrac32 f32Velocity const SWLIBS_2Syst_F32 *const pIAB const SWLIBS_2Syst_F32 *const pUAB const AMCLIB_FLUXESTIMAB_T_F32 *const pParam
tFloat	AMCLIB_FluxEstimAB_FLT	tFloat fltVelocity const SWLIBS_2Syst_FLT *const pIAB const SWLIBS_2Syst_FLT *const pUAB const AMCLIB_FLUXESTIMAB_T_FLT *const pParam
void	AMCLIB_SpeedLoopDebug_F16	tFrac16 f16VelocityReq tFrac16 f16VelocityFbck SWLIBS_2Syst_F16 *const pIDQReq AMCLIB_SPEED_LOOP_DEBUG_T_F16 * pCtrl
void	AMCLIB_SpeedLoopDebug_F32	tFrac32 f32VelocityReq tFrac32 f32VelocityFbck SWLIBS_2Syst_F32 *const pIDQReq AMCLIB_SPEED_LOOP_DEBUG_T_F32 * pCtrl

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
void	AMCLIB_SpeedLoopDebug_FLT	tFloat fltVelocityReq tFloat fltVelocityFbck SWLIBS_2Syst_FLT *const pIDQReq AMCLIB_SPEED_LOOP_DEBUG_T_FLT * pCtrl
void	AMCLIB_SpeedLoopInit_F16	AMCLIB_SPEED_LOOP_T_F16 *const pCtrl
void	AMCLIB_SpeedLoopInit_F32	AMCLIB_SPEED_LOOP_T_F32 *const pCtrl
void	AMCLIB_SpeedLoopInit_FLT	AMCLIB_SPEED_LOOP_T_FLT *const pCtrl
void	AMCLIB_SpeedLoopSetState_F16	tFrac16 f16FilterMAWOut tFrac16 f16ControllerPIpAWQOut tFrac32 f32RampOut AMCLIB_SPEED_LOOP_T_F16 * pCtrl
void	AMCLIB_SpeedLoopSetState_F32	tFrac32 f32FilterMAWOut tFrac32 f32ControllerPIpAWQOut tFrac32 f32RampOut AMCLIB_SPEED_LOOP_T_F32 * pCtrl
void	AMCLIB_SpeedLoopSetState_FLT	tFloat fltFilterMAWOut tFloat fltControllerPIpAWQOut tFloat fltRampOut AMCLIB_SPEED_LOOP_T_FLT * pCtrl
void	AMCLIB_SpeedLoop_F16	tFrac16 f16VelocityReq tFrac16 f16VelocityFbck SWLIBS_2Syst_F16 *const pIDQReq AMCLIB_SPEED_LOOP_T_F16 * pCtrl
void	AMCLIB_SpeedLoop_F32	tFrac32 f32VelocityReq tFrac32 f32VelocityFbck SWLIBS_2Syst_F32 *const pIDQReq AMCLIB_SPEED_LOOP_T_F32 * pCtrl
void	AMCLIB_SpeedLoop_FLT	tFloat fltVelocityReq tFloat fltVelocityFbck SWLIBS_2Syst_FLT *const pIDQReq AMCLIB_SPEED_LOOP_T_FLT * pCtrl
void	AMCLIB_TrackObsrvInit_F16	AMCLIB_TRACK_OBSRV_T_F16 * pCtrl
void	AMCLIB_TrackObsrvInit_F32	AMCLIB_TRACK_OBSRV_T_F32 * pCtrl
void	AMCLIB_TrackObsrvInit_FLT	AMCLIB_TRACK_OBSRV_T_FLT * pCtrl
void	AMCLIB_TrackObsrvSetState_F16	tFrac16 f16PosOut tFrac16 f16VelocityOut AMCLIB_TRACK_OBSRV_T_F16 * pCtrl
void	AMCLIB_TrackObsrvSetState_F32	tFrac32 f32PosOut tFrac32 f32VelocityOut AMCLIB_TRACK_OBSRV_T_F32 * pCtrl

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
void	AMCLIB_TrackObsrvSetState_FLT	tFloat fltPosOut tFloat fltVelocityOut AMCLIB_TRACK_OBSRV_T_FLT * pCtrl
void	AMCLIB_TrackObsrv_F16	tFrac16 f16PhaseErr tFrac16 * pPosEst tFrac16 * pVelocityEst AMCLIB_TRACK_OBSRV_T_F16 * pCtrl
void	AMCLIB_TrackObsrv_F32	tFrac32 f32PhaseErr tFrac32 * pPosEst tFrac32 * pVelocityEst AMCLIB_TRACK_OBSRV_T_F32 * pCtrl
void	AMCLIB_TrackObsrv_FLT	tFloat fltPhaseErr tFloat * pPosEst tFloat * pVelocityEst AMCLIB_TRACK_OBSRV_T_FLT * pCtrl
void	AMCLIB_WindmillingInit_F16	tFrac16 f16ADCMaxError AMCLIB_WINDMILLING_T_F16 *const pCtrl
void	AMCLIB_WindmillingInit_F32	tFrac32 f32ADCMaxError AMCLIB_WINDMILLING_T_F32 *const pCtrl
void	AMCLIB_WindmillingInit_FLT	tFloat fltADCMaxError AMCLIB_WINDMILLING_T_FLT *const pCtrl
AMCLIB_WINDMILLING_RET_T	AMCLIB_Windmilling_F16	const SWLIBS_3Syst_F16 * pUabcIn tFrac16 * pPosEst tFrac16 * pVelocityEst AMCLIB_WINDMILLING_T_F16 *const pCtrl
AMCLIB_WINDMILLING_RET_T	AMCLIB_Windmilling_F32	const SWLIBS_3Syst_F32 * pUabcIn tFrac32 * pPosEst tFrac32 * pVelocityEst AMCLIB_WINDMILLING_T_F32 *const pCtrl
AMCLIB_WINDMILLING_RET_T	AMCLIB_Windmilling_FLT	const SWLIBS_3Syst_FLT * pUabcIn tFloat * pPosEst tFloat * pVelocityEst AMCLIB_WINDMILLING_T_FLT *const pCtrl
void	GDFLIB_FilterFIRInit_F16	const GDFLIB_FILTERFIR_PARAM_T_F16 *const pParam GDFLIB_FILTERFIR_STATE_T_F16 *const pState tFrac16 * pInBuf

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
void	GDFLIB_FilterFIRInit_F32	const GDFLIB_FILTERFIR_PARAM_T_F32 *const pParam GDFLIB_FILTERFIR_STATE_T_F32 *const pState tFrac32 * pInBuf
void	GDFLIB_FilterFIRInit_FLT	const GDFLIB_FILTERFIR_PARAM_T_FLT *const pParam GDFLIB_FILTERFIR_STATE_T_FLT *const pState tFloat * pInBuf
tFrac16	GDFLIB_FilterFIR_F16	tFrac16 f16In const GDFLIB_FILTERFIR_PARAM_T_F16 *const pParam GDFLIB_FILTERFIR_STATE_T_F16 *const pState
tFrac32	GDFLIB_FilterFIR_F32	tFrac32 f32In const GDFLIB_FILTERFIR_PARAM_T_F32 *const pParam GDFLIB_FILTERFIR_STATE_T_F32 *const pState
tFloat	GDFLIB_FilterFIR_FLT	tFloat fltIn const GDFLIB_FILTERFIR_PARAM_T_FLT *const pParam GDFLIB_FILTERFIR_STATE_T_FLT *const pState
void	GDFLIB_FilterIIR1Init_F16	GDFLIB_FILTER_IIR1_T_F16 *const pParam
void	GDFLIB_FilterIIR1Init_F32	GDFLIB_FILTER_IIR1_T_F32 *const pParam
void	GDFLIB_FilterIIR1Init_FLT	GDFLIB_FILTER_IIR1_T_FLT *const pParam
tFrac16	GDFLIB_FilterIIR1_F16	tFrac16 f16In GDFLIB_FILTER_IIR1_T_F16 *const pParam
tFrac32	GDFLIB_FilterIIR1_F32	tFrac32 f32In GDFLIB_FILTER_IIR1_T_F32 *const pParam
tFloat	GDFLIB_FilterIIR1_FLT	tFloat fltIn GDFLIB_FILTER_IIR1_T_FLT *const pParam
void	GDFLIB_FilterIIR2Init_F16	GDFLIB_FILTER_IIR2_T_F16 *const pParam
void	GDFLIB_FilterIIR2Init_F32	GDFLIB_FILTER_IIR2_T_F32 *const pParam
void	GDFLIB_FilterIIR2Init_FLT	GDFLIB_FILTER_IIR2_T_FLT *const pParam

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
tFrac16	GDFLIB_FilterIIR2_F16	tFrac16 f16In GDFLIB_FILTER_IIR2_T_F16 *const pParam
tFrac32	GDFLIB_FilterIIR2_F32	tFrac32 f32In GDFLIB_FILTER_IIR2_T_F32 *const pParam
tFloat	GDFLIB_FilterIIR2_FLT	tFloat fltIn GDFLIB_FILTER_IIR2_T_FLT *const pParam
void	GDFLIB_FilterMAInit_F16	GDFLIB_FILTER_MA_T_F16 * pParam
void	GDFLIB_FilterMAInit_F32	GDFLIB_FILTER_MA_T_F32 * pParam
void	GDFLIB_FilterMAInit_FLT	GDFLIB_FILTER_MA_T_FLT * pParam
void	GDFLIB_FilterMASetState_F16	tFrac16 f16FilterMAOut GDFLIB_FILTER_MA_T_F16 * pParam
void	GDFLIB_FilterMASetState_F32	tFrac32 f32FilterMAOut GDFLIB_FILTER_MA_T_F32 * pParam
void	GDFLIB_FilterMASetState_FLT	tFloat fltFilterMAOut GDFLIB_FILTER_MA_T_FLT * pParam
tFrac16	GDFLIB_FilterMA_F16	tFrac16 f16In GDFLIB_FILTER_MA_T_F16 * pParam
tFrac32	GDFLIB_FilterMA_F32	tFrac32 f32In GDFLIB_FILTER_MA_T_F32 * pParam
tFloat	GDFLIB_FilterMA_FLT	tFloat fltIn GDFLIB_FILTER_MA_T_FLT * pParam
tFrac16	GFLIB_Acos_F16	tFrac16 f16In const GFLIB_ACOS_T_F16 *const pParam
tFrac32	GFLIB_Acos_F32	tFrac32 f32In const GFLIB_ACOS_T_F32 *const pParam
tFloat	GFLIB_Acos_FLT	tFloat fltIn const GFLIB_ACOS_T_FLT *const pParam
tFrac16	GFLIB_Asin_F16	tFrac16 f16In const GFLIB_ASIN_T_F16 *const pParam
tFrac32	GFLIB_Asin_F32	tFrac32 f32In const GFLIB_ASIN_T_F32 *const pParam
tFloat	GFLIB_Asin_FLT	tFloat fltIn const GFLIB_ASIN_T_FLT *const pParam
tFrac16	GFLIB_AtanYXShifted_F16	tFrac16 f16InY tFrac16 f16InX const GFLIB_ATANYXSHIFTED_T_F16 * pParam

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
tFrac32	GFLIB_AtanYXShifted_F32	tFrac32 f32InY tFrac32 f32InX const GFLIB_ATANYXSHIFTED_T_F32 * pParam
tFloat	GFLIB_AtanYXShifted_FLT	tFloat fltInY tFloat fltInX const GFLIB_ATANYXSHIFTED_T_FLT * pParam
tFrac16	GFLIB_AtanYX_F16	tFrac16 f16InY tFrac16 f16InX
tFrac32	GFLIB_AtanYX_F32	tFrac32 f32InY tFrac32 f32InX
tFloat	GFLIB_AtanYX_FLT	tFloat fltInY tFloat fltInX
tFrac16	GFLIB_Atan_F16	tFrac16 f16In const GFLIB_ATAN_T_F16 *const pParam
tFrac32	GFLIB_Atan_F32	tFrac32 f32In const GFLIB_ATAN_T_F32 *const pParam
tFloat	GFLIB_Atan_FLT	tFloat fltIn const GFLIB_ATAN_T_FLT *const pParam
void	GFLIB_ControllerPIDpAWInit_F16	GFLIB_CONTROLLER_PID_P_AW_T_F16 *const pParam
void	GFLIB_ControllerPIDpAWInit_F32	GFLIB_CONTROLLER_PID_P_AW_T_F32 *const pParam
void	GFLIB_ControllerPIDpAWInit_FLT	GFLIB_CONTROLLER_PID_P_AW_T_FLT *const pParam
void	GFLIB_ ControllerPIDpAWSetState_F16	tFrac16 f16ControllerPIDpAWOut GFLIB_CONTROLLER_PID_P_AW_T_F16 *const pParam
void	GFLIB_ ControllerPIDpAWSetState_F32	tFrac32 f32ControllerPIDpAWOut GFLIB_CONTROLLER_PID_P_AW_T_F32 *const pParam
void	GFLIB_ ControllerPIDpAWSetState_FLT	tFloat fltControllerPIDpAWOut GFLIB_CONTROLLER_PID_P_AW_T_FLT *const pParam
tFrac16	GFLIB_ControllerPIDpAW_F16	tFrac16 f16InErr GFLIB_CONTROLLER_PID_P_AW_T_F16 *const pParam
tFrac32	GFLIB_ControllerPIDpAW_F32	tFrac32 f32InErr GFLIB_CONTROLLER_PID_P_AW_T_F32 *const pParam
tFloat	GFLIB_ControllerPIDpAW_FLT	tFloat fltInErr GFLIB_CONTROLLER_PID_P_AW_T_FLT *const pParam

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
void	GFLIB_ControllerPIpAWInit_F16	GFLIB_CONTROLLER_PIAW_P_T_F16 *const pParam
void	GFLIB_ControllerPIpAWInit_F32	GFLIB_CONTROLLER_PIAW_P_T_F32 *const pParam
void	GFLIB_ControllerPIpAWInit_FLT	GFLIB_CONTROLLER_PIAW_P_T_FLT *const pParam
void	GFLIB_ ControllerPIpAWSetState_F16	tFrac16 f16ControllerPIpAWOut GFLIB_CONTROLLER_PIAW_P_T_F16 *const pParam
void	GFLIB_ ControllerPIpAWSetState_F32	tFrac32 f32ControllerPIpAWOut GFLIB_CONTROLLER_PIAW_P_T_F32 *const pParam
void	GFLIB_ ControllerPIpAWSetState_FLT	tFloat fltControllerPIpAWOut GFLIB_CONTROLLER_PIAW_P_T_FLT *const pParam
tFrac16	GFLIB_ControllerPIpAW_F16	tFrac16 f16InErr GFLIB_CONTROLLER_PIAW_P_T_F16 *const pParam
tFrac32	GFLIB_ControllerPIpAW_F32	tFrac32 f32InErr GFLIB_CONTROLLER_PIAW_P_T_F32 *const pParam
tFloat	GFLIB_ControllerPIpAW_FLT	tFloat fltInErr GFLIB_CONTROLLER_PIAW_P_T_FLT *const pParam
void	GFLIB_ControllerPIpInit_F16	GFLIB_CONTROLLER_PI_P_T_F16 *const pParam
void	GFLIB_ControllerPIpInit_F32	GFLIB_CONTROLLER_PI_P_T_F32 *const pParam
void	GFLIB_ControllerPIpInit_FLT	GFLIB_CONTROLLER_PI_P_T_FLT *const pParam
void	GFLIB_ControllerPIpSetState_F16	tFrac16 f16ControllerPIpOut GFLIB_CONTROLLER_PI_P_T_F16 *const pParam
void	GFLIB_ControllerPIpSetState_F32	tFrac32 f32ControllerPIpOut GFLIB_CONTROLLER_PI_P_T_F32 *const pParam
void	GFLIB_ControllerPIpSetState_FLT	tFloat fltControllerPIpOut GFLIB_CONTROLLER_PI_P_T_FLT *const pParam
tFrac16	GFLIB_ControllerPIp_F16	tFrac16 f16InErr GFLIB_CONTROLLER_PI_P_T_F16 *const pParam
tFrac32	GFLIB_ControllerPIp_F32	tFrac32 f32InErr GFLIB_CONTROLLER_PI_P_T_F32 *const pParam

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
tFloat	GFLIB_ControllerPIp_FLT	tFloat fltInErr GFLIB_CONTROLLER_PI_P_T_FLT *const pParam
void	GFLIB_ControllerPIrAWInit_F16	GFLIB_CONTROLLER_PIAW_R_T_F16 *const pParam
void	GFLIB_ControllerPIrAWInit_F32	GFLIB_CONTROLLER_PIAW_R_T_F32 *const pParam
void	GFLIB_ControllerPIrAWInit_FLT	GFLIB_CONTROLLER_PIAW_R_T_FLT *const pParam
void	GFLIB_ ControllerPIrAWSetState_F16	tFrac16 f16ControllerPIrAWOut GFLIB_CONTROLLER_PIAW_R_T_F16 *const pParam
void	GFLIB_ ControllerPIrAWSetState_F32	tFrac32 f32ControllerPIrAWOut GFLIB_CONTROLLER_PIAW_R_T_F32 *const pParam
void	GFLIB_ ControllerPIrAWSetState_FLT	tFloat fltControllerPIrAWOut GFLIB_CONTROLLER_PIAW_R_T_FLT *const pParam
tFrac16	GFLIB_ControllerPIrAW_F16	tFrac16 f16InErr GFLIB_CONTROLLER_PIAW_R_T_F16 *const pParam
tFrac32	GFLIB_ControllerPIrAW_F32	tFrac32 f32InErr GFLIB_CONTROLLER_PIAW_R_T_F32 *const pParam
tFloat	GFLIB_ControllerPIrAW_FLT	tFloat fltInErr GFLIB_CONTROLLER_PIAW_R_T_FLT *const pParam
void	GFLIB_ControllerPIrInit_F16	GFLIB_CONTROLLER_PI_R_T_F16 *const pParam
void	GFLIB_ControllerPIrInit_F32	GFLIB_CONTROLLER_PI_R_T_F32 *const pParam
void	GFLIB_ControllerPIrInit_FLT	GFLIB_CONTROLLER_PI_R_T_FLT *const pParam
void	GFLIB_ControllerPIrSetState_F16	tFrac16 f16ControllerPIrOut GFLIB_CONTROLLER_PI_R_T_F16 *const pParam
void	GFLIB_ControllerPIrSetState_F32	tFrac32 f32ControllerPIrOut GFLIB_CONTROLLER_PI_R_T_F32 *const pParam
void	GFLIB_ControllerPIrSetState_FLT	tFloat fltControllerPIrOut GFLIB_CONTROLLER_PI_R_T_FLT *const pParam
tFrac16	GFLIB_ControllerPIr_F16	tFrac16 f16InErr GFLIB_CONTROLLER_PI_R_T_F16 *const pParam

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
tFrac32	GFLIB_ControllerPIr_F32	tFrac32 f32InErr GFLIB_CONTROLLER_PI_R_T_F32 *const pParam
tFloat	GFLIB_ControllerPIr_FLT	tFloat fltInErr GFLIB_CONTROLLER_PI_R_T_FLT *const pParam
tFrac16	GFLIB_Cos_F16	tFrac16 f16In const GFLIB_COS_T_F16 *const pParam
tFrac32	GFLIB_Cos_F32	tFrac32 f32In const GFLIB_COS_T_F32 *const pParam
tFloat	GFLIB_Cos_FLT	tFloat fltIn const GFLIB_COS_T_FLT *const pParam
tFrac16	GFLIB_Hyst_F16	tFrac16 f16In GFLIB_HYST_T_F16 *const pParam
tFrac32	GFLIB_Hyst_F32	tFrac32 f32In GFLIB_HYST_T_F32 *const pParam
tFloat	GFLIB_Hyst_FLT	tFloat fltIn GFLIB_HYST_T_FLT *const pParam
void	GFLIB_IntegratorTRSetState_F16	tFrac16 f16IntegratorTROut GFLIB_INTEGRATOR_TR_T_F16 *const pParam
void	GFLIB_IntegratorTRSetState_F32	tFrac32 f32IntegratorTROut GFLIB_INTEGRATOR_TR_T_F32 *const pParam
void	GFLIB_IntegratorTRSetState_FLT	tFloat fltIntegratorTROut GFLIB_INTEGRATOR_TR_T_FLT *const pParam
tFrac16	GFLIB_IntegratorTR_F16	tFrac16 f16In GFLIB_INTEGRATOR_TR_T_F16 *const pParam
tFrac32	GFLIB_IntegratorTR_F32	tFrac32 f32In GFLIB_INTEGRATOR_TR_T_F32 *const pParam
tFloat	GFLIB_IntegratorTR_FLT	tFloat fltIn GFLIB_INTEGRATOR_TR_T_FLT *const pParam
tFrac16	GFLIB_Limit_F16	tFrac16 f16In const GFLIB_LIMIT_T_F16 *const pParam
tFrac32	GFLIB_Limit_F32	tFrac32 f32In const GFLIB_LIMIT_T_F32 *const pParam
tFloat	GFLIB_Limit_FLT	tFloat fltIn const GFLIB_LIMIT_T_FLT *const pParam

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
tFloat	GFLIB_Log10_FLT	tFloat fltIn const GFLIB_LOG10_T_FLT *const pParam
tFrac16	GFLIB_LowerLimit_F16	tFrac16 f16In const GFLIB_LOWERLIMIT_T_F16 *const pParam
tFrac32	GFLIB_LowerLimit_F32	tFrac32 f32In const GFLIB_LOWERLIMIT_T_F32 *const pParam
tFloat	GFLIB_LowerLimit_FLT	tFloat fltIn const GFLIB_LOWERLIMIT_T_FLT *const pParam
tFrac16	GFLIB_Lut1D_F16	tFrac16 f16In const GFLIB_LUT1D_T_F16 *const pParam
tFrac32	GFLIB_Lut1D_F32	tFrac32 f32In const GFLIB_LUT1D_T_F32 *const pParam
tFloat	GFLIB_Lut1D_FLT	tFloat fltIn const GFLIB_LUT1D_T_FLT *const pParam
tFrac16	GFLIB_Lut2D_F16	tFrac16 f16In1 tFrac16 f16In2 const GFLIB_LUT2D_T_F16 *const pParam
tFrac32	GFLIB_Lut2D_F32	tFrac32 f32In1 tFrac32 f32In2 const GFLIB_LUT2D_T_F32 *const pParam
tFloat	GFLIB_Lut2D_FLT	tFloat fltIn1 tFloat fltIn2 const GFLIB_LUT2D_T_FLT *const pParam
tFrac16	GFLIB_Ramp_F16	tFrac16 f16In GFLIB_RAMP_T_F16 *const pParam
tFrac32	GFLIB_Ramp_F32	tFrac32 f32In GFLIB_RAMP_T_F32 *const pParam
tFloat	GFLIB_Ramp_FLT	tFloat fltIn GFLIB_RAMP_T_FLT *const pParam
tFrac16	GFLIB_Sign_F16	tFrac16 f16In
tFrac32	GFLIB_Sign_F32	tFrac32 f32In
tFloat	GFLIB_Sign_FLT	tFloat fltIn

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
void	GFLIB_SinCos_F16	tFrac16 f16In SWLIBS_2Syst_F16 * pOut const GFLIB_SINCOS_T_F16 *const pParam
void	GFLIB_SinCos_F32	tFrac32 f32In SWLIBS_2Syst_F32 * pOut const GFLIB_SINCOS_T_F32 *const pParam
void	GFLIB_SinCos_FLT	tFloat fltIn SWLIBS_2Syst_FLT * pOut const GFLIB_SINCOS_T_FLT *const pParam
tFrac16	GFLIB_Sin_F16	tFrac16 f16In const GFLIB_SIN_T_F16 *const pParam
tFrac32	GFLIB_Sin_F32	tFrac32 f32In const GFLIB_SIN_T_F32 *const pParam
tFloat	GFLIB_Sin_FLT	tFloat fltIn const GFLIB_SIN_T_FLT *const pParam
tFrac16	GFLIB_Sqrt_F16	tFrac16 f16In
tFrac32	GFLIB_Sqrt_F32	tFrac32 f32In
tFloat	GFLIB_Sqrt_FLT	tFloat fltIn
tFrac16	GFLIB_Tan_F16	tFrac16 f16In const GFLIB_TAN_T_F16 *const pParam
tFrac32	GFLIB_Tan_F32	tFrac32 f32In const GFLIB_TAN_T_F32 *const pParam
tFloat	GFLIB_Tan_FLT	tFloat fltIn const GFLIB_TAN_T_FLT *const pParam
tFrac16	GFLIB_UpperLimit_F16	tFrac16 f16In const GFLIB_UPPERLIMIT_T_F16 *const pParam
tFrac32	GFLIB_UpperLimit_F32	tFrac32 f32In const GFLIB_UPPERLIMIT_T_F32 *const pParam
tFloat	GFLIB_UpperLimit_FLT	tFloat fltIn const GFLIB_UPPERLIMIT_T_FLT *const pParam
void	GFLIB_VLog10_FLT	tFloat * pInOut tU32 u32N const GFLIB_VLOG10_T_FLT *const pParam
INLINE tU16	GFLIB_VMin10_F16	const tFrac16 * pIn
INLINE tU16	GFLIB_VMin11_F16	const tFrac16 * pIn
INLINE tU16	GFLIB_VMin12_F16	const tFrac16 * pIn

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
INLINE tU16	GFLIB_VMin13_F16	const tFrac16 * pIn
INLINE tU16	GFLIB_VMin14_F16	const tFrac16 * pIn
INLINE tU16	GFLIB_VMin15_F16	const tFrac16 * pIn
INLINE tU16	GFLIB_VMin16_F16	const tFrac16 * pIn
INLINE tU16	GFLIB_VMin4_F16	const tFrac16 * pIn
INLINE tU16	GFLIB_VMin5_F16	const tFrac16 * pIn
INLINE tU16	GFLIB_VMin6_F16	const tFrac16 * pIn
INLINE tU16	GFLIB_VMin7_F16	const tFrac16 * pIn
INLINE tU16	GFLIB_VMin8_F16	const tFrac16 * pIn
INLINE tU16	GFLIB_VMin9_F16	const tFrac16 * pIn
tU16	GFLIB_VMin_F16	const tFrac16 * pIn tU16 u16N
tU32	GFLIB_VMin_F32	const tFrac32 * pIn tU32 u32N
tU32	GFLIB_VMin_FLT	const tFloat * pIn tU32 u32N
tBool	GFLIB_VectorLimit_F16	SWLIBS_2Syst_F16 *const pOut const SWLIBS_2Syst_F16 *const pIn const GFLIB_VECTORLIMIT_T_F16 *const pParam
tBool	GFLIB_VectorLimit_F32	SWLIBS_2Syst_F32 *const pOut const SWLIBS_2Syst_F32 *const pIn const GFLIB_VECTORLIMIT_T_F32 *const pParam
tBool	GFLIB_VectorLimit_FLT	SWLIBS_2Syst_FLT *const pOut const SWLIBS_2Syst_FLT *const pIn const GFLIB_VECTORLIMIT_T_FLT *const pParam
void	GMCLIB_BetaProjection3Ph_F16	const SWLIBS_3Syst_F16 *const pIn SWLIBS_3Syst_F16 *const pOut
void	GMCLIB_BetaProjection3Ph_F32	const SWLIBS_3Syst_F32 *const pIn SWLIBS_3Syst_F32 *const pOut
void	GMCLIB_BetaProjection3Ph_FLT	const SWLIBS_3Syst_FLT *const pIn SWLIBS_3Syst_FLT *const pOut
void	GMCLIB_BetaProjection_F16	const SWLIBS_3Syst_F16 *const pIn SWLIBS_2Syst_F16 *const pOut
void	GMCLIB_BetaProjection_F32	const SWLIBS_3Syst_F32 *const pIn SWLIBS_2Syst_F32 *const pOut
void	GMCLIB_BetaProjection_FLT	const SWLIBS_3Syst_FLT *const pIn SWLIBS_2Syst_FLT *const pOut

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
void	GMCLIB_Clark6PhInit_F16	tFrac16 f16WindingShift GMCLIB_CLARK_6PH_T_F16 *const pParam
void	GMCLIB_Clark6PhInit_F32	tFrac32 f32WindingShift GMCLIB_CLARK_6PH_T_F32 *const pParam
void	GMCLIB_Clark6PhInit_FLT	tFloat fltWindingShift GMCLIB_CLARK_6PH_T_FLT *const pParam
void	GMCLIB_Clark6Ph_F16	const SWLIBS_6Syst_F16 *const pIn SWLIBS_2Syst_F16 *const pOut const GMCLIB_CLARK_6PH_T_F16 *const pParam
void	GMCLIB_Clark6Ph_F32	SWLIBS_2Syst_F32 *const pOut const SWLIBS_6Syst_F32 *const pIn const GMCLIB_CLARK_6PH_T_F32 *const pParam
void	GMCLIB_Clark6Ph_FLT	const SWLIBS_6Syst_FLT *const pIn SWLIBS_2Syst_FLT *const pOut const GMCLIB_CLARK_6PH_T_FLT *const pParam
void	GMCLIB_ClarkInv6PhInit_F16	tFrac16 f16WindingShift GMCLIB_CLARKINV_6PH_T_F16 *const pParam
void	GMCLIB_ClarkInv6PhInit_F32	tFrac32 f32WindingShift GMCLIB_CLARKINV_6PH_T_F32 *const pParam
void	GMCLIB_ClarkInv6PhInit_FLT	tFloat fltWindingShift GMCLIB_CLARKINV_6PH_T_FLT *const pParam
void	GMCLIB_ClarkInv6Ph_F16	SWLIBS_6Syst_F16 *const pOut const SWLIBS_2Syst_F16 *const pIn const GMCLIB_CLARKINV_6PH_T_F16 *const pParam
void	GMCLIB_ClarkInv6Ph_F32	SWLIBS_6Syst_F32 *const pOut const SWLIBS_2Syst_F32 *const pIn const GMCLIB_CLARKINV_6PH_T_F32 *const pParam
void	GMCLIB_ClarkInv6Ph_FLT	const SWLIBS_2Syst_FLT *const pIn const GMCLIB_CLARKINV_6PH_T_FLT *const pParam
void	GMCLIB_ClarkInv_F16	const SWLIBS_2Syst_F16 *const pIn SWLIBS_3Syst_F16 *const pOut
void	GMCLIB_ClarkInv_F32	const SWLIBS_2Syst_F32 *const pIn SWLIBS_3Syst_F32 *const pOut

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
void	GMCLIB_ClarkInv_FLT	const SWLIBS_2Syst_FLT *const pIn SWLIBS_3Syst_FLT *const pOut
void	GMCLIB_Clark_F16	const SWLIBS_3Syst_F16 *const pIn SWLIBS_2Syst_F16 *const pOut
void	GMCLIB_Clark_F32	const SWLIBS_3Syst_F32 *const pIn SWLIBS_2Syst_F32 *const pOut
void	GMCLIB_Clark_FLT	const SWLIBS_3Syst_FLT *const pIn SWLIBS_2Syst_FLT *const pOut
void	GMCLIB_DecouplingPMSM_F16	SWLIBS_2Syst_F16 *const pUdqDec const SWLIBS_2Syst_F16 *const pUdq const SWLIBS_2Syst_F16 *const pIdq tFrac16 f16AngularVel const GMCLIB_DECOUPLINGPMSM_T_F16 *const pParam
void	GMCLIB_DecouplingPMSM_F32	SWLIBS_2Syst_F32 *const pUdqDec const SWLIBS_2Syst_F32 *const pUdq const SWLIBS_2Syst_F32 *const pIdq tFrac32 f32AngularVel const GMCLIB_DECOUPLINGPMSM_T_F32 *const pParam
void	GMCLIB_DecouplingPMSM_FLT	SWLIBS_2Syst_FLT *const pUdqDec const SWLIBS_2Syst_FLT *const pUdq const SWLIBS_2Syst_FLT *const pIdq tFloat fltAngularVel const GMCLIB_DECOUPLINGPMSM_T_FLT *const pParam
void	GMCLIB_DoubleSwitching_F16	const SWLIBS_3Syst_F16 *const pInPwm tU16 u16Sector const GMCLIB_DOUBLESWITCHING_T_F16 *const pParam GMCLIB_DOUBLESWITCHING_3PH_EDGES_T_F16 *const pOutEdges GMCLIB_DOUBLESWITCHING_TRIGGERS_T_F16 *const pOutTriggers
void	GMCLIB_ElimDcBusRip_F16	SWLIBS_2Syst_F16 *const pOut const SWLIBS_2Syst_F16 *const pIn const GMCLIB_ELIMDCBUSRIP_T_F16 *const pParam
void	GMCLIB_ElimDcBusRip_F32	SWLIBS_2Syst_F32 *const pOut const SWLIBS_2Syst_F32 *const pIn const GMCLIB_ELIMDCBUSRIP_T_F32 *const pParam
void	GMCLIB_ElimDcBusRip_FLT	SWLIBS_2Syst_FLT *const pOut const SWLIBS_2Syst_FLT *const pIn const GMCLIB_ELIMDCBUSRIP_T_FLT *const pParam

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
void	GMCLIB_ParkInv_F16	SWLIBS_2Syst_F16 *const pOut const SWLIBS_2Syst_F16 *const pInAngle const SWLIBS_2Syst_F16 *const pIn
void	GMCLIB_ParkInv_F32	SWLIBS_2Syst_F32 *const pOut const SWLIBS_2Syst_F32 *const pInAngle const SWLIBS_2Syst_F32 *const pIn
void	GMCLIB_ParkInv_FLT	SWLIBS_2Syst_FLT *const pOut const SWLIBS_2Syst_FLT *const pInAngle const SWLIBS_2Syst_FLT *const pIn
void	GMCLIB_Park_F16	SWLIBS_2Syst_F16 * pOut const SWLIBS_2Syst_F16 *const pInAngle const SWLIBS_2Syst_F16 *const pIn
void	GMCLIB_Park_F32	SWLIBS_2Syst_F32 * pOut const SWLIBS_2Syst_F32 *const pInAngle const SWLIBS_2Syst_F32 *const pIn
void	GMCLIB_Park_FLT	SWLIBS_2Syst_FLT * pOut const SWLIBS_2Syst_FLT *const pInAngle const SWLIBS_2Syst_FLT *const pIn
tU16	GMCLIB_PwmIct_F16	SWLIBS_3Syst_F16 * pOut const SWLIBS_2Syst_F16 *const pIn
tU32	GMCLIB_PwmIct_F32	SWLIBS_3Syst_F32 * pOut const SWLIBS_2Syst_F32 *const pIn
tU32	GMCLIB_PwmIct_FLT	SWLIBS_3Syst_FLT * pOut const SWLIBS_2Syst_FLT *const pIn
tU16	GMCLIB_SpreadPWM	GMCLIB_SPREADPWM_T *const pParam
void	GMCLIB_SpreadPWMInit	tU32 u32PWMFreq tU16 u16Mod tU32 u32SpreadFreq tU8 u8FreqDev GMCLIB_SPREADPWM_T *const pParam
tU16	GMCLIB_SvmSci_F16	SWLIBS_3Syst_F16 * pOut const SWLIBS_2Syst_F16 *const pIn
tU32	GMCLIB_SvmSci_F32	SWLIBS_3Syst_F32 * pOut const SWLIBS_2Syst_F32 *const pIn
tU32	GMCLIB_SvmSci_FLT	SWLIBS_3Syst_FLT * pOut const SWLIBS_2Syst_FLT *const pIn
void	GMCLIB_SvmStd6PhInit_F16	tFrac16 f16WindingShift GMCLIB_SVMSTD_6PH_T_F16 *const pParam

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
void	GMCLIB_SvmStd6PhInit_F32	tFrac32 f32WindingShift GMCLIB_SVMSTD_6PH_T_F32 *const pParam
void	GMCLIB_SvmStd6PhInit_FLT	tFloat fltWindingShift GMCLIB_SVMSTD_6PH_T_FLT *const pParam
void	GMCLIB_SvmStd6Ph_F16	pOut SWLIBS_2Syst_U16 *const pOutSector const SWLIBS_2Syst_F16 *const pIn const GMCLIB_SVMSTD_6PH_T_F16 *const pParam
void	GMCLIB_SvmStd6Ph_F32	pOut SWLIBS_2Syst_U32 *const pOutSector const SWLIBS_2Syst_F32 *const pIn const GMCLIB_SVMSTD_6PH_T_F32 *const pParam
void	GMCLIB_SvmStd6Ph_FLT	SWLIBS_2Syst_U32 *const pOutSector const SWLIBS_2Syst_FLT *const pIn const GMCLIB_SVMSTD_6PH_T_FLT *const pParam
tU16	GMCLIB_SvmStd_F16	SWLIBS_3Syst_F16 * pOut const SWLIBS_2Syst_F16 *const pIn
tU32	GMCLIB_SvmStd_F32	SWLIBS_3Syst_F32 * pOut const SWLIBS_2Syst_F32 *const pIn
tU32	GMCLIB_SvmStd_FLT	SWLIBS_3Syst_FLT * pOut const SWLIBS_2Syst_FLT *const pIn
tU16	GMCLIB_SvmU0n_F16	SWLIBS_3Syst_F16 * pOut const SWLIBS_2Syst_F16 *const pIn
tU32	GMCLIB_SvmU0n_F32	SWLIBS_3Syst_F32 * pOut const SWLIBS_2Syst_F32 *const pIn
tU32	GMCLIB_SvmU0n_FLT	SWLIBS_3Syst_FLT * pOut const SWLIBS_2Syst_FLT *const pIn
tU16	GMCLIB_SvmU7n_F16	SWLIBS_3Syst_F16 * pOut const SWLIBS_2Syst_F16 *const pIn
tU32	GMCLIB_SvmU7n_F32	SWLIBS_3Syst_F32 * pOut const SWLIBS_2Syst_F32 *const pIn
tU32	GMCLIB_SvmU7n_FLT	SWLIBS_3Syst_FLT * pOut const SWLIBS_2Syst_FLT *const pIn
void	GMCLIB_VRot_F16	SWLIBS_2Syst_F16 *const f16OutVec const SWLIBS_2Syst_F16 *const f16InVec tFrac16 f16Angle

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
void	GMCLIB_VRot_F32	SWLIBS_2Syst_F32 *const f32OutVec const SWLIBS_2Syst_F32 *const f32InVec tFrac32 f32Angle
void	GMCLIB_VRot_FLT	SWLIBS_2Syst_FLT *const fltOutVec const SWLIBS_2Syst_FLT *const fltInVec tFloat fltAngle
void	GMCLIB_VUnit_F16	SWLIBS_2Syst_F16 *const f16OutVec const SWLIBS_2Syst_F16 *const f16InVec
void	GMCLIB_VUnit_F32	SWLIBS_2Syst_F32 *const f32OutVec const SWLIBS_2Syst_F32 *const f32InVec
void	GMCLIB_VUnit_FLT	SWLIBS_2Syst_FLT *const fltOutVec const SWLIBS_2Syst_FLT *const fltInVec
tFrac16	GMCLIB_Vmag2coord_F16	tFrac16 f16InMag tFrac16 f16InCoord
tFrac32	GMCLIB_Vmag2coord_F32	tFrac32 f32InMag tFrac32 f32InCoord
tFloat	GMCLIB_Vmag2coord_FLT	f32InMag f32InCoord
INLINE tFrac16	MLIB_AbsSat_F16	register tFrac16 f16In
INLINE tFrac32	MLIB_AbsSat_F32	register tFrac32 f32In
INLINE tFrac16	MLIB_Abs_F16	register tFrac16 f16In
INLINE tFrac32	MLIB_Abs_F32	register tFrac32 f32In
INLINE tFloat	MLIB_Abs_FLT	register tFloat fltIn
INLINE tFrac16	MLIB_AddSat_F16	register tFrac16 f16In1 register tFrac16 f16In2
INLINE tFrac32	MLIB_AddSat_F32	register tFrac32 f32In1 register tFrac32 f32In2
INLINE tFrac16	MLIB_Add_F16	register tFrac16 f16In1 register tFrac16 f16In2
INLINE tFrac32	MLIB_Add_F32	register tFrac32 f32In1 register tFrac32 f32In2
INLINE tFloat	MLIB_Add_FLT	register tFloat fltIn1 register tFloat fltIn2
INLINE tFrac16	MLIB_ConvertPU_F16F32	register tFrac32 f32In
INLINE tFrac16	MLIB_ConvertPU_F16FLT	register tFloat fltIn
INLINE tFrac32	MLIB_ConvertPU_F32F16	register tFrac16 f16In
INLINE tFrac32	MLIB_ConvertPU_F32FLT	register tFloat fltIn
INLINE tFloat	MLIB_ConvertPU_FLTF16	register tFrac16 f16In

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
INLINE tFloat	MLIB_ConvertPU_FLTF32	register tFrac32 f32In
INLINE tFrac16	MLIB_Convert_F16F32	register tFrac32 f32In1 register tFrac32 f32In2
INLINE tFrac16	MLIB_Convert_F16FLT	register tFloat fltIn1 register tFloat fltIn2
INLINE tFrac32	MLIB_Convert_F32F16	register tFrac16 f16In1 register tFrac16 f16In2
INLINE tFrac32	MLIB_Convert_F32FLT	register tFloat fltIn1 register tFloat fltIn2
INLINE tFloat	MLIB_Convert_FLTF16	register tFrac16 f16In1 register tFrac16 f16In2
INLINE tFloat	MLIB_Convert_FLTF32	register tFrac32 f32In1 register tFrac32 f32In2
INLINE tFrac32	MLIB_DivSatAccurate_F32	register tFrac32 f32In1 register tFrac32 f32In2
INLINE tFrac16	MLIB_DivSat_F16	register tFrac16 f16In1 register tFrac16 f16In2
INLINE tFrac32	MLIB_DivSat_F32	register tFrac32 f32In1 register tFrac32 f32In2
INLINE tFrac16	MLIB_Div_F16	register tFrac16 f16In1 register tFrac16 f16In2
INLINE tFrac32	MLIB_Div_F32	register tFrac32 f32In1 register tFrac32 f32In2
INLINE tFloat	MLIB_Div_FLT	register tFloat fltIn1 register tFloat fltIn2
INLINE tFrac16	MLIB_MacSat_F16	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In3
INLINE tFrac32	MLIB_MacSat_F32	register tFrac32 f32In1 register tFrac32 f32In2 register tFrac32 f32In3
INLINE tFrac32	MLIB_MacSat_F32F16F16	register tFrac32 f32In1 register tFrac16 f16In2 register tFrac16 f16In3
INLINE tFrac16	MLIB_Mac_F16	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In3
INLINE tFrac32	MLIB_Mac_F32	register tFrac32 f32In1 register tFrac32 f32In2 register tFrac32 f32In3
INLINE tFrac32	MLIB_Mac_F32F16F16	register tFrac32 f32In1 register tFrac16 f16In2 register tFrac16 f16In3

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
INLINE tFloat	MLIB_Mac_FLT	register tFloat fltIn1 register tFloat fltIn2 register tFloat fltIn3
INLINE tFrac16	MLIB_Mnac_F16	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In3
INLINE tFrac32	MLIB_Mnac_F32	register tFrac32 f32In1 register tFrac32 f32In2 register tFrac32 f32In3
INLINE tFrac32	MLIB_Mnac_F32F16F16	register tFrac32 f32In1 register tFrac16 f16In2 register tFrac16 f16In3
INLINE tFloat	MLIB_Mnac_FLT	register tFloat fltIn1 register tFloat fltIn2 register tFloat fltIn3
INLINE tFrac16	MLIB_Msu_F16	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In3
INLINE tFrac32	MLIB_Msu_F32	register tFrac32 f32In1 register tFrac32 f32In2 register tFrac32 f32In3
INLINE tFrac32	MLIB_Msu_F32F16F16	register tFrac32 f32In1 register tFrac16 f16In2 register tFrac16 f16In3
INLINE tFloat	MLIB_Msu_FLT	register tFloat fltIn1 register tFloat fltIn2 register tFloat fltIn3
INLINE tFrac16	MLIB_MulSat_F16	register tFrac16 f16In1 register tFrac16 f16In2
INLINE tFrac32	MLIB_MulSat_F32	register tFrac32 f32In1 register tFrac32 f32In2
INLINE tFrac32	MLIB_MulSat_F32F16F16	register tFrac16 f16In1 register tFrac16 f16In2
INLINE tFrac16	MLIB_Mul_F16	register tFrac16 f16In1 register tFrac16 f16In2
INLINE tFrac32	MLIB_Mul_F32	register tFrac32 f32In1 register tFrac32 f32In2
INLINE tFrac32	MLIB_Mul_F32F16F16	register tFrac16 f16In1 register tFrac16 f16In2
INLINE tFloat	MLIB_Mul_FLT	register tFloat fltIn1 register tFloat fltIn2
INLINE tFrac16	MLIB_NegSat_F16	register tFrac16 f16In
INLINE tFrac32	MLIB_NegSat_F32	register tFrac32 f32In

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
INLINE tFrac16	MLIB_Neg_F16	register tFrac16 f16In
INLINE tFrac32	MLIB_Neg_F32	register tFrac32 f32In
INLINE tFloat	MLIB_Neg_FLT	register tFloat fltIn
INLINE tU16	MLIB_Norm_F16	register tFrac16 f16In
INLINE tU16	MLIB_Norm_F32	register tFrac32 f32In
INLINE tFrac16	MLIB_RndSat_F16F32	register tFrac32 f32In
INLINE tFrac16	MLIB_Round_F16	register tFrac16 f16In1 register tU16 u16In2
INLINE tFrac32	MLIB_Round_F32	register tFrac32 f32In1 register tU16 u16In2
INLINE tFrac16	MLIB_ShBiSat_F16	register tFrac16 f16In1 register tS16 s16In2
INLINE tFrac32	MLIB_ShBiSat_F32	register tFrac32 f32In1 register tS16 s16In2
INLINE tFrac16	MLIB_ShBi_F16	register tFrac16 f16In1 register tS16 s16In2
INLINE tFrac32	MLIB_ShBi_F32	register tFrac32 f32In1 register tS16 s16In2
INLINE tFrac16	MLIB_ShLSat_F16	register tFrac16 f16In1 register tU16 u16In2
INLINE tFrac32	MLIB_ShLSat_F32	register tFrac32 f32In1 register tU16 u16In2
INLINE tFrac16	MLIB_ShL_F16	register tFrac16 f16In1 register tU16 u16In2
INLINE tFrac32	MLIB_ShL_F32	register tFrac32 f32In1 register tU16 u16In2
INLINE tFrac16	MLIB_ShR_F16	register tFrac16 f16In1 register tU16 u16In2
INLINE tFrac32	MLIB_ShR_F32	register tFrac32 f32In1 register tU16 u16In2
INLINE tFrac16	MLIB_SubSat_F16	register tFrac16 f16In1 register tFrac16 f16In2
INLINE tFrac32	MLIB_SubSat_F32	register tFrac32 f32In1 register tFrac32 f32In2
INLINE tFrac16	MLIB_Sub_F16	register tFrac16 f16In1 register tFrac16 f16In2
INLINE tFrac32	MLIB_Sub_F32	register tFrac32 f32In1 register tFrac32 f32In2
INLINE tFloat	MLIB_Sub_FLT	register tFloat fltIn1 register tFloat fltIn2

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
INLINE void	MLIB_VAdd_F16	SWLIBS_2Syst_F16 *const f16Out const SWLIBS_2Syst_F16 *const f16In1 const SWLIBS_2Syst_F16 *const f16In2
INLINE void	MLIB_VAdd_F32	SWLIBS_2Syst_F32 *const f32Out const SWLIBS_2Syst_F32 *const f32In1 const SWLIBS_2Syst_F32 *const f32In2
INLINE void	MLIB_VAdd_FLT	SWLIBS_2Syst_FLT *const fltOut const SWLIBS_2Syst_FLT *const fltIn1 const SWLIBS_2Syst_FLT *const fltIn2
INLINE tFrac16	MLIB_VMac_F16	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In3 register tFrac16 f16In4
INLINE tFrac32	MLIB_VMac_F32	register tFrac32 f32In1 register tFrac32 f32In2 register tFrac32 f32In3 register tFrac32 f32In4
INLINE tFrac32	MLIB_VMac_F32F16F16	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In3 register tFrac16 f16In4
INLINE tFloat	MLIB_VMac_FLT	register tFloat fltIn1 register tFloat fltIn2 register tFloat fltIn3 register tFloat fltIn4
INLINE void	MLIB_VScale_F16	SWLIBS_2Syst_F16 *const f16OutVec const SWLIBS_2Syst_F16 *const f16InVec tFrac16 f16InScale
INLINE void	MLIB_VScale_F32	SWLIBS_2Syst_F32 *const f32OutVec const SWLIBS_2Syst_F32 *const f32InVec tFrac32 f32InScale
INLINE void	MLIB_VScale_FLT	SWLIBS_2Syst_FLT *const fltOutVec const SWLIBS_2Syst_FLT *const fltInVec tFloat fltInScale
INLINE void	MLIB_VSub_F16	SWLIBS_2Syst_F16 *const f16Out const SWLIBS_2Syst_F16 *const f16In1 const SWLIBS_2Syst_F16 *const f16In2
INLINE void	MLIB_VSub_F32	SWLIBS_2Syst_F32 *const f32Out const SWLIBS_2Syst_F32 *const f32In1 const SWLIBS_2Syst_F32 *const f32In2
INLINE void	MLIB_VSub_FLT	SWLIBS_2Syst_FLT *const fltOut const SWLIBS_2Syst_FLT *const fltIn1 const SWLIBS_2Syst_FLT *const fltIn2

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 2. Quick function reference...*continued*

Type	Name	Arguments
const SWLIBS_VERSION_T *	SWLIBS_GetVersion	void

2.2 Function AMCLIB_BemfObsrvDQ

This function calculates the algorithm of the back electromotive force observer in the rotating reference frame and returns a phase error between the real rotating reference frame and the estimated one.

2.2.1 Description

The Back Electromotive Force (BEMF) observer detects the voltages induced by the permanent magnets of a Permanent Magnet Synchronous Motor (PMSM) in a quasi-synchronous reference frame. The observed BEMF allows estimation of the motor speed and position in a sensorless motor control application with Field-Oriented Control (FOC) and it calculates the phase error Θ_{err} required by the the AMCLIB_TrackObsrv. The BEMF observer is suitable for medium to high motor speeds.

Figure 33. Principal Field Oriented Control scheme with AMCLIB_BemfObsrvDQ

The input voltages and currents are supplied in a stationary reference frame α/β. The BEMF observer transforms these quantities into a quasi-synchronous reference frame γ/δ that follows the real synchronous rotor flux frame d/q with an error θ_{err}, see Figure 34 .

© NXP B.V. 2023. All rights reserved.

Figure 34. Rotor reference frames

The BEMF observer fits the input voltages and currents to a mathematical model of the motor. The model mirrors the behavior of the PMSM; see the following equation:

$$\begin{bmatrix} u_\gamma \\ u_\delta \end{bmatrix} = \begin{bmatrix} R_s + s \cdot L_d & -\omega_{\gamma\delta} \cdot L_q \\ \omega_{\gamma\delta} \cdot L_q & R_s + s \cdot L_d \end{bmatrix} \begin{bmatrix} i_\gamma \\ i_\delta \end{bmatrix} + E_{sal} \begin{bmatrix} -\sin(\theta_{err}) \\ \cos(\theta_{err}) \end{bmatrix}$$

Equation AMCLIB_BemfObsrvDQ_Eq1

where

- R_s is the resistance of one stator phase [Ω],
- L_d and L_q are the d-axis and q-axis inductances [H],
- $\omega_{\gamma\delta}$ is the estimated electrical angular velocity of the rotor [rad/s],
- u_γ and u_δ are the estimated stator voltages [V],
- i_γ and i_δ are the estimated stator currents [A],
- E_{sal} is the saliency-based BEMF magnitude [V],
- θ_{err} is the phase error between the estimated quasi-synchronous frame γ/δ and the synchronous rotor flux frame d/q [rad],
- *s* is the Laplace-Carson differential operator.

Note that in this motor model, the voltage in the δ coordinate is calculated from the L_d inductance instead of L_q. Because of this, the response to the measurement errors of the R_s and L_d parameters is the same in both axes. The BEMF observer is formed in both of these axes and the resulting θ_{err} is extracted from the division E_γ/E_δ. Assuming sufficient motor speed, the result of the division is insensitive to the E_{sal}. This allows correct setup of the controllers.

As seen from AMCLIB_BemfObsrvDQ_Eq1 only the BEMF terms depend on the phase error θ_{err} between the quasi-synchronous reference frame γ/δ and the synchronous rotor flux frame d/q. The saliency-based BEMF term is not modeled in the stator current observer however it is estimated as a disturbance, produced by the observer PI controller. The observer is a closed loop current observer and acts as a BEMF state filter. The estimated BEMF values are used for calculating the phase error θ_{err}, which is provided as an output of the BEMF observer.

The structure of the BEMF observer is depicted in Figure 35 .

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 35. BEMF observer structure

The BEMF observer loop consists of a model of an R-L circuit which represents the motor winding and a pair of controllers with keep the motor model aligned with the real motor. Refer to the GFLIB_ControllerPIrAW function documentation for details on the implementation of the controller and its parameters. Discrete-time integrators are approximated using the trapezoidal rule. The motor model is characterized in the γ/δ frame by the following difference equations:

$$i_\gamma(n) = UGain \cdot u_\gamma(n) + WIGain \cdot \omega(n) \cdot i_\delta(n) + EGain \cdot e_\gamma(n) + IGain \cdot i_\gamma(n-1) + UGain \cdot u_\gamma(n-1) + WIGain \cdot \omega(n-1) \cdot i_\delta(n-1) + EGain \cdot e_\gamma(n-1)$$

$$i_\delta(n) = UGain \cdot u_\delta(n) - WIGain \cdot \omega(n) \cdot i_\gamma(n) + EGain \cdot e_\delta(n) + IGain \cdot i_\delta(n-1) + UGain \cdot u_\delta(n-1) - WIGain \cdot \omega(n-1) \cdot i_\gamma(n-1) + EGain \cdot e_\delta(n-1)$$

Equation AMCLIB_BemfObsrvDQ_Eq2

Before using the BEMF observer with a particular motor, the user needs to provide a set of coefficients through the `pCtrl` input pointer. The BEMF observer coefficient values can be calculated from motor parameters. A method for measuring the motor parameters is described in PMSM Electrical Parameters Measurement (document AN4680) or the library function AMCLIB_EstimRL can be used.

Refer to the following resources to find out how the NXP motor control tuning and debugging tools for NXP microcontrollers can help you deploy the AMCLIB BEMF observer in your application:

- AN4642 - Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM
- FREEMASTER - FreeMASTER Run-Time Debugging Tool

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.2.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

© NXP B.V. 2023. All rights reserved.

2.2.3 Function AMCLIB_BemfObsrvDQ_F32

2.2.3.1 Declaration

```
tFrac32 AMCLIB_BemfObsrvDQ_F32(const SWLIBS_2Syst_F32 *const
pIAB, const SWLIBS_2Syst_F32 *const pUAB, tFrac32 f32Velocity,
tFrac32 f32Phase, AMCLIB_BEMF_OBSRV_DQ_T_F32 *const pCtrl);
```

2.2.3.2 Arguments

Table 3. AMCLIB_BemfObsrvDQ_F32 arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_F32 *const	pIAB	**input**	Pointer to the structure with Alpha/Beta current components.
const SWLIBS_2Syst_F32 *const	pUAB	**input**	Pointer to the structure with Alpha/Beta voltage components.
tFrac32	f32Velocity	**input**	Estimated electrical angular velocity.
tFrac32	f32Phase	**input**	Estimated rotor flux angle.
AMCLIB_BEMF_OBSRV_DQ_T_F32 *const	pCtrl	**input, output**	Pointer to the structure with BEMF observer parameters and state.

2.2.3.3 Return

Phase error between the real rotating reference frame and the estimated one.

2.2.3.4 Implementation details

Prior to calculating the BEMF observer coefficients, it is necessary to set the scaling constants. All inputs and outputs of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation. There are two different scaling systems involved, one for the FOC part of the motor control algorithm, and another for the BEMF observer. Scaling constants must be positive values equal to or greater than the expected maxima of the corresponding physical quantities. The following scaling constants are applied to the BEMF observer coefficients:

Table 4. Scaling constants

Scaling constant	Symbol	Calculation
Maximum stator phase voltage [V]	U_{MAX}	$U_{MAX}=U_{DC_Bus_Max}$
Maximum phase current [A]	I_{MAX}	Maximum current of the inverter or nominal current of the motor (whichever is lower).
Maximum speed [rad/s]	Ω_{MAX}	Maximum application required speed, at least the motor electrical rated speed.
Maximum BEMF voltage [V]	E_{MAX}	In normal operation is equal to U_{MAX}, in case of, e.g., field weakening, might be much higher.

The BEMF observer contains two controllers (GFLIB_ControllerPIrAW_F32). The proportional gain K_P and integral gain K_I of these controllers can be calculated as follows:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$K_P = 2 \cdot \xi \cdot \omega_0 \cdot L_d - R_S$$
$$K_I = \omega_0^2 \cdot L_d$$

Equation AMCLIB_BemfObsrvDQ_F32_Eq1

where ξ is the current loop attenuation, ω_0 is the current loop natural frequency [rad/s], L_d is the d-axis stator inductance [H], and R_S is the winding resistance [ohm]. Coefficients ξ and ω_0 should correspond to the values chosen for the FOC current loop.

The continuous-time controller parameters K_P and K_I are then discretized using the bilinear transform. The resulting difference equation coefficients must be scaled so that the input and output values of the controllers fit in the fractional range.

$$CC1_{SC} = \left(K_P + \frac{K_I T_S}{2} \right) \cdot \frac{I_{MAX}}{E_{MAX}}$$
$$CC2_{SC} = \left(-K_P + \frac{K_I T_S}{2} \right) \cdot \frac{I_{MAX}}{E_{MAX}}$$

Equation AMCLIB_BemfObsrvDQ_F32_Eq2

T_S is the sampling period [s]. In case the coefficients $CC1_{SC}$ and $CC2_{SC}$ exceed the fractional range, their value must be right-shifted by *NShift*. The binary shift amount corresponds to

$$NShift = \max\left(0, \left\lceil \log_2\left(\max\left(\left|CC1_{SC}\right|, \left|CC2_{SC}\right| \right) \right) \right\rceil \right)$$

Equation AMCLIB_BemfObsrvDQ_F32_Eq3

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 31st power of two (corresponds to the Q1.31 encoding):

$$pParamD.f32CC1sc = \left(CC1_{SC} \cdot 2^{-NShift} \right) \cdot 2^{31}$$
$$pParamD.f32CC2sc = \left(CC2_{SC} \cdot 2^{-NShift} \right) \cdot 2^{31}$$
$$pParamD.u16NShift = NShift$$
$$pParamQ.f32CC1sc = \left(CC1_{SC} \cdot 2^{-NShift} \right) \cdot 2^{31}$$
$$pParamQ.f32CC2sc = \left(CC2_{SC} \cdot 2^{-NShift} \right) \cdot 2^{31}$$
$$pParamQ.u16NShift = NShift$$

Equation AMCLIB_BemfObsrvDQ_F32_Eq4

The upper and lower limits of the controllers should be set based on the expected dynamics of the system.

The stator winding model (R-L circuit) and cross-coupling coefficients can be set according to the following equations (including scaling):

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$IGain_{SC} = \frac{2 \cdot L_d \cdot T_S \cdot R_S}{2 \cdot L_d + T_S \cdot R_S}$$

$$UGain_{SC} = \frac{T_S}{2 \cdot L_d + T_S \cdot R_S} \cdot \frac{U_{MAX}}{I_{MAX}}$$

$$WIGain_{SC} = \frac{T_S \cdot L_q}{2 \cdot L_d + T_S \cdot R_S} \cdot \Omega_{MAX}$$

$$EGain_{SC} = \frac{T_S}{2 \cdot L_d + T_S \cdot R_S} \cdot \frac{E_{MAX}}{I_{MAX}}$$

Equation AMCLIB_BemfObsrvDQ_F32_Eq5

L_q is the q-axis stator inductance [H]. In case the coefficients exceed the fractional range or they are extremely small (which could lead to loss of precision), their value must be bit-shifted by *NShiftRL*. The shift amount can be calculated as

$$NShiftRL = \lceil \log_2(\max(|UGain_{SC}|, |WIGain_{SC}|, |EGain_{SC}|)) \rceil$$

Equation AMCLIB_BemfObsrvDQ_F32_Eq6

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 31st power of two (corresponds to the Q1.31 encoding):

$$f32IGain = IGain_{SC} \cdot 2^{31}$$

$$f32UGain = \left(UGain_{SC} \cdot 2^{-NShiftRL}\right) \cdot 2^{31}$$

$$f32WIGain = \left(WIGain_{SC} \cdot 2^{-NShiftRL}\right) \cdot 2^{31}$$

$$f32EGain = \left(EGain_{SC} \cdot 2^{-NShiftRL}\right) \cdot 2^{31}$$

$$s16Shift = NShiftRL$$

Equation AMCLIB_BemfObsrvDQ_F32_Eq7

The following m-script can be passed to the Matlab® command window to calculate the BEMF observer coefficients from the motor parameters:

```
% Motor parameters
% (to be set according to measurements)
Ld = 3.0e-4; % inductance in d-axis [H]
Lq = 3.0e-4; % inductance in q-axis [H]
Rs = 0.33;  % resistance of one stator phase [Ω]

% Scaling constants
% (to be set according to known maxima)
Imax = 20;   % maximum stator phase current [A]
Umax = 14.4; % maximum stator phase voltage [V]
Wmax = 2618; % maximum electrical angular velocity [rad/s]
Emax = 14.4; % maximum BEMF [V]

% Control system parameters
% (to be set according to the chosen control system dynamics)
Ts = 1e-4;            % sampling period [s]
i_Ksi = 1;            % current loop attenuation
i_fo = 350;           % current loop natural frequency [Hz]
i_wo = 2*pi()*i_fo;   % current loop natural angular frequency [rad/s]
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
Kp = 2*i_Ksi*i_wo*Ld-Rs;
Ki = i_wo^2*Ld;

disp('--- AMCLIB_BemfObsrvDQ_F32 coefficients ---')
% PIrAW controller parameters
maxCoeff = max(abs([( Kp + Ki*Ts/2)*Imax/Emax,...
                    (-Kp + Ki*Ts/2)*Imax/Emax]));
NShift = max(0, ceil(log2(maxCoeff)));
if (NShift > 14)
    error('Inputted parameters cannot be used - u16NShift exceeds 14');
end
pCtrl_pParamD_f32CC1sc = (Kp + Ki*Ts/2)*Imax/Emax*2^-NShift;
pCtrl_pParamD_f32CC1sc = round(pCtrl_pParamD_f32CC1sc * 2^31);
pCtrl_pParamD_f32CC1sc(pCtrl_pParamD_f32CC1sc < -(2^31)) = -(2^31);
pCtrl_pParamD_f32CC1sc(pCtrl_pParamD_f32CC1sc > (2^31)-1) = (2^31)-1;
pCtrl_pParamD_f32CC2sc = (-Kp + Ki*Ts/2)*Imax/Emax*2^-NShift;
pCtrl_pParamD_f32CC2sc = round(pCtrl_pParamD_f32CC2sc * 2^31);
pCtrl_pParamD_f32CC2sc(pCtrl_pParamD_f32CC2sc < -(2^31)) = -(2^31);
pCtrl_pParamD_f32CC2sc(pCtrl_pParamD_f32CC2sc > (2^31)-1) = (2^31)-1;
pCtrl_pParamD_u16NShift = NShift;
pCtrl_pParamQ_f32CC1sc = pCtrl_pParamD_f32CC1sc;
pCtrl_pParamQ_f32CC2sc = pCtrl_pParamD_f32CC2sc;
pCtrl_pParamQ_u16NShift = NShift;
disp(['Ctrl.pParamD.f32CC1sc = ' num2str(pCtrl_pParamD_f32CC1sc) ';'])
disp(['Ctrl.pParamD.f32CC2sc = ' num2str(pCtrl_pParamD_f32CC2sc) ';'])
disp(['Ctrl.pParamD.u16NShift = ' num2str(NShift) ';'])
disp(['Ctrl.pParamQ.f32CC1sc = ' num2str(pCtrl_pParamQ_f32CC1sc) ';'])
disp(['Ctrl.pParamQ.f32CC2sc = ' num2str(pCtrl_pParamQ_f32CC2sc) ';'])
disp(['Ctrl.pParamQ.u16NShift = ' num2str(NShift) ';'])
disp('  Ctrl.pParamD.f32UpperLimit, Ctrl.pParamD.f32LowerLimit, ')
disp('  Ctrl.pParamQ.f32UpperLimit, and Ctrl.pParamQ.f32LowerLimit')
disp('  shall be set according to the expected dynamics')

% RL circuit parameters
maxCoeffRL = max(abs([Ts/(2*Ld+Ts*Rs)*Umax/Imax,...
                      Ts*Lq/(2*Ld+Ts*Rs)*Wmax,...
                      Ts/(2*Ld+Ts*Rs)*Emax/Imax]));
NShiftRL = ceil(log2(maxCoeffRL));
if (NShiftRL < -14)
    NShiftRL = -14;
end
if (NShiftRL > 14)
    error('Inputted parameters cannot be used - s16Shift exceeds 14');
end
pCtrl_f32IGain = (2*Ld-Ts*Rs)/(2*Ld+Ts*Rs);
pCtrl_f32IGain = round(pCtrl_f32IGain * 2^31);
pCtrl_f32IGain(pCtrl_f32IGain < -(2^31)) = -(2^31);
pCtrl_f32IGain(pCtrl_f32IGain > (2^31)-1) = (2^31)-1;
pCtrl_f32UGain = Ts/(2*Ld+Ts*Rs)*Umax/Imax*2^-NShiftRL;
pCtrl_f32UGain = round(pCtrl_f32UGain * 2^31);
pCtrl_f32UGain(pCtrl_f32UGain < -(2^31)) = -(2^31);
pCtrl_f32UGain(pCtrl_f32UGain > (2^31)-1) = (2^31)-1;
pCtrl_f32WIGain = Ts*Lq/(2*Ld+Ts*Rs)*Wmax*2^-NShiftRL;
pCtrl_f32WIGain = round(pCtrl_f32WIGain * 2^31);
pCtrl_f32WIGain(pCtrl_f32WIGain < -(2^31)) = -(2^31);
pCtrl_f32WIGain(pCtrl_f32WIGain > (2^31)-1) = (2^31)-1;
pCtrl_f32EGain = Ts/(2*Ld+Ts*Rs)*Emax/Imax*2^-NShiftRL;
pCtrl_f32EGain = round(pCtrl_f32EGain * 2^31);
pCtrl_f32EGain(pCtrl_f32EGain < -(2^31)) = -(2^31);
pCtrl_f32EGain(pCtrl_f32EGain > (2^31)-1) = (2^31)-1;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
disp(['Ctrl.f32IGain = ' num2str(pCtrl_f32IGain) ';'])
disp(['Ctrl.f32UGain = ' num2str(pCtrl_f32UGain) ';'])
disp(['Ctrl.f32WIGain = ' num2str(pCtrl_f32WIGain) ';'])
disp(['Ctrl.f32EGain = ' num2str(pCtrl_f32EGain) ';'])
disp(['Ctrl.s16Shift = ' num2str(NShiftRL) ';'])
```

The followin block diagram shows the internal structure of the 32-bit BEMF observer.

Figure 36. 32-bit BEMF observer structure

The accuracy of results is guaranteed for the outputs pEObsrv.f32Arg1 and pEObsrv.f32Arg2 only in cases when pParamD.u16NShift, pParamQ.u16NShift, and s16Shift are not greater than 1. There is no limit of computational error specified for the returned value. The actual error depends on the values of pEObsrv.f32Arg1 and pEObsrv.f32Arg2. The following figure shows the expected values of absolute error [16-bit LSB] contained in the returned value in the cases when all shifts are equal to 1.

Figure 37. Absolute error of the returned value, assuming all shifts are 1

Note: *The BEMF observer coefficients f32IGain, f32UGain, f32WIGain, and f32EGain must not contain the largest negative value, otherwise the accuracy of the results is not guaranteed.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Keep the values of pParamD.u16NShift, pParamQ.u16NShift, and s16Shift within the allowed limits to prevent an overflow of intermediate results.

The function performs the fastest when s16Shift is equal to zero.

This function is implemented using inline assembly and is therefore not ANSI-C compliant.

2.2.3.5 Code Example

```
#include "amclib.h"

SWLIBS_2Syst_F32              mcIab, mcUab;
AMCLIB_BEMF_OBSRV_DQ_T_F32    BemfObsrv;
tFrac32                       f32Velocity;
tFrac32                       f32Phase;

void main (void)
{
    // Clear BEMF observer state variables:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQInit_F32(&BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQInit(&BemfObsrv, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_BemfObsrvDQInit(&BemfObsrv);

    // Set BEMF observer parameters:
    BemfObsrv.pParamD.f32CC1sc      = (tFrac32)1583784859;
    BemfObsrv.pParamD.f32CC2sc      = (tFrac32)-1367421132;
    BemfObsrv.pParamD.f32UpperLimit = (tFrac32)2147483647;
    BemfObsrv.pParamD.f32LowerLimit = (tFrac32)-2147483648;
    BemfObsrv.pParamD.u16NShift     = (tU16)1;
    BemfObsrv.pParamQ.f32CC1sc      = (tFrac32)1583784859;
    BemfObsrv.pParamQ.f32CC2sc      = (tFrac32)-1367421132;
    BemfObsrv.pParamQ.f32UpperLimit = (tFrac32)2147483647;
    BemfObsrv.pParamQ.f32LowerLimit = (tFrac32)-2147483648;
    BemfObsrv.pParamQ.u16NShift     = (tU16)1;
    BemfObsrv.f32IGain  = (tFrac32)1923575400;
    BemfObsrv.f32UGain  = (tFrac32)1954108343;
    BemfObsrv.f32WIGain = (tFrac32)2131606518;
    BemfObsrv.f32EGain  = (tFrac32)1954108343;
    BemfObsrv.s16Shift  = (tS16)-3;

    // Obtain the initial estimate of the rotor position and velocity,
    // measure the alpha/beta voltages and currents
    // (...)

    // Initialize the internal states of the observer to achieve
    // seamless transition from an uncontrolled state of the motor
    // to the full feedback control:
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQSetState_F32(&mcIab, &mcUab, f32Velocity,
      f32Phase, &BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQSetState(&mcIab, &mcUab, f32Velocity,
      f32Phase, &BemfObsrv, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_BemfObsrvDQSetState(&mcIab, &mcUab, f32Velocity,
      f32Phase, &BemfObsrv);

    while(1);
}

// Periodical function or interrupt
void ISR(void)
{
    tFrac32 f32PhaseErr;

    // Read the A/D, calculate alpha-beta values, etc.
    // (...)

    // Calculate one iteration of the observer:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f32PhaseErr = AMCLIB_BemfObsrvDQ_F32(&mcIab, &mcUab, f32Velocity,
      f32Phase, &BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f32PhaseErr = AMCLIB_BemfObsrvDQ(&mcIab, &mcUab, f32Velocity,
      f32Phase, &BemfObsrv, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    f32PhaseErr = AMCLIB_BemfObsrvDQ(&mcIab, &mcUab, f32Velocity,
      f32Phase, &BemfObsrv);

    // Pass f32PhaseErr to the #AMCLIB_TrackObsrv_F32
    // (...)
}
```

2.2.4 Function AMCLIB_BemfObsrvDQ_F16

2.2.4.1 Declaration

```
tFrac16 AMCLIB_BemfObsrvDQ_F16(const SWLIBS_2Syst_F16 *const
pIAB, const SWLIBS_2Syst_F16 *const pUAB, tFrac16 f16Velocity,
tFrac16 f16Phase, AMCLIB_BEMF_OBSRV_DQ_T_F16 *const pCtrl);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.2.4.2 Arguments

Table 5. AMCLIB_BemfObsrvDQ_F16 arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_F16 *const	pIAB	**input**	Pointer to the structure with Alpha/Beta current components.
const SWLIBS_2Syst_F16 *const	pUAB	**input**	Pointer to the structure with Alpha/Beta voltage components.
tFrac16	f16Velocity	**input**	Estimated electrical angular velocity.
tFrac16	f16Phase	**input**	Estimated rotor flux angle.
AMCLIB_BEMF_OBSRV_DQ_T_F16 *const	pCtrl	**input, output**	Pointer to the structure with BEMF observer parameters and state.

2.2.4.3 Return

Phase error between the real rotating reference frame and the estimated one.

2.2.4.4 Implementation details

Prior to calculating the BEMF observer coefficients, it is necessary to set the scaling constants. All inputs and outputs of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation. There are two different scaling systems involved, one for the FOC part of the motor control algorithm, and another for the BEMF observer. Scaling constants must be positive values equal to or greater than the expected maxima of the corresponding physical quantities. The following scaling constants are applied to the BEMF observer coefficients:

Table 6. Scaling constants

Scaling constant	Symbol	Calculation
Maximum stator phase voltage [V]	U_{MAX}	$U_{MAX}=U_{DC_Bus_Max}$
Maximum phase current [A]	I_{MAX}	Maximum current of the inverter or nominal current of the motor (whichever is lower).
Maximum speed [rad/s]	Ω_{MAX}	Maximum application required speed, at least the motor electrical rated speed.
Maximum BEMF voltage [V]	E_{MAX}	In normal operation is equal to U_{MAX}, in case of, e.g., field weakening, might be much higher.

The BEMF observer contains two controllers (GFLIB_ControllerPIrAW_F16). The proportional gain K_P and integral gain K_I of these controllers can be calculated as follows:

$$K_P = 2 \cdot \xi \cdot \omega_0 \cdot L_d - R_S$$
$$K_I = \omega_0^2 \cdot L_d$$

Equation AMCLIB_BemfObsrvDQ_F16_Eq1

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

where ξ is the current loop attenuation, ω_0 is the current loop natural frequency [rad/s], L_d is the d-axis stator inductance [H], and R_S is the winding resistance [ohm]. Coefficients ξ and ω_0 should correspond to the values chosen for the FOC current loop.

The continuous-time controller parameters K_P and K_I are then discretized using the bilinear transform. The resulting difference equation coefficients must be scaled so that the input and output values of the controllers fit in the fractional range.

$$CC1_{SC} = \left(K_P + \frac{K_I T_S}{2}\right) \cdot \frac{I_{MAX}}{E_{MAX}}$$
$$CC2_{SC} = \left(-K_P + \frac{K_I T_S}{2}\right) \cdot \frac{I_{MAX}}{E_{MAX}}$$

Equation AMCLIB_BemfObsrvDQ_F16_Eq2

T_S is the sampling period [s]. In case the coefficients $CC1_{SC}$ and $CC2_{SC}$ exceed the fractional range, their value must be right-shifted by *NShift*. The binary shift amount corresponds to

$$NShift = \max\left(0, \left\lceil \log_2\left(\max\left(|CC1_{SC}|, |CC2_{SC}|\right)\right)\right\rceil\right)$$

Equation AMCLIB_BemfObsrvDQ_F16_Eq3

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 15th power of two (corresponds to the Q1.15 encoding):

$$pParamD.f16CC1sc = \left(CC1_{SC} \cdot 2^{-NShift}\right) \cdot 2^{15}$$
$$pParamD.f16CC2sc = \left(CC2_{SC} \cdot 2^{-NShift}\right) \cdot 2^{15}$$
$$pParamD.u16NShift = NShift$$
$$pParamQ.f16CC1sc = \left(CC1_{SC} \cdot 2^{-NShift}\right) \cdot 2^{15}$$
$$pParamQ.f16CC2sc = \left(CC2_{SC} \cdot 2^{-NShift}\right) \cdot 2^{15}$$
$$pParamQ.u16NShift = NShift$$

Equation AMCLIB_BemfObsrvDQ_F16_Eq4

The upper and lower limits of the controllers should be set based on the expected dynamics of the system.

The stator winding model (R-L circuit) and cross-coupling coefficients can be set according to the following equations (including scaling):

$$IGain_{SC} = \frac{2 \cdot L_d - T_S \cdot R_S}{2 \cdot L_d + T_S \cdot R_S}$$
$$UGain_{SC} = \frac{T_S}{2 \cdot L_d + T_S \cdot R_S} \cdot \frac{U_{MAX}}{I_{MAX}}$$
$$WIGain_{SC} = \frac{T_S \cdot L_q}{2 \cdot L_d + T_S \cdot R_S} \cdot \Omega_{MAX}$$
$$EGain_{SC} = \frac{T_S}{2 \cdot L_d + T_S \cdot R_S} \cdot \frac{E_{MAX}}{I_{MAX}}$$

Equation AMCLIB_BemfObsrvDQ_F16_Eq5

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

L_q is the q-axis stator inductance [H]. In case the coefficients exceed the fractional range or they are extremely small (which could lead to loss of precision), their value must be bit-shifted by *NShiftRL*. The shift amount can be calculated as

$$NShiftRL = \lceil \log_2(\max(|UGain_{SC}|, |WIGain_{SC}|, |EGain_{SC}|)) \rceil$$

Equation AMCLIB_BemfObsrvDQ_F16_Eq6

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 15th power of two (corresponds to the Q1.15 encoding):

$$f16IGain = IGain_{SC} \cdot 2^{15}$$
$$f16UGain = \left(UGain_{SC} \cdot 2^{-NShiftRL}\right) \cdot 2^{15}$$
$$f16WIGain = \left(WIGain_{SC} \cdot 2^{-NShiftRL}\right) \cdot 2^{15}$$
$$f16EGain = \left(EGain_{SC} \cdot 2^{-NShiftRL}\right) \cdot 2^{15}$$
$$s16Shift = NShiftRL$$

Equation AMCLIB_BemfObsrvDQ_F16_Eq7

The following m-script can be passed to the Matlab® command window to calculate the BEMF observer coefficients from the motor parameters:

```
% Motor parameters
% (to be set according to measurements)
Ld = 3.0e-4; % inductance in d-axis [H]
Lq = 3.0e-4; % inductance in q-axis [H]
Rs = 0.33;  % resistance of one stator phase [Ω]

% Scaling constants
% (to be set according to known maxima)
Imax = 20;   % maximum stator phase current [A]
Umax = 14.4; % maximum stator phase voltage [V]
Wmax = 2618; % maximum electrical angular velocity [rad/s]
Emax = 14.4; % maximum BEMF [V]

% Control system parameters
% (to be set according to the chosen control system dynamics)
Ts = 1e-4;            % sampling period [s]
i_Ksi = 1;            % current loop attenuation
i_fo = 350;           % current loop natural frequency [Hz]
i_wo = 2*pi()*i_fo;   % current loop natural angular frequency [rad/s]
Kp = 2*i_Ksi*i_wo*Ld-Rs;
Ki = i_wo^2*Ld;

disp('--- AMCLIB_BemfObsrvDQ_F16 coefficients ---')
% PIrAW controller parameters
maxCoeff = max(abs([( Kp + Ki*Ts/2)*Imax/Emax,...
                    (-Kp + Ki*Ts/2)*Imax/Emax]));
NShift = max(0, ceil(log2(maxCoeff)));
if (NShift > 14)
    error('Inputted parameters cannot be used - u16NShift exceeds 14');
end
pCtrl_pParamD_f16CC1sc = (Kp + Ki*Ts/2)*Imax/Emax*2^-NShift;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
pCtrl_pParamD_f16CC1sc = round(pCtrl_pParamD_f16CC1sc * 2^15);
pCtrl_pParamD_f16CC1sc(pCtrl_pParamD_f16CC1sc < -(2^15)) = -(2^15);
pCtrl_pParamD_f16CC1sc(pCtrl_pParamD_f16CC1sc > (2^15)-1) = (2^15)-1;
pCtrl_pParamD_f16CC2sc = (-Kp + Ki*Ts/2)*Imax/Emax*2^-NShift;
pCtrl_pParamD_f16CC2sc = round(pCtrl_pParamD_f16CC2sc * 2^15);
pCtrl_pParamD_f16CC2sc(pCtrl_pParamD_f16CC2sc < -(2^15)) = -(2^15);
pCtrl_pParamD_f16CC2sc(pCtrl_pParamD_f16CC2sc > (2^15)-1) = (2^15)-1;
pCtrl_pParamD_u16NShift = NShift;
pCtrl_pParamQ_f16CC1sc = pCtrl_pParamD_f16CC1sc;
pCtrl_pParamQ_f16CC2sc = pCtrl_pParamD_f16CC2sc;
pCtrl_pParamQ_u16NShift = NShift;
disp(['Ctrl.pParamD.f16CC1sc = ' num2str(pCtrl_pParamD_f16CC1sc) ';'])
disp(['Ctrl.pParamD.f16CC2sc = ' num2str(pCtrl_pParamD_f16CC2sc) ';'])
disp(['Ctrl.pParamD.u16NShift = ' num2str(NShift) ';'])
disp(['Ctrl.pParamQ.f16CC1sc = ' num2str(pCtrl_pParamQ_f16CC1sc) ';'])
disp(['Ctrl.pParamQ.f16CC2sc = ' num2str(pCtrl_pParamQ_f16CC2sc) ';'])
disp(['Ctrl.pParamQ.u16NShift = ' num2str(NShift) ';'])
disp('  Ctrl.pParamD.f16UpperLimit, Ctrl.pParamD.f16LowerLimit, ')
disp('  Ctrl.pParamQ.f16UpperLimit, and Ctrl.pParamQ.f16LowerLimit')
disp('  shall be set according to the expected dynamics')

% RL circuit parameters
maxCoeffRL = max(abs([Ts/(2*Ld+Ts*Rs)*Umax/Imax,...
                      Ts*Lq/(2*Ld+Ts*Rs)*Wmax,...
                      Ts/(2*Ld+Ts*Rs)*Emax/Imax]));
NShiftRL = ceil(log2(maxCoeffRL));
if (NShiftRL < -14)
    NShiftRL = -14;
end
if (NShiftRL > 14)
    error('Inputted parameters cannot be used - s16Shift exceeds 14');
end
pCtrl_f16IGain = (2*Ld-Ts*Rs)/(2*Ld+Ts*Rs);
pCtrl_f16IGain = round(pCtrl_f16IGain * 2^15);
pCtrl_f16IGain(pCtrl_f16IGain < -(2^15)) = -(2^15);
pCtrl_f16IGain(pCtrl_f16IGain > (2^15)-1) = (2^15)-1;
pCtrl_f16UGain = Ts/(2*Ld+Ts*Rs)*Umax/Imax*2^-NShiftRL;
pCtrl_f16UGain = round(pCtrl_f16UGain * 2^15);
pCtrl_f16UGain(pCtrl_f16UGain < -(2^15)) = -(2^15);
pCtrl_f16UGain(pCtrl_f16UGain > (2^15)-1) = (2^15)-1;
pCtrl_f16WIGain = Ts*Lq/(2*Ld+Ts*Rs)*Wmax*2^-NShiftRL;
pCtrl_f16WIGain = round(pCtrl_f16WIGain * 2^15);
pCtrl_f16WIGain(pCtrl_f16WIGain < -(2^15)) = -(2^15);
pCtrl_f16WIGain(pCtrl_f16WIGain > (2^15)-1) = (2^15)-1;
pCtrl_f16EGain = Ts/(2*Ld+Ts*Rs)*Emax/Imax*2^-NShiftRL;
pCtrl_f16EGain = round(pCtrl_f16EGain * 2^15);
pCtrl_f16EGain(pCtrl_f16EGain < -(2^15)) = -(2^15);
pCtrl_f16EGain(pCtrl_f16EGain > (2^15)-1) = (2^15)-1;
disp(['Ctrl.f16IGain = ' num2str(pCtrl_f16IGain) ';'])
disp(['Ctrl.f16UGain = ' num2str(pCtrl_f16UGain) ';'])
disp(['Ctrl.f16WIGain = ' num2str(pCtrl_f16WIGain) ';'])
disp(['Ctrl.f16EGain = ' num2str(pCtrl_f16EGain) ';'])
disp(['Ctrl.s16Shift = ' num2str(NShiftRL) ';'])
```

The followin block diagram shows the internal structure of the 16-bit BEMF observer.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 38. 16-bit BEMF observer structure

The accuracy of results is guaranteed for the outputs pEObsrv.f16Arg1 and pEObsrv.f16Arg2 only in cases when pParamD.u16NShift, pParamQ.u16NShift, and s16Shift are not greater than 1. There is no limit of computational error specified for the returned value. The actual error depends on the values of pEObsrv.f16Arg1 and pEObsrv.f16Arg2. The following figure shows the expected values of absolute error [16-bit LSB] contained in the returned value in the cases when all shifts are equal to 1.

Figure 39. Absolute error of the returned value, assuming all shifts are 1

Note: *The BEMF observer coefficients f16IGain, f16UGain, f16WIGain, and f16EGain must not contain the largest negative value, otherwise the accuracy of the results is not guaranteed.*

Keep the values of pParamD.u16NShift, pParamQ.u16NShift, and s16Shift within the allowed limits to prevent an overflow of intermediate results.

The function performs the fastest when s16Shift is equal to zero.

This function is implemented using inline assembly and is therefore not ANSI-C compliant.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.2.4.5 Code Example

```
#include "amclib.h"

SWLIBS_2Syst_F16              mcIab, mcUab;
AMCLIB_BEMF_OBSRV_DQ_T_F16    BemfObsrv;
tFrac16                       f16Velocity;
tFrac16                       f16Phase;

void main (void)
{
    // Clear BEMF observer state variables:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQInit_F16(&BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQInit(&BemfObsrv, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_BemfObsrvDQInit(&BemfObsrv);

    // Set BEMF observer parameters:
    BemfObsrv.pParamD.f16CC1sc      = (tFrac16)24167;
    BemfObsrv.pParamD.f16CC2sc      = (tFrac16)-20865;
    BemfObsrv.pParamD.f16UpperLimit = (tFrac16)32767;
    BemfObsrv.pParamD.f16LowerLimit = (tFrac16)-32768;
    BemfObsrv.pParamD.u16NShift     = (tU16)1;
    BemfObsrv.pParamQ.f16CC1sc      = (tFrac16)24167;
    BemfObsrv.pParamQ.f16CC2sc      = (tFrac16)-20865;
    BemfObsrv.pParamQ.f16UpperLimit = (tFrac16)32767;
    BemfObsrv.pParamQ.f16LowerLimit = (tFrac16)-32768;
    BemfObsrv.pParamQ.u16NShift     = (tU16)1;
    BemfObsrv.f16IGain  = (tFrac16)29351;
    BemfObsrv.f16UGain  = (tFrac16)29817;
    BemfObsrv.f16WIGain = (tFrac16)32526;
    BemfObsrv.f16EGain  = (tFrac16)29817;
    BemfObsrv.s16Shift  = (tS16)-3;

    // Obtain the initial estimate of the rotor position and velocity,
    // measure the alpha/beta voltages and currents
    // (...)

    // Initialize the internal states of the observer to achieve
    // seamless transition from an uncontrolled state of the motor
    // to the full feedback control:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQSetState_F16(&mcIab, &mcUab, f16Velocity,
      f16Phase, &BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQSetState(&mcIab, &mcUab, f16Velocity,
      f16Phase, &BemfObsrv, F16);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_BemfObsrvDQSetState(&mcIab, &mcUab, f16Velocity,
      f16Phase, &BemfObsrv);

    while(1);
}

// Periodical function or interrupt
void ISR(void)
{
    tFrac16 f16PhaseErr;

    // Read the A/D, calculate alpha-beta values, etc.
    // (...)

    // Calculate one iteration of the observer:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f16PhaseErr = AMCLIB_BemfObsrvDQ_F16(&mcIab, &mcUab, f16Velocity,
      f16Phase, &BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f16PhaseErr = AMCLIB_BemfObsrvDQ(&mcIab, &mcUab, f16Velocity,
      f16Phase, &BemfObsrv, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    f16PhaseErr = AMCLIB_BemfObsrvDQ(&mcIab, &mcUab, f16Velocity,
      f16Phase, &BemfObsrv);

    // Pass f16PhaseErr to the #AMCLIB_TrackObsrv_F16
    // (...)
}
```

2.2.5 Function AMCLIB_BemfObsrvDQ_FLT

2.2.5.1 Declaration

```
tFloat AMCLIB_BemfObsrvDQ_FLT(const SWLIBS_2Syst_FLT *const pIAB,
const SWLIBS_2Syst_FLT *const pUAB, tFloat fltVelocity, tFloat
fltPhase, AMCLIB_BEMF_OBSRV_DQ_T_FLT *const pCtrl);
```

2.2.5.2 Arguments

Table 7. AMCLIB_BemfObsrvDQ_FLT arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_FLT *const	pIAB	**input**	Pointer to the structure with Alpha/Beta current components [A].
const SWLIBS_2Syst_FLT *const	pUAB	**input**	Pointer to the structure with Alpha/Beta voltage components [V].
tFloat	fltVelocity	**input**	Estimated electrical angular velocity [rad/s].

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 7. AMCLIB_BemfObsrvDQ_FLT arguments...*continued*

Type	Name	Direction	Description
tFloat	fltPhase	**input**	Estimated rotor flux angle [rad], must be in range [-π, π].
AMCLIB_BEMF_OBSRV_DQ_T_FLT *const	pCtrl	**input, output**	Pointer to the structure with BEMF observer parameters and state.

2.2.5.3 Return

Phase error between the real rotating reference frame and the estimated one [rad].

2.2.5.4 Implementation details

The BEMF observer contains two controllers (GFLIB_ControllerPIrAW_FLT). The proportional gain K_P and integral gain K_I of these controllers can be calculated as follows:

$$K_P = 2 \cdot \xi \cdot \omega_0 \cdot L_d - R_S$$
$$K_I = \omega_0^2 \cdot L_d$$

Equation AMCLIB_BemfObsrvDQ_FLT_Eq1

where ξ is the current loop attenuation, ω_0 is the current loop natural frequency [rad/s], L_d is the d-axis stator inductance [H], and R_S is the winding resistance [ohm]. Coefficients ξ and ω_0 should correspond to the values chosen for the FOC current loop.

The continuous-time controller parameters K_P and K_I are then discretized using the bilinear transform. The resulting difference equation coefficients can be used in the parameters structure.

$$pParamD.fltCC1sc = K_P + \frac{K_I T_S}{2}$$
$$pParamD.fltCC2sc = -K_P + \frac{K_I T_S}{2}$$
$$pParamQ.fltCC1sc = K_P + \frac{K_I T_S}{2}$$
$$pParamQ.fltCC2sc = -K_P + \frac{K_I T_S}{2}$$

Equation AMCLIB_BemfObsrvDQ_FLT_Eq2

T_S is the sampling period [s]. The upper and lower limits of the controllers should be set based on the expected dynamics of the system.

The stator winding model (R-L circuit) and cross-coupling coefficients can be set according to the following equations:

$$fltIGain = \frac{2 \cdot L_d - T_S \cdot R_S}{2 \cdot L_d + T_S \cdot R_S}$$
$$fltUGain = \frac{T_S}{2 \cdot L_d + T_S \cdot R_S}$$
$$fltWIGain = \frac{T_S \cdot L_q}{2 \cdot L_d + T_S \cdot R_S}$$
$$fltEGain = \frac{T_S}{2 \cdot L_d + T_S \cdot R_S}$$

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Equation AMCLIB_BemfObsrvDQ_FLT_Eq3

L_q is the q-axis stator inductance [H]. The following m-script can be passed to the Matlab® command window to calculate the BEMF observer coefficients from the motor parameters:

```
% Motor parameters
% (to be set according to measurements)
Ld = 3.0e-4; % inductance in d-axis [H]
Lq = 3.0e-4; % inductance in q-axis [H]
Rs = 0.33;  % resistance of one stator phase [Ω]

% Control system parameters
% (to be set according to the chosen control system dynamics)
Ts = 1e-4;            % sampling period [s]
i_Ksi = 1;            % current loop attenuation
i_fo = 350;           % current loop natural frequency [Hz]
i_wo = 2*pi()*i_fo;   % current loop natural angular frequency [rad/s]
Kp = 2*i_Ksi*i_wo*Ld-Rs;
Ki = i_wo^2*Ld;

disp('--- AMCLIB_BemfObsrvDQ_FLT coefficients ---')
% PIrAW controller coefficients
pCtrl_pParamD_fltCC1sc = Kp + Ki*Ts/2;
pCtrl_pParamD_fltCC2sc = -Kp + Ki*Ts/2;
pCtrl_pParamQ_fltCC1sc = pCtrl_pParamD_fltCC1sc;
pCtrl_pParamQ_fltCC2sc = pCtrl_pParamD_fltCC2sc;
disp(['Ctrl.pParamD.fltCC1sc = '...
    num2str(pCtrl_pParamD_fltCC1sc,'%1.16e\n') ';'])
disp(['Ctrl.pParamD.fltCC2sc = ' ...
    num2str(pCtrl_pParamD_fltCC2sc,'%1.16e\n') ';'])
disp(['Ctrl.pParamQ.fltCC1sc = ' ...
    num2str(pCtrl_pParamQ_fltCC1sc,'%1.16e\n') ';'])
disp(['Ctrl.pParamQ.fltCC2sc = ' ...
    num2str(pCtrl_pParamQ_fltCC2sc,'%1.16e\n') ';'])
disp('  Ctrl.pParamD.fltUpperLimit, Ctrl.pParamD.fltLowerLimit, ')
disp('  Ctrl.pParamQ.fltUpperLimit, and Ctrl.pParamQ.fltLowerLimit')
disp('  shall be set according to the expected dynamics')

% R-L circuit coefficients
pCtrl_fltIGain = (2*Ld-Ts*Rs)/(2*Ld+Ts*Rs);
pCtrl_fltUGain = Ts/(2*Ld+Ts*Rs);
pCtrl_fltWIGain = Ts*Lq/(2*Ld+Ts*Rs);
pCtrl_fltEGain = Ts/(2*Ld+Ts*Rs);
disp(['Ctrl.fltIGain = ' num2str(pCtrl_fltIGain,'%1.16e\n') ';'])
disp(['Ctrl.fltUGain = ' num2str(pCtrl_fltUGain,'%1.16e\n') ';'])
disp(['Ctrl.fltWIGain = ' num2str(pCtrl_fltWIGain,'%1.16e\n') ';'])
disp(['Ctrl.fltEGain = ' num2str(pCtrl_fltEGain,'%1.16e\n') ';'])
```

The following histogram shows the empirical probability of the relative error magnitude contained in the returned value in a typical motor control application. The x-axis scale corresponds to the floating-point mantissa binary digits; 0 = LSB, 23 = MSB.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 40. Histogram of the returned value error

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.2.5.5 Code Example

```
#include "amclib.h"

#define Ts  (1e-4F)
#define Ksi (1.0F)
#define w0  (2.0F*3.14F*350.0F)
#define Ld  (3e-4F)
#define Lq  (3e-4F)
#define Rs  (0.33F)
#define Kp  (2.0F*Ksi*w0*Ld-Rs)
#define Ki  (w0*w0*Ld)

SWLIBS_2Syst_FLT            mcIab, mcUab;
AMCLIB_BEMF_OBSRV_DQ_T_FLT  BemfObsrv;
tFloat                      fltVelocity;
tFloat                      fltPhase;

void main (void)
{
    // Clear BEMF observer state variables:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQInit_FLT(&BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQInit(&BemfObsrv, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if floating point implementation is selected as default.
    AMCLIB_BemfObsrvDQInit(&BemfObsrv);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Set BEMF observer parameters:
    BemfObsrv.pParamD.fltCC1sc      =  Kp+Ki*Ts/2.0F;
    BemfObsrv.pParamD.fltCC2sc      = -Kp+Ki*Ts/2.0F;
    BemfObsrv.pParamD.fltUpperLimit =  1000.0F;
    BemfObsrv.pParamD.fltLowerLimit = -1000.0F;
    BemfObsrv.pParamQ.fltCC1sc      =  Kp+Ki*Ts/2.0F;
    BemfObsrv.pParamQ.fltCC2sc      = -Kp+Ki*Ts/2.0F;
    BemfObsrv.pParamQ.fltUpperLimit =  1000.0F;
    BemfObsrv.pParamQ.fltLowerLimit = -1000.0F;
    BemfObsrv.fltIGain  = (2.0F*Ld-Ts*Rs)/(2.0F*Ld+Ts*Rs);
    BemfObsrv.fltUGain  = Ts/(2.0F*Ld+Ts*Rs);
    BemfObsrv.fltWIGain = Ts*Lq/(2.0F*Ld+Ts*Rs);
    BemfObsrv.fltEGain  = Ts/(2.0F*Ld+Ts*Rs);

    // Obtain the initial estimate of the rotor position and velocity,
    // measure the alpha/beta voltages and currents
    // (...)

    // Initialize the internal states of the observer to achieve
    // seamless transition from an uncontrolled state of the motor
    // to the full feedback control:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQSetState_FLT(&mcIab, &mcUab, fltVelocity,
      fltPhase, &BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQSetState(&mcIab, &mcUab, fltVelocity,
      fltPhase, &BemfObsrv,FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if floating point implementation is selected as default.
    AMCLIB_BemfObsrvDQSetState(&mcIab, &mcUab, fltVelocity,
      fltPhase, &BemfObsrv);

    while(1);
}

// Periodical function or interrupt
void ISR(void)
{
    tFloat fltPhaseErr;

    // Read the A/D, calculate alpha-beta values, etc.
    // (...)

    // Calculate one iteration of the observer:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    fltPhaseErr = AMCLIB_BemfObsrvDQ_FLT(&mcIab, &mcUab, fltVelocity,
      fltPhase, &BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    fltPhaseErr = AMCLIB_BemfObsrvDQ(&mcIab, &mcUab, fltVelocity,
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
      fltPhase, &BemfObsrv,FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if floating point implementation is selected as default.
    fltPhaseErr = AMCLIB_BemfObsrvDQ(&mcIab, &mcUab, fltVelocity,
      fltPhase, &BemfObsrv);

    // Pass fltPhaseErr to the #AMCLIB_TrackObsrv_FLT
    // (...)
}
```

2.2.6 Function AMCLIB_BemfObsrvDQInit

2.2.6.1 Description

This function initializes all state variables of the BEMF observer to zero.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.2.6.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.2.6.3 Function AMCLIB_BemfObsrvDQInit_F32

2.2.6.3.1 Declaration

```
void AMCLIB_BemfObsrvDQInit_F32(AMCLIB_BEMF_OBSRV_DQ_T_F32 *const
pCtrl);
```

2.2.6.3.2 Arguments

Table 8. AMCLIB_BemfObsrvDQInit_F32 arguments

Type	Name	Direction	Description
AMCLIB_BEMF_OBSRV_DQ_T_F32 *const	pCtrl	**input, output**	Pointer to the structure with BEMF observer parameters and state.

2.2.6.3.3 Code Example

```
#include "amclib.h"

SWLIBS_2Syst_F32              mcIab, mcUab;
AMCLIB_BEMF_OBSRV_DQ_T_F32    BemfObsrv;
tFrac32                       f32Velocity;
tFrac32                       f32Phase;

void main (void)
{
    // Clear BEMF observer state variables:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQInit_F32(&BemfObsrv);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQInit(&BemfObsrv, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_BemfObsrvDQInit(&BemfObsrv);

    // Set BEMF observer parameters:
    BemfObsrv.pParamD.f32CC1sc      = (tFrac32)1583784859;
    BemfObsrv.pParamD.f32CC2sc      = (tFrac32)-1367421132;
    BemfObsrv.pParamD.f32UpperLimit = (tFrac32)2147483647;
    BemfObsrv.pParamD.f32LowerLimit = (tFrac32)-2147483648;
    BemfObsrv.pParamD.u16NShift     = (tU16)1;
    BemfObsrv.pParamQ.f32CC1sc      = (tFrac32)1583784859;
    BemfObsrv.pParamQ.f32CC2sc      = (tFrac32)-1367421132;
    BemfObsrv.pParamQ.f32UpperLimit = (tFrac32)2147483647;
    BemfObsrv.pParamQ.f32LowerLimit = (tFrac32)-2147483648;
    BemfObsrv.pParamQ.u16NShift     = (tU16)1;
    BemfObsrv.f32IGain  = (tFrac32)1923575400;
    BemfObsrv.f32UGain  = (tFrac32)1954108343;
    BemfObsrv.f32WIGain = (tFrac32)2131606518;
    BemfObsrv.f32EGain  = (tFrac32)1954108343;
    BemfObsrv.s16Shift  = (tS16)-3;

    // Obtain the initial estimate of the rotor position and velocity,
    // measure the alpha/beta voltages and currents
    // (...)

    // Initialize the internal states of the observer to achieve
    // seamless transition from an uncontrolled state of the motor
    // to the full feedback control:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQSetState_F32(&mcIab, &mcUab, f32Velocity,
      f32Phase, &BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQSetState(&mcIab, &mcUab, f32Velocity,
      f32Phase, &BemfObsrv, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_BemfObsrvDQSetState(&mcIab, &mcUab, f32Velocity,
      f32Phase, &BemfObsrv);

    while(1);
}

// Periodical function or interrupt
void ISR(void)
{
    tFrac32 f32PhaseErr;

    // Read the A/D, calculate alpha-beta values, etc.
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // (...)

    // Calculate one iteration of the observer:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f32PhaseErr = AMCLIB_BemfObsrvDQ_F32(&mcIab, &mcUab, f32Velocity,
      f32Phase, &BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f32PhaseErr = AMCLIB_BemfObsrvDQ(&mcIab, &mcUab, f32Velocity,
      f32Phase, &BemfObsrv, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    f32PhaseErr = AMCLIB_BemfObsrvDQ(&mcIab, &mcUab, f32Velocity,
      f32Phase, &BemfObsrv);

    // Pass f32PhaseErr to the #AMCLIB_TrackObsrv_F32
    // (...)
}
```

2.2.6.4 Function AMCLIB_BemfObsrvDQInit_F16

2.2.6.4.1 Declaration

```
void AMCLIB_BemfObsrvDQInit_F16(AMCLIB_BEMF_OBSRV_DQ_T_F16 *const
pCtrl);
```

2.2.6.4.2 Arguments

Table 9. AMCLIB_BemfObsrvDQInit_F16 arguments

Type	Name	Direction	Description
AMCLIB_BEMF_OBSRV_DQ_T_F16 *const	pCtrl	**input, output**	Pointer to the structure with BEMF observer parameters and state.

2.2.6.4.3 Code Example

```
#include "amclib.h"

SWLIBS_2Syst_F16              mcIab, mcUab;
AMCLIB_BEMF_OBSRV_DQ_T_F16    BemfObsrv;
tFrac16                       f16Velocity;
tFrac16                       f16Phase;

void main (void)
{
    // Clear BEMF observer state variables:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQInit_F16(&BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    AMCLIB_BemfObsrvDQInit(&BemfObsrv, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_BemfObsrvDQInit(&BemfObsrv);

    // Set BEMF observer parameters:
    BemfObsrv.pParamD.f16CC1sc      = (tFrac16)24167;
    BemfObsrv.pParamD.f16CC2sc      = (tFrac16)-20865;
    BemfObsrv.pParamD.f16UpperLimit = (tFrac16)32767;
    BemfObsrv.pParamD.f16LowerLimit = (tFrac16)-32768;
    BemfObsrv.pParamD.u16NShift     = (tU16)1;
    BemfObsrv.pParamQ.f16CC1sc      = (tFrac16)24167;
    BemfObsrv.pParamQ.f16CC2sc      = (tFrac16)-20865;
    BemfObsrv.pParamQ.f16UpperLimit = (tFrac16)32767;
    BemfObsrv.pParamQ.f16LowerLimit = (tFrac16)-32768;
    BemfObsrv.pParamQ.u16NShift     = (tU16)1;
    BemfObsrv.f16IGain  = (tFrac16)29351;
    BemfObsrv.f16UGain  = (tFrac16)29817;
    BemfObsrv.f16WIGain = (tFrac16)32526;
    BemfObsrv.f16EGain  = (tFrac16)29817;
    BemfObsrv.s16Shift  = (tS16)-3;

    // Obtain the initial estimate of the rotor position and velocity,
    // measure the alpha/beta voltages and currents
    // (...)

    // Initialize the internal states of the observer to achieve
    // seamless transition from an uncontrolled state of the motor
    // to the full feedback control:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQSetState_F16(&mcIab, &mcUab, f16Velocity,
      f16Phase, &BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQSetState(&mcIab, &mcUab, f16Velocity,
      f16Phase, &BemfObsrv, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_BemfObsrvDQSetState(&mcIab, &mcUab, f16Velocity,
      f16Phase, &BemfObsrv);

    while(1);
}

// Periodical function or interrupt
void ISR(void)
{
    tFrac16 f16PhaseErr;

    // Read the A/D, calculate alpha-beta values, etc.
    // (...)

    // Calculate one iteration of the observer:
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f16PhaseErr = AMCLIB_BemfObsrvDQ_F16(&mcIab, &mcUab, f16Velocity,
      f16Phase, &BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f16PhaseErr = AMCLIB_BemfObsrvDQ(&mcIab, &mcUab, f16Velocity,
      f16Phase, &BemfObsrv, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    f16PhaseErr = AMCLIB_BemfObsrvDQ(&mcIab, &mcUab, f16Velocity,
      f16Phase, &BemfObsrv);

    // Pass f16PhaseErr to the #AMCLIB_TrackObsrv_F16
    // (...)
}
```

2.2.6.5 Function AMCLIB_BemfObsrvDQInit_FLT

2.2.6.5.1 Declaration

```
void AMCLIB_BemfObsrvDQInit_FLT(AMCLIB_BEMF_OBSRV_DQ_T_FLT *const
pCtrl);
```

2.2.6.5.2 Arguments

Table 10. AMCLIB_BemfObsrvDQInit_FLT arguments

Type	Name	Direction	Description
AMCLIB_BEMF_OBSRV_DQ_T_FLT *const	pCtrl	**input, output**	Pointer to the structure with BEMF observer parameters and state.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.2.6.5.3 Code Example

```
#include "amclib.h"

#define Ts   (1e-4F)
#define Ksi  (1.0F)
#define w0   (2.0F*3.14F*350.0F)
#define Ld   (3e-4F)
#define Lq   (3e-4F)
#define Rs   (0.33F)
#define Kp   (2.0F*Ksi*w0*Ld-Rs)
#define Ki   (w0*w0*Ld)

SWLIBS_2Syst_FLT            mcIab, mcUab;
AMCLIB_BEMF_OBSRV_DQ_T_FLT  BemfObsrv;
tFloat                      fltVelocity;
tFloat                      fltPhase;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
void main (void)
{
    // Clear BEMF observer state variables:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQInit_FLT(&BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQInit(&BemfObsrv, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if floating point implementation is selected as default.
    AMCLIB_BemfObsrvDQInit(&BemfObsrv);

    // Set BEMF observer parameters:
    BemfObsrv.pParamD.fltCC1sc      =  Kp+Ki*Ts/2.0F;
    BemfObsrv.pParamD.fltCC2sc      = -Kp+Ki*Ts/2.0F;
    BemfObsrv.pParamD.fltUpperLimit =  1000.0F;
    BemfObsrv.pParamD.fltLowerLimit = -1000.0F;
    BemfObsrv.pParamQ.fltCC1sc      =  Kp+Ki*Ts/2.0F;
    BemfObsrv.pParamQ.fltCC2sc      = -Kp+Ki*Ts/2.0F;
    BemfObsrv.pParamQ.fltUpperLimit =  1000.0F;
    BemfObsrv.pParamQ.fltLowerLimit = -1000.0F;
    BemfObsrv.fltIGain  = (2.0F*Ld-Ts*Rs)/(2.0F*Ld+Ts*Rs);
    BemfObsrv.fltUGain  = Ts/(2.0F*Ld+Ts*Rs);
    BemfObsrv.fltWIGain = Ts*Lq/(2.0F*Ld+Ts*Rs);
    BemfObsrv.fltEGain  = Ts/(2.0F*Ld+Ts*Rs);

    // Obtain the initial estimate of the rotor position and velocity,
    // measure the alpha/beta voltages and currents
    // (...)

    // Initialize the internal states of the observer to achieve
    // seamless transition from an uncontrolled state of the motor
    // to the full feedback control:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQSetState_FLT(&mcIab, &mcUab, fltVelocity,
      fltPhase, &BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQSetState(&mcIab, &mcUab, fltVelocity,
      fltPhase, &BemfObsrv,FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if floating point implementation is selected as default.
    AMCLIB_BemfObsrvDQSetState(&mcIab, &mcUab, fltVelocity,
      fltPhase, &BemfObsrv);

    while(1);
}

// Periodical function or interrupt
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
void ISR(void)
{
    tFloat fltPhaseErr;

    // Read the A/D, calculate alpha-beta values, etc.
    // (...)

    // Calculate one iteration of the observer:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    fltPhaseErr = AMCLIB_BemfObsrvDQ_FLT(&mcIab, &mcUab, fltVelocity,
      fltPhase, &BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    fltPhaseErr = AMCLIB_BemfObsrvDQ(&mcIab, &mcUab, fltVelocity,
      fltPhase, &BemfObsrv,FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if floating point implementation is selected as default.
    fltPhaseErr = AMCLIB_BemfObsrvDQ(&mcIab, &mcUab, fltVelocity,
      fltPhase, &BemfObsrv);

    // Pass fltPhaseErr to the #AMCLIB_TrackObsrv_FLT
    // (...)
}
```

2.2.7 Function AMCLIB_BemfObsrvDQSetState

2.2.7.1 Description

This function sets the BEMF observer state variables to achieve zero difference between the measured and the predicted current and a zero voltage sum on the input of the current observer. The input voltages and currents are supplied in a stationary reference frame α/β and should be derived from the actual measured 3-phase motor voltages and currents (refer to the function GMCLIB_Clark for a transformation between the 3-phase coordinate system and the 2-phase α/β reference frame).

Setting the input flux angle and the angular velocity in accordance with the actual estimated position and velocity of the rotor enables seamless transition from an uncontrolled rotation of the rotor into a controlled state. A robust set of initial estimates can be obtained from function AMCLIB_Windmilling.

Note: *The observer parameters must be already set in the state structure pointed to by* `pCtrl` *before this function can be called.*

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.2.7.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.2.7.3 Function AMCLIB_BemfObsrvDQSetState_F32

2.2.7.3.1 Declaration

```
void AMCLIB_BemfObsrvDQSetState_F32(const SWLIBS_2Syst_F32 *const
pIAB, const SWLIBS_2Syst_F32 *const pUAB, tFrac32 f32Velocity,
tFrac32 f32Phase, AMCLIB_BEMF_OBSRV_DQ_T_F32 *const pCtrl);
```

2.2.7.3.2 Arguments

Table 11. AMCLIB_BemfObsrvDQSetState_F32 arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_F32 *const	pIAB	**input**	Pointer to the structure with Alpha/Beta current components.
const SWLIBS_2Syst_F32 *const	pUAB	**input**	Pointer to the structure with Alpha/Beta voltage components.
tFrac32	f32Velocity	**input**	Estimated electrical angular velocity.
tFrac32	f32Phase	**input**	Estimated rotor flux angle.
AMCLIB_BEMF_OBSRV_DQ_T_F32 *const	pCtrl	**input, output**	Pointer to the structure with BEMF observer parameters and state.

2.2.7.3.3 Implementation details

The scaling of the input values follows the same rules as the corresponding inputs of AMCLIB_BemfObsrvDQ_F32.

2.2.7.3.4 Code Example

```
#include "amclib.h"

SWLIBS_2Syst_F32                mcIab, mcUab;
AMCLIB_BEMF_OBSRV_DQ_T_F32      BemfObsrv;
tFrac32                         f32Velocity;
tFrac32                         f32Phase;

void main (void)
{
    // Clear BEMF observer state variables:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQInit_F32(&BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQInit(&BemfObsrv, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_BemfObsrvDQInit(&BemfObsrv);

    // Set BEMF observer parameters:
    BemfObsrv.pParamD.f32CC1sc      = (tFrac32)1583784859;
    BemfObsrv.pParamD.f32CC2sc      = (tFrac32)-1367421132;
    BemfObsrv.pParamD.f32UpperLimit = (tFrac32)2147483647;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    BemfObsrv.pParamD.f32LowerLimit = (tFrac32)-2147483648;
    BemfObsrv.pParamD.u16NShift     = (tU16)1;
    BemfObsrv.pParamQ.f32CC1sc      = (tFrac32)1583784859;
    BemfObsrv.pParamQ.f32CC2sc      = (tFrac32)-1367421132;
    BemfObsrv.pParamQ.f32UpperLimit = (tFrac32)2147483647;
    BemfObsrv.pParamQ.f32LowerLimit = (tFrac32)-2147483648;
    BemfObsrv.pParamQ.u16NShift     = (tU16)1;
    BemfObsrv.f32IGain  = (tFrac32)1923575400;
    BemfObsrv.f32UGain  = (tFrac32)1954108343;
    BemfObsrv.f32WIGain = (tFrac32)2131606518;
    BemfObsrv.f32EGain  = (tFrac32)1954108343;
    BemfObsrv.s16Shift  = (tS16)-3;

    // Obtain the initial estimate of the rotor position and velocity,
    // measure the alpha/beta voltages and currents
    // (...)

    // Initialize the internal states of the observer to achieve
    // seamless transition from an uncontrolled state of the motor
    // to the full feedback control:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQSetState_F32(&mcIab, &mcUab, f32Velocity,
      f32Phase, &BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQSetState(&mcIab, &mcUab, f32Velocity,
      f32Phase, &BemfObsrv, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_BemfObsrvDQSetState(&mcIab, &mcUab, f32Velocity,
      f32Phase, &BemfObsrv);

    while(1);
}

// Periodical function or interrupt
void ISR(void)
{
    tFrac32 f32PhaseErr;

    // Read the A/D, calculate alpha-beta values, etc.
    // (...)

    // Calculate one iteration of the observer:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f32PhaseErr = AMCLIB_BemfObsrvDQ_F32(&mcIab, &mcUab, f32Velocity,
      f32Phase, &BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f32PhaseErr = AMCLIB_BemfObsrvDQ(&mcIab, &mcUab, f32Velocity,
      f32Phase, &BemfObsrv, F32);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    f32PhaseErr = AMCLIB_BemfObsrvDQ(&mcIab, &mcUab, f32Velocity,
      f32Phase, &BemfObsrv);

    // Pass f32PhaseErr to the #AMCLIB_TrackObsrv_F32
    // (...)
}
```

2.2.7.4 Function AMCLIB_BemfObsrvDQSetState_F16

2.2.7.4.1 Declaration

```
void AMCLIB_BemfObsrvDQSetState_F16(const SWLIBS_2Syst_F16 *const
pIAB, const SWLIBS_2Syst_F16 *const pUAB, tFrac16 f16Velocity,
tFrac16 f16Phase, AMCLIB_BEMF_OBSRV_DQ_T_F16 *const pCtrl);
```

2.2.7.4.2 Arguments

Table 12. AMCLIB_BemfObsrvDQSetState_F16 arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_F16 *const	pIAB	**input**	Pointer to the structure with Alpha/Beta current components.
const SWLIBS_2Syst_F16 *const	pUAB	**input**	Pointer to the structure with Alpha/Beta voltage components.
tFrac16	f16Velocity	**input**	Estimated electrical angular velocity.
tFrac16	f16Phase	**input**	Estimated rotor flux angle.
AMCLIB_BEMF_OBSRV_DQ_T_F16 *const	pCtrl	**input, output**	Pointer to the structure with BEMF observer parameters and state.

2.2.7.4.3 Implementation details

The scaling of the input values follows the same rules as the corresponding inputs of AMCLIB_BemfObsrvDQ_F16.

2.2.7.4.4 Code Example

```
#include "amclib.h"

SWLIBS_2Syst_F16              mcIab, mcUab;
AMCLIB_BEMF_OBSRV_DQ_T_F16    BemfObsrv;
tFrac16                       f16Velocity;
tFrac16                       f16Phase;

void main (void)
{
    // Clear BEMF observer state variables:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQInit_F16(&BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    AMCLIB_BemfObsrvDQInit(&BemfObsrv, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_BemfObsrvDQInit(&BemfObsrv);

    // Set BEMF observer parameters:
    BemfObsrv.pParamD.f16CC1sc      = (tFrac16)24167;
    BemfObsrv.pParamD.f16CC2sc      = (tFrac16)-20865;
    BemfObsrv.pParamD.f16UpperLimit = (tFrac16)32767;
    BemfObsrv.pParamD.f16LowerLimit = (tFrac16)-32768;
    BemfObsrv.pParamD.u16NShift     = (tU16)1;
    BemfObsrv.pParamQ.f16CC1sc      = (tFrac16)24167;
    BemfObsrv.pParamQ.f16CC2sc      = (tFrac16)-20865;
    BemfObsrv.pParamQ.f16UpperLimit = (tFrac16)32767;
    BemfObsrv.pParamQ.f16LowerLimit = (tFrac16)-32768;
    BemfObsrv.pParamQ.u16NShift     = (tU16)1;
    BemfObsrv.f16IGain  = (tFrac16)29351;
    BemfObsrv.f16UGain  = (tFrac16)29817;
    BemfObsrv.f16WIGain = (tFrac16)32526;
    BemfObsrv.f16EGain  = (tFrac16)29817;
    BemfObsrv.s16Shift  = (tS16)-3;

    // Obtain the initial estimate of the rotor position and velocity,
    // measure the alpha/beta voltages and currents
    // (...)

    // Initialize the internal states of the observer to achieve
    // seamless transition from an uncontrolled state of the motor
    // to the full feedback control:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQSetState_F16(&mcIab, &mcUab, f16Velocity,
      f16Phase, &BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQSetState(&mcIab, &mcUab, f16Velocity,
      f16Phase, &BemfObsrv, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_BemfObsrvDQSetState(&mcIab, &mcUab, f16Velocity,
      f16Phase, &BemfObsrv);

    while(1);
}

// Periodical function or interrupt
void ISR(void)
{
    tFrac16 f16PhaseErr;

    // Read the A/D, calculate alpha-beta values, etc.
    // (...)

    // Calculate one iteration of the observer:
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f16PhaseErr = AMCLIB_BemfObsrvDQ_F16(&mcIab, &mcUab, f16Velocity,
      f16Phase, &BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f16PhaseErr = AMCLIB_BemfObsrvDQ(&mcIab, &mcUab, f16Velocity,
      f16Phase, &BemfObsrv, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    f16PhaseErr = AMCLIB_BemfObsrvDQ(&mcIab, &mcUab, f16Velocity,
      f16Phase, &BemfObsrv);

    // Pass f16PhaseErr to the #AMCLIB_TrackObsrv_F16
    // (...)
}
```

2.2.7.5 Function AMCLIB_BemfObsrvDQSetState_FLT

2.2.7.5.1 Declaration

```
void AMCLIB_BemfObsrvDQSetState_FLT(const SWLIBS_2Syst_FLT *const
pIAB, const SWLIBS_2Syst_FLT *const pUAB, tFloat fltVelocity,
tFloat fltPhase, AMCLIB_BEMF_OBSRV_DQ_T_FLT *const pCtrl);
```

2.2.7.5.2 Arguments

Table 13. AMCLIB_BemfObsrvDQSetState_FLT arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_FLT *const	pIAB	**input**	Pointer to the structure with Alpha/Beta current components [A].
const SWLIBS_2Syst_FLT *const	pUAB	**input**	Pointer to the structure with Alpha/Beta voltage components [V].
tFloat	fltVelocity	**input**	Estimated electrical angular velocity [rad/s].
tFloat	fltPhase	**input**	Estimated rotor flux angle [rad], must be in range [-π, π].
AMCLIB_BEMF_OBSRV_DQ_T_FLT *const	pCtrl	**input, output**	Pointer to the structure with BEMF observer parameters and state.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.2.7.5.3 Code Example

```
#include "amclib.h"

#define Ts   (1e-4F)
#define Ksi  (1.0F)
#define w0   (2.0F*3.14F*350.0F)
#define Ld   (3e-4F)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
#define Lq  (3e-4F)
#define Rs  (0.33F)
#define Kp  (2.0F*Ksi*w0*Ld-Rs)
#define Ki  (w0*w0*Ld)

SWLIBS_2Syst_FLT            mcIab, mcUab;
AMCLIB_BEMF_OBSRV_DQ_T_FLT  BemfObsrv;
tFloat                      fltVelocity;
tFloat                      fltPhase;

void main (void)
{
    // Clear BEMF observer state variables:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQInit_FLT(&BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQInit(&BemfObsrv, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if floating point implementation is selected as default.
    AMCLIB_BemfObsrvDQInit(&BemfObsrv);

    // Set BEMF observer parameters:
    BemfObsrv.pParamD.fltCC1sc      =  Kp+Ki*Ts/2.0F;
    BemfObsrv.pParamD.fltCC2sc      = -Kp+Ki*Ts/2.0F;
    BemfObsrv.pParamD.fltUpperLimit =  1000.0F;
    BemfObsrv.pParamD.fltLowerLimit = -1000.0F;
    BemfObsrv.pParamQ.fltCC1sc      =  Kp+Ki*Ts/2.0F;
    BemfObsrv.pParamQ.fltCC2sc      = -Kp+Ki*Ts/2.0F;
    BemfObsrv.pParamQ.fltUpperLimit =  1000.0F;
    BemfObsrv.pParamQ.fltLowerLimit = -1000.0F;
    BemfObsrv.fltIGain  = (2.0F*Ld-Ts*Rs)/(2.0F*Ld+Ts*Rs);
    BemfObsrv.fltUGain  = Ts/(2.0F*Ld+Ts*Rs);
    BemfObsrv.fltWIGain = Ts*Lq/(2.0F*Ld+Ts*Rs);
    BemfObsrv.fltEGain  = Ts/(2.0F*Ld+Ts*Rs);

    // Obtain the initial estimate of the rotor position and velocity,
    // measure the alpha/beta voltages and currents
    // (...)

    // Initialize the internal states of the observer to achieve
    // seamless transition from an uncontrolled state of the motor
    // to the full feedback control:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQSetState_FLT(&mcIab, &mcUab, fltVelocity,
      fltPhase, &BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_BemfObsrvDQSetState(&mcIab, &mcUab, fltVelocity,
      fltPhase, &BemfObsrv,FLT);

    // Alternative 3: API call with global configuration of implementation
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // (only one alternative shall be used). This alternative is available
    // only if floating point implementation is selected as default.
    AMCLIB_BemfObsrvDQSetState(&mcIab, &mcUab, fltVelocity,
      fltPhase, &BemfObsrv);

    while(1);
}

// Periodical function or interrupt
void ISR(void)
{
    tFloat fltPhaseErr;

    // Read the A/D, calculate alpha-beta values, etc.
    // (...)

    // Calculate one iteration of the observer:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    fltPhaseErr = AMCLIB_BemfObsrvDQ_FLT(&mcIab, &mcUab, fltVelocity,
      fltPhase, &BemfObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    fltPhaseErr = AMCLIB_BemfObsrvDQ(&mcIab, &mcUab, fltVelocity,
      fltPhase, &BemfObsrv,FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if floating point implementation is selected as default.
    fltPhaseErr = AMCLIB_BemfObsrvDQ(&mcIab, &mcUab, fltVelocity,
      fltPhase, &BemfObsrv);

    // Pass fltPhaseErr to the #AMCLIB_TrackObsrv_FLT
    // (...)
}
```

2.3 Function AMCLIB_CurrentLoop

This library function implements the current control loop which is an integral part of FOC and represents the most inner loop in the cascade control structure. The current control loop contains two PI controllers in recurrent form.

2.3.1 Description

Current control is essential in a variable-frequency drive control methods known as field-oriented control (FOC). The transformation of the stator phase currents into the orthogonal rotational two-phase system results into two independent current components. The direct (d-axis) current component is known as flux-producing component and the orthogonal (q-axis) component to the magnetic flux is known as torque-producing component. The current control loop is used to control the torque and magnetic flux of the 3-phase motor with high accuracy, dynamics and bandwidth. To achieve this, PI controllers are typically used to control measured current components to their reference values. These reference values are usually given by an outer loop speed and field-weakening controllers.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

AMCLIB_CurrentLoop implements two current PI controllers in recurrent form to control both components of the current vector separately, as highlighted in Figure 41 .

Figure 41. Principal Field Oriented Control scheme with AMCLIB_CurrentLoop

Before using the FOC with a particular motor, the user needs to provide a set of coefficients through the `pCtrl` input pointer. The controller coefficient values can be calculated from motor parameters. A method for measuring the motor parameters is described in PMSM Electrical Parameters Measurement (document AN4680).

Refer to the following resources to find out how the NXP motor control tuning and debugging tools for NXP microcontrollers can help you deploy the FOC in your application:

- AN4642 - Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM
- FREEMASTER - FreeMASTER Run-Time Debugging Tool

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.3.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.3.3 Function AMCLIB_CurrentLoop_F32

2.3.3.1 Declaration

```
void AMCLIB_CurrentLoop_F32(tFrac32 f32UDcBus, SWLIBS_2Syst_F32
*const pUDQReq, AMCLIB_CURRENT_LOOP_T_F32 *pCtrl);
```

2.3.3.2 Arguments

Table 14. AMCLIB_CurrentLoop_F32 arguments

Type	Name	Direction	Description
tFrac32	f32UDcBus	**input**	DC bus voltage.
SWLIBS_2Syst_F32 *const	pUDQReq	**output**	Pointer to the structure with the required stator voltages in the two-phase rotational orthogonal system (d-q).
AMCLIB_CURRENT_LOOP_T_F32 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_CurrentLoop state.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.3.3.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_CurrentLoop_F32.

Figure 42. Functions and data structures in AMCLIB_CurrentLoop_F32

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. All inputs and outputs of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

Table 15. Scaling constants

Scaling constant	Symbol	Calculation
Maximum stator phase voltage [V]	U_{MAX}	$U_{MAX}=U_{DC_Bus_Max}$
Maximum phase current [A]	I_{MAX}	Maximum current of the inverter or nominal current of the motor (whichever is lower).

The current loop contains two controllers (GFLIB_ControllerPIrAW_F32). The proportional gain K_P and integral gain K_I of these controllers can be calculated as follows:

$$K_{pD} = 2 \cdot \xi \cdot \omega_0 \cdot L_D - R_S$$
$$K_{iD} = \omega_0^2 \cdot L_D$$
$$K_{pQ} = 2 \cdot \xi \cdot \omega_0 \cdot L_Q - R_S$$
$$K_{iQ} = \omega_0^2 \cdot L_Q$$

Equation AMCLIB_CurrentLoop_F32_Eq1

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

where ξ is the current loop attenuation, and ω_0 is the current loop natural frequency [rad/s], R_S is the phase resistance [ohm], L_D and L_Q are phase inductances [H] in the d and q axes, respectively.

The continuous-time controller parameters K_P and K_I are then discretized using the bilinear transform. The resulting difference equation coefficients must be scaled so that the input and output values of the controllers fit in the fractional range.

$$CC1D_{SC} = \left(K_{pD} + \frac{K_{iD} T_S}{2}\right) \cdot \frac{I_{MAX}}{U_{MAX}}$$
$$CC2D_{SC} = \left(-K_{pD} + \frac{K_{iD} T_S}{2}\right) \cdot \frac{I_{MAX}}{U_{MAX}}$$
$$CC1Q_{SC} = \left(K_{pQ} + \frac{K_{iQ} T_S}{2}\right) \cdot \frac{I_{MAX}}{U_{MAX}}$$
$$CC2Q_{SC} = \left(-K_{pQ} + \frac{K_{iQ} T_S}{2}\right) \cdot \frac{I_{MAX}}{U_{MAX}}$$

Equation AMCLIB_CurrentLoop_F32_Eq2

T_S is the sampling period [s]. In case the coefficients $CC1D_{SC}$ or $CC2D_{SC}$ exceed the fractional range [-1, 1), their value must be right-shifted by *NShiftD*. Similarly, if $CC1Q_{SC}$ or $CC2Q_{SC}$ exceed the fractional range, their value must be right-shifted by *NShiftQ*. The binary shift amount corresponds to

$$NShiftD = \max\left(0, \left\lceil \log_2\left(\max\left(\left|CC1D_{SC}\right|, \left|CC2D_{SC}\right|\right)\right)\right\rceil\right)$$
$$NShiftQ = \max\left(0, \left\lceil \log_2\left(\max\left(\left|CC1Q_{SC}\right|, \left|CC2Q_{SC}\right|\right)\right)\right\rceil\right)$$

Equation AMCLIB_CurrentLoop_F32_Eq3

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 31st power of two (corresponds to the Q1.31 encoding):

$$pPIrAWD.f32CC1sc = \left(CC1D_{SC} \cdot 2^{-NShiftD}\right) \cdot 2^{31}$$
$$pPIrAWD.f32CC2sc = \left(CC2D_{SC} \cdot 2^{-NShiftD}\right) \cdot 2^{31}$$
$$pPIrAWD.u16NShift = NShiftD$$
$$pPIrAWQ.f32CC1sc = \left(CC1Q_{SC} \cdot 2^{-NShiftQ}\right) \cdot 2^{31}$$
$$pPIrAWQ.f32CC2sc = \left(CC2Q_{SC} \cdot 2^{-NShiftQ}\right) \cdot 2^{31}$$
$$pPIrAWQ.u16NShift = NShiftQ$$

Equation AMCLIB_CurrentLoop_F32_Eq4

The specified accuracy of results is guaranteed only in cases when pCtrl->pPIrAWQ.f32UpperLimit > FRAC32(0.0000305176).

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

2.3.3.4 Code Example

```
#include "amclib.h"
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
AMCLIB_CURRENT_LOOP_T_F32 CurrentLoop;
SWLIBS_2Syst_F32 IDQReq;       // required dq currents
SWLIBS_2Syst_F32 IDQFbck;      // feedback dq currents
SWLIBS_2Syst_F32 UDQReq;       // required dq voltages
tFrac32 f32VelocityReq;        // required velocity
tFrac32 f32VelocityFbck;       // actual velocity
tFrac32 f32UDcBus;             // DC bus voltage

void main (void)
{
    tFrac32 f32ControllerPIrAWDOut;
    tFrac32 f32ControllerPIrAWQOut;

    // Initialize the parameters and pointers in CurrentLoop
    CurrentLoop.pPIrAWD.f32CC1sc      = (tFrac32)FRAC32(0.1345);
    CurrentLoop.pPIrAWD.f32CC2sc      = (tFrac32)FRAC32(0.3498);
    CurrentLoop.pPIrAWD.u16NShift     = 1u;
    CurrentLoop.pPIrAWQ.f32CC1sc      = (tFrac32)FRAC32(0.6432);
    CurrentLoop.pPIrAWQ.f32CC2sc      = (tFrac32)FRAC32(0.2735);
    CurrentLoop.pPIrAWQ.u16NShift     = 1u;
    CurrentLoop.pIDQReq = &IDQReq;
    CurrentLoop.pIDQFbck = &IDQFbck;

    // Clear AMCLIB_CurrentLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopInit_F32(&CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopInit(&CurrentLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_CurrentLoopInit(&CurrentLoop);

    // Initialize the AMCLIB_CurrentLoop state variables to predefined values
    // Warning: Parameters in CurrentLoop must be already initialized.
    f32ControllerPIrAWDOut = (tFrac32)123L;
    f32ControllerPIrAWQOut = (tFrac32)123L;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopSetState_F32(f32ControllerPIrAWDOut,
      f32ControllerPIrAWQOut, &CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopSetState(f32ControllerPIrAWDOut,
      f32ControllerPIrAWQOut, &CurrentLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_CurrentLoopSetState(f32ControllerPIrAWDOut,
      f32ControllerPIrAWQOut, &CurrentLoop);

    f32VelocityReq = (tFrac32)100L;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    while(1);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of IDQFbck, f32VelocityFbck, measure f32UDcBus
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoop_F32(f32UDcBus, &UDQReq, &CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoop(f32UDcBus, &UDQReq, &CurrentLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_CurrentLoop(f32UDcBus, &UDQReq, &CurrentLoop);

    // Calculate new PWM values from UDQReq
    // (...)
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Calculate new values of IDQReq
    // (...)
}
```

2.3.4 Function AMCLIB_CurrentLoop_F16

2.3.4.1 Declaration

```
void AMCLIB_CurrentLoop_F16(tFrac16 f16UDcBus, SWLIBS_2Syst_F16
*const pUDQReq, AMCLIB_CURRENT_LOOP_T_F16 *pCtrl);
```

2.3.4.2 Arguments

Table 16. AMCLIB_CurrentLoop_F16 arguments

Type	Name	Direction	Description
tFrac16	f16UDcBus	**input**	DC bus voltage.
SWLIBS_2Syst_F16 *const	pUDQReq	**output**	Pointer to the structure with the required stator voltages in the two-phase rotational orthogonal system (d-q).
AMCLIB_CURRENT_LOOP_T_F16 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_CurrentLoop state.

2.3.4.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_CurrentLoop_F16.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 43. Functions and data structures in AMCLIB_CurrentLoop_F16

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. All inputs and outputs of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

Table 17. Scaling constants

Scaling constant	Symbol	Calculation
Maximum stator phase voltage [V]	U_{MAX}	$U_{MAX}=U_{DC_Bus_Max}$
Maximum phase current [A]	I_{MAX}	Maximum current of the inverter or nominal current of the motor (whichever is lower).

The current loop contains two controllers (GFLIB_ControllerPIrAW_F16). The proportional gain K_P and integral gain K_I of these controllers can be calculated as follows:

$$K_{pD} = 2 \cdot \xi \cdot \omega_0 \cdot L_D - R_S$$
$$K_{iD} = \omega_0^2 \cdot L_D$$
$$K_{pQ} = 2 \cdot \xi \cdot \omega_0 \cdot L_Q - R_S$$
$$K_{iQ} = \omega_0^2 \cdot L_Q$$

Equation AMCLIB_CurrentLoop_F16_Eq1

where ξ is the current loop attenuation, and ω_0 is the current loop natural frequency [rad/s], R_S is the phase resistance [ohm], L_D and L_Q are phase inductances [H] in the d and q axes, respectively.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The continuous-time controller parameters K_P and K_I are then discretized using the bilinear transform. The resulting difference equation coefficients must be scaled so that the input and output values of the controllers fit in the fractional range.

$$CC1D_{SC} = \left(K_{pD} + \frac{K_{iD} \cdot T_S}{2}\right) \cdot \frac{I_{MAX}}{U_{MAX}}$$
$$CC2D_{SC} = \left(-K_{pD} + \frac{K_{iD} \cdot T_S}{2}\right) \cdot \frac{I_{MAX}}{U_{MAX}}$$
$$CC1Q_{SC} = \left(K_{pQ} + \frac{K_{iQ} \cdot T_S}{2}\right) \cdot \frac{I_{MAX}}{U_{MAX}}$$
$$CC2Q_{SC} = \left(-K_{pQ} + \frac{K_{iQ} \cdot T_S}{2}\right) \cdot \frac{I_{MAX}}{U_{MAX}}$$

Equation AMCLIB_CurrentLoop_F16_Eq2

T_S is the sampling period [s]. In case the coefficients $CC1D_{SC}$ or $CC2D_{SC}$ exceed the fractional range [-1, 1), their value must be right-shifted by *NShiftD*. Similarly, if $CC1Q_{SC}$ or $CC2Q_{SC}$ exceed the fractional range, their value must be right-shifted by *NShiftQ*. The binary shift amount corresponds to

$$NShiftD = \max\left(0, \left\lceil \log_2\left(\max\left(\left|CC1D_{SC}\right|, \left|CC2D_{SC}\right|\right)\right)\right\rceil\right)$$
$$NShiftQ = \max\left(0, \left\lceil \log_2\left(\max\left(\left|CC1Q_{SC}\right|, \left|CC2Q_{SC}\right|\right)\right)\right\rceil\right)$$

Equation AMCLIB_CurrentLoop_F16_Eq3

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 15th power of two (corresponds to the Q1.15 encoding):

$$pPIrAWD.f16CC1sc = \left(CC1D_{SC} \cdot 2^{-NShiftD}\right) \cdot 2^{15}$$
$$pPIrAWD.f16CC2sc = \left(CC2D_{SC} \cdot 2^{-NShiftD}\right) \cdot 2^{15}$$
$$pPIrAWD.u16NShift = NShiftD$$
$$pPIrAWQ.f16CC1sc = \left(CC1Q_{SC} \cdot 2^{-NShiftQ}\right) \cdot 2^{15}$$
$$pPIrAWQ.f16CC2sc = \left(CC2Q_{SC} \cdot 2^{-NShiftQ}\right) \cdot 2^{15}$$
$$pPIrAWQ.u16NShift = NShiftQ$$

Equation AMCLIB_CurrentLoop_F16_Eq4

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

2.3.4.4 Code Example

```
#include "amclib.h"

AMCLIB_CURRENT_LOOP_T_F16 CurrentLoop;
SWLIBS_2Syst_F16 IDQReq;       // required dq currents
SWLIBS_2Syst_F16 IDQFbck;      // feedback dq currents
SWLIBS_2Syst_F16 UDQReq;       // required dq voltages
tFrac16 f16VelocityReq;        // required velocity
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
tFrac16 f16VelocityFbck;       // actual velocity
tFrac16 f16UDcBus;             // DC bus voltage

void main (void)
{
    tFrac16 f16ControllerPIrAWDOut;
    tFrac16 f16ControllerPIrAWQOut;

    // Initialize the parameters and pointers in CurrentLoop
    CurrentLoop.pPIrAWD.f16CC1sc      = (tFrac16)FRAC16(0.1345);
    CurrentLoop.pPIrAWD.f16CC2sc      = (tFrac16)FRAC16(0.3498);
    CurrentLoop.pPIrAWD.u16NShift     = 1u;
    CurrentLoop.pPIrAWQ.f16CC1sc      = (tFrac16)FRAC16(0.6432);
    CurrentLoop.pPIrAWQ.f16CC2sc      = (tFrac16)FRAC16(0.2735);
    CurrentLoop.pPIrAWQ.u16NShift     = 1u;
    CurrentLoop.pIDQReq = &IDQReq;
    CurrentLoop.pIDQFbck = &IDQFbck;

    // Clear AMCLIB_CurrentLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopInit_F16(&CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopInit(&CurrentLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_CurrentLoopInit(&CurrentLoop);

    // Initialize the AMCLIB_CurrentLoop state variables to predefined values
    // Warning: Parameters in CurrentLoop must be already initialized.
    f16ControllerPIrAWDOut = (tFrac16)123;
    f16ControllerPIrAWQOut = (tFrac16)123;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopSetState_F16(f16ControllerPIrAWDOut,
      f16ControllerPIrAWQOut, &CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopSetState(f16ControllerPIrAWDOut,
      f16ControllerPIrAWQOut, &CurrentLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_CurrentLoopSetState(f16ControllerPIrAWDOut,
      f16ControllerPIrAWQOut, &CurrentLoop);

    f16VelocityReq = (tFrac16)100;

    while(1);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Calculate new values of IDQFbck, f16VelocityFbck, measure f16UDcBus
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoop_F16(f16UDcBus, &UDQReq, &CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoop(f16UDcBus, &UDQReq, &CurrentLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_CurrentLoop(f16UDcBus, &UDQReq, &CurrentLoop);

    // Calculate new PWM values from UDQReq
    // (...)
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Calculate new values of IDQReq
    // (...)
}
```

2.3.5 Function AMCLIB_CurrentLoop_FLT

2.3.5.1 Declaration

```
void AMCLIB_CurrentLoop_FLT(tFloat fltUDcBus, SWLIBS_2Syst_FLT
*const pUDQReq, AMCLIB_CURRENT_LOOP_T_FLT *pCtrl);
```

2.3.5.2 Arguments

Table 18. AMCLIB_CurrentLoop_FLT arguments

Type	Name	Direction	Description
tFloat	fltUDcBus	**input**	DC bus voltage.
SWLIBS_2Syst_FLT *const	pUDQReq	**output**	Pointer to the structure with the required stator voltages in the two-phase rotational orthogonal system (d-q).
AMCLIB_CURRENT_LOOP_T_FLT *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_CurrentLoop state.

2.3.5.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_CurrentLoop_FLT.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 44. Functions and data structures in AMCLIB_CurrentLoop_FLT

The current loop contains two controllers (GFLIB_ControllerPIrAW_FLT). The proportional gain K_P and integral gain K_I of these controllers can be calculated as follows:

$$K_{pD} = 2 \cdot \xi \cdot \omega_0 \cdot L_D - R_S$$
$$K_{iD} = \omega_0^2 \cdot L_D$$
$$K_{pQ} = 2 \cdot \xi \cdot \omega_0 \cdot L_Q - R_S$$
$$K_{iQ} = \omega_0^2 \cdot L_Q$$

Equation AMCLIB_CurrentLoop_FLT_Eq1

where ξ is the current loop attenuation, and ω_0 is the current loop natural frequency [rad/s], R_S is the phase resistance [ohm], L_D and L_Q are phase inductances [H] in the d and q axes, respectively.

The continuous-time controller parameters K_P and K_I are then discretized using the bilinear transform.

$$pPIrAWD.fltCC1sc = K_{pD} + \frac{K_{iD} \cdot T_S}{2}$$
$$pPIrAWD.fltCC2sc = -K_{pD} + \frac{K_{iD} \cdot T_S}{2}$$
$$pPIrAWQ.fltCC1sc = K_{pQ} + \frac{K_{iQ} \cdot T_S}{2}$$
$$pPIrAWQ.fltCC2sc = -K_{pQ} + \frac{K_{iQ} \cdot T_S}{2}$$

Equation AMCLIB_CurrentLoop_FLT_Eq2

T_S is the sampling period [s].

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.3.5.4 Code Example

```
#include "amclib.h"

AMCLIB_CURRENT_LOOP_T_FLT CurrentLoop;
SWLIBS_2Syst_FLT IDQReq;       // required dq currents
SWLIBS_2Syst_FLT IDQFbck;      // feedback dq currents
SWLIBS_2Syst_FLT UDQReq;       // required dq voltages
tFloat fltVelocityReq;         // required velocity
tFloat fltVelocityFbck;        // actual velocity
tFloat fltUDcBus;              // DC bus voltage

void main (void)
{
    tFloat fltControllerPIrAWDOut;
    tFloat fltControllerPIrAWQOut;

    // Initialize the parameters and pointers in CurrentLoop
    CurrentLoop.pPIrAWD.fltCC1sc      = (tFloat)0.1345;
    CurrentLoop.pPIrAWD.fltCC2sc      = (tFloat)0.3498;
    CurrentLoop.pPIrAWQ.fltCC1sc      = (tFloat)0.6432;
    CurrentLoop.pPIrAWQ.fltCC2sc      = (tFloat)0.2735;
    CurrentLoop.pIDQReq = &IDQReq;
    CurrentLoop.pIDQFbck = &IDQFbck;

    // Clear AMCLIB_CurrentLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopInit_FLT(&CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopInit(&CurrentLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if floating point implementation is selected as default.
    AMCLIB_CurrentLoopInit(&CurrentLoop);

    // Initialize the AMCLIB_CurrentLoop state variables to predefined values
    // Warning: Parameters in CurrentLoop must be already initialized.
    fltControllerPIrAWDOut = 123.0F;
    fltControllerPIrAWQOut = 123.0F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopSetState_FLT(fltControllerPIrAWDOut,
      fltControllerPIrAWQOut, &CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopSetState(fltControllerPIrAWDOut,
      fltControllerPIrAWQOut, &CurrentLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // only if floating point implementation is selected as default.
    AMCLIB_CurrentLoopSetState(fltControllerPIrAWDOut,
      fltControllerPIrAWQOut, &CurrentLoop);

    fltVelocityReq = 123.0F;
    while(1);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of IDQFbck, fltVelocityFbck, measure fltUDcBus
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoop_FLT(fltUDcBus, &UDQReq, &CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoop(fltUDcBus, &UDQReq, &CurrentLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if floating point implementation is selected as default.
    AMCLIB_CurrentLoop(fltUDcBus, &UDQReq, &CurrentLoop);

    // Calculate new PWM values from UDQReq
    // (...)
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Calculate new values of IDQReq
    // (...)
}
```

2.3.6 Function AMCLIB_CurrentLoopInit

2.3.6.1 Description

This function clears the AMCLIB_CurrentLoop state variables.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.3.6.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.3.6.3 Function AMCLIB_CurrentLoopInit_F32

2.3.6.3.1 Declaration

```
void AMCLIB_CurrentLoopInit_F32(AMCLIB_CURRENT_LOOP_T_F32 *const
pCtrl);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.3.6.3.2 Arguments

Table 19. AMCLIB_CurrentLoopInit_F32 arguments

Type	Name	Direction	Description
AMCLIB_CURRENT_LOOP_T_F32 *const	pCtrl	**input, output**	Pointer to the structure with AMCLIB_CurrentLoop state.

2.3.6.3.3 Code Example

```
#include "amclib.h"

AMCLIB_CURRENT_LOOP_T_F32 CurrentLoop;
SWLIBS_2Syst_F32 IDQReq;       // required dq currents
SWLIBS_2Syst_F32 IDQFbck;      // feedback dq currents
SWLIBS_2Syst_F32 UDQReq;       // required dq voltages
tFrac32 f32VelocityReq;        // required velocity
tFrac32 f32VelocityFbck;       // actual velocity
tFrac32 f32UDcBus;             // DC bus voltage

void main (void)
{
    tFrac32 f32ControllerPIrAWDOut;
    tFrac32 f32ControllerPIrAWQOut;

    // Initialize the parameters and pointers in CurrentLoop
    CurrentLoop.pPIrAWD.f32CC1sc      = (tFrac32)FRAC32(0.1345);
    CurrentLoop.pPIrAWD.f32CC2sc      = (tFrac32)FRAC32(0.3498);
    CurrentLoop.pPIrAWD.u16NShift     = 1u;
    CurrentLoop.pPIrAWQ.f32CC1sc      = (tFrac32)FRAC32(0.6432);
    CurrentLoop.pPIrAWQ.f32CC2sc      = (tFrac32)FRAC32(0.2735);
    CurrentLoop.pPIrAWQ.u16NShift     = 1u;
    CurrentLoop.pIDQReq = &IDQReq;
    CurrentLoop.pIDQFbck = &IDQFbck;

    // Clear AMCLIB_CurrentLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopInit_F32(&CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopInit(&CurrentLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_CurrentLoopInit(&CurrentLoop);

    // Initialize the AMCLIB_CurrentLoop state variables to predefined values
    // Warning: Parameters in CurrentLoop must be already initialized.
    f32ControllerPIrAWDOut = (tFrac32)123L;
    f32ControllerPIrAWQOut = (tFrac32)123L;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopSetState_F32(f32ControllerPIrAWDOut,
      f32ControllerPIrAWQOut, &CurrentLoop);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopSetState(f32ControllerPIrAWDOut,
      f32ControllerPIrAWQOut, &CurrentLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_CurrentLoopSetState(f32ControllerPIrAWDOut,
      f32ControllerPIrAWQOut, &CurrentLoop);

    f32VelocityReq = (tFrac32)100L;

    while(1);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of IDQFbck, f32VelocityFbck, measure f32UDcBus
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoop_F32(f32UDcBus, &UDQReq, &CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoop(f32UDcBus, &UDQReq, &CurrentLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_CurrentLoop(f32UDcBus, &UDQReq, &CurrentLoop);

    // Calculate new PWM values from UDQReq
    // (...)
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Calculate new values of IDQReq
    // (...)
}
```

2.3.6.4 Function AMCLIB_CurrentLoopInit_F16

2.3.6.4.1 Declaration

```
void AMCLIB_CurrentLoopInit_F16(AMCLIB_CURRENT_LOOP_T_F16 *const
pCtrl);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.3.6.4.2 Arguments

Table 20. AMCLIB_CurrentLoopInit_F16 arguments

Type	Name	Direction	Description
AMCLIB_CURRENT_LOOP_T_F16 *const	pCtrl	**input, output**	Pointer to the structure with AMCLIB_CurrentLoop state.

2.3.6.4.3 Code Example

```
#include "amclib.h"

AMCLIB_CURRENT_LOOP_T_F16 CurrentLoop;
SWLIBS_2Syst_F16 IDQReq;       // required dq currents
SWLIBS_2Syst_F16 IDQFbck;      // feedback dq currents
SWLIBS_2Syst_F16 UDQReq;       // required dq voltages
tFrac16 f16VelocityReq;        // required velocity
tFrac16 f16VelocityFbck;       // actual velocity
tFrac16 f16UDcBus;             // DC bus voltage

void main (void)
{
    tFrac16 f16ControllerPIrAWDOut;
    tFrac16 f16ControllerPIrAWQOut;

    // Initialize the parameters and pointers in CurrentLoop
    CurrentLoop.pPIrAWD.f16CC1sc      = (tFrac16)FRAC16(0.1345);
    CurrentLoop.pPIrAWD.f16CC2sc      = (tFrac16)FRAC16(0.3498);
    CurrentLoop.pPIrAWD.u16NShift     = 1u;
    CurrentLoop.pPIrAWQ.f16CC1sc      = (tFrac16)FRAC16(0.6432);
    CurrentLoop.pPIrAWQ.f16CC2sc      = (tFrac16)FRAC16(0.2735);
    CurrentLoop.pPIrAWQ.u16NShift     = 1u;
    CurrentLoop.pIDQReq = &IDQReq;
    CurrentLoop.pIDQFbck = &IDQFbck;

    // Clear AMCLIB_CurrentLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopInit_F16(&CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopInit(&CurrentLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_CurrentLoopInit(&CurrentLoop);

    // Initialize the AMCLIB_CurrentLoop state variables to predefined values
    // Warning: Parameters in CurrentLoop must be already initialized.
    f16ControllerPIrAWDOut = (tFrac16)123;
    f16ControllerPIrAWQOut = (tFrac16)123;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopSetState_F16(f16ControllerPIrAWDOut,
      f16ControllerPIrAWQOut, &CurrentLoop);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopSetState(f16ControllerPIrAWDOut,
      f16ControllerPIrAWQOut, &CurrentLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_CurrentLoopSetState(f16ControllerPIrAWDOut,
      f16ControllerPIrAWQOut, &CurrentLoop);

    f16VelocityReq = (tFrac16)100;

    while(1);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of IDQFbck, f16VelocityFbck, measure f16UDcBus
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoop_F16(f16UDcBus, &UDQReq, &CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoop(f16UDcBus, &UDQReq, &CurrentLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_CurrentLoop(f16UDcBus, &UDQReq, &CurrentLoop);

    // Calculate new PWM values from UDQReq
    // (...)
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Calculate new values of IDQReq
    // (...)
}
```

2.3.6.5 Function AMCLIB_CurrentLoopInit_FLT

2.3.6.5.1 Declaration

```
void AMCLIB_CurrentLoopInit_FLT(AMCLIB_CURRENT_LOOP_T_FLT *const
pCtrl);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.3.6.5.2 Arguments

Table 21. AMCLIB_CurrentLoopInit_FLT arguments

Type	Name	Direction	Description
AMCLIB_CURRENT_LOOP_T_FLT *const	pCtrl	**input, output**	Pointer to the structure with AMCLIB_CurrentLoop state.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.3.6.5.3 Code Example

```
#include "amclib.h"

AMCLIB_CURRENT_LOOP_T_FLT CurrentLoop;
SWLIBS_2Syst_FLT IDQReq;       // required dq currents
SWLIBS_2Syst_FLT IDQFbck;      // feedback dq currents
SWLIBS_2Syst_FLT UDQReq;       // required dq voltages
tFloat fltVelocityReq;         // required velocity
tFloat fltVelocityFbck;        // actual velocity
tFloat fltUDcBus;              // DC bus voltage

void main (void)
{
    tFloat fltControllerPIrAWDOut;
    tFloat fltControllerPIrAWQOut;

    // Initialize the parameters and pointers in CurrentLoop
    CurrentLoop.pPIrAWD.fltCC1sc      = (tFloat)0.1345;
    CurrentLoop.pPIrAWD.fltCC2sc      = (tFloat)0.3498;
    CurrentLoop.pPIrAWQ.fltCC1sc      = (tFloat)0.6432;
    CurrentLoop.pPIrAWQ.fltCC2sc      = (tFloat)0.2735;
    CurrentLoop.pIDQReq = &IDQReq;
    CurrentLoop.pIDQFbck = &IDQFbck;

    // Clear AMCLIB_CurrentLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopInit_FLT(&CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopInit(&CurrentLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if floating point implementation is selected as default.
    AMCLIB_CurrentLoopInit(&CurrentLoop);

    // Initialize the AMCLIB_CurrentLoop state variables to predefined values
    // Warning: Parameters in CurrentLoop must be already initialized.
    fltControllerPIrAWDOut = 123.0F;
    fltControllerPIrAWQOut = 123.0F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopSetState_FLT(fltControllerPIrAWDOut,
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
      fltControllerPIrAWQOut, &CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopSetState(fltControllerPIrAWDOut,
      fltControllerPIrAWQOut, &CurrentLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if floating point implementation is selected as default.
    AMCLIB_CurrentLoopSetState(fltControllerPIrAWDOut,
      fltControllerPIrAWQOut, &CurrentLoop);

    fltVelocityReq = 123.0F;
    while(1);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of IDQFbck, fltVelocityFbck, measure fltUDcBus
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoop_FLT(fltUDcBus, &UDQReq, &CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoop(fltUDcBus, &UDQReq, &CurrentLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if floating point implementation is selected as default.
    AMCLIB_CurrentLoop(fltUDcBus, &UDQReq, &CurrentLoop);

    // Calculate new PWM values from UDQReq
    // (...)
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Calculate new values of IDQReq
    // (...)
}
```

2.3.7 Function AMCLIB_CurrentLoopSetState

2.3.7.1 Description

This function initializes the AMCLIB_CurrentLoop state variables to achieve the required output values.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.3.7.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.3.7.3 Function AMCLIB_CurrentLoopSetState_F32

2.3.7.3.1 Declaration

```
void AMCLIB_CurrentLoopSetState_F32(tFrac32
f32ControllerPIrAWDOut, tFrac32 f32ControllerPIrAWQOut,
AMCLIB_CURRENT_LOOP_T_F32 *pCtrl);
```

2.3.7.3.2 Arguments

Table 22. AMCLIB_CurrentLoopSetState_F32 arguments

Type	Name	Direction	Description
tFrac32	f32ControllerPIrAWDOut	**output**	Required output of the d-axis ControllerPIrAW.
tFrac32	f32ControllerPIrAWQOut	**output**	Required output of the q-axis ControllerPIrAW.
AMCLIB_CURRENT_LOOP_T_F32 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_CurrentLoop state.

CAUTION: *Set the parameters in the structure pointed to by pCtrl before calling this function.*

2.3.7.3.3 Code Example

```
#include "amclib.h"

AMCLIB_CURRENT_LOOP_T_F32 CurrentLoop;
SWLIBS_2Syst_F32 IDQReq;       // required dq currents
SWLIBS_2Syst_F32 IDQFbck;      // feedback dq currents
SWLIBS_2Syst_F32 UDQReq;       // required dq voltages
tFrac32 f32VelocityReq;        // required velocity
tFrac32 f32VelocityFbck;       // actual velocity
tFrac32 f32UDcBus;             // DC bus voltage

void main (void)
{
    tFrac32 f32ControllerPIrAWDOut;
    tFrac32 f32ControllerPIrAWQOut;

    // Initialize the parameters and pointers in CurrentLoop
    CurrentLoop.pPIrAWD.f32CC1sc      = (tFrac32)FRAC32(0.1345);
    CurrentLoop.pPIrAWD.f32CC2sc      = (tFrac32)FRAC32(0.3498);
    CurrentLoop.pPIrAWD.u16NShift     = 1u;
    CurrentLoop.pPIrAWQ.f32CC1sc      = (tFrac32)FRAC32(0.6432);
    CurrentLoop.pPIrAWQ.f32CC2sc      = (tFrac32)FRAC32(0.2735);
    CurrentLoop.pPIrAWQ.u16NShift     = 1u;
    CurrentLoop.pIDQReq = &IDQReq;
    CurrentLoop.pIDQFbck = &IDQFbck;

    // Clear AMCLIB_CurrentLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopInit_F32(&CurrentLoop);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopInit(&CurrentLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_CurrentLoopInit(&CurrentLoop);

    // Initialize the AMCLIB_CurrentLoop state variables to predefined values
    // Warning: Parameters in CurrentLoop must be already initialized.
    f32ControllerPIrAWDOut = (tFrac32)123L;
    f32ControllerPIrAWQOut = (tFrac32)123L;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopSetState_F32(f32ControllerPIrAWDOut,
      f32ControllerPIrAWQOut, &CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopSetState(f32ControllerPIrAWDOut,
      f32ControllerPIrAWQOut, &CurrentLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_CurrentLoopSetState(f32ControllerPIrAWDOut,
      f32ControllerPIrAWQOut, &CurrentLoop);

    f32VelocityReq = (tFrac32)100L;

    while(1);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of IDQFbck, f32VelocityFbck, measure f32UDcBus
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoop_F32(f32UDcBus, &UDQReq, &CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoop(f32UDcBus, &UDQReq, &CurrentLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_CurrentLoop(f32UDcBus, &UDQReq, &CurrentLoop);

    // Calculate new PWM values from UDQReq
    // (...)
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Calculate new values of IDQReq
    // (...)
}
```

2.3.7.4 Function AMCLIB_CurrentLoopSetState_F16

2.3.7.4.1 Declaration

```
void AMCLIB_CurrentLoopSetState_F16(tFrac16
f16ControllerPIrAWDOut, tFrac16 f16ControllerPIrAWQOut,
AMCLIB_CURRENT_LOOP_T_F16 *pCtrl);
```

2.3.7.4.2 Arguments

Table 23. AMCLIB_CurrentLoopSetState_F16 arguments

Type	Name	Direction	Description
tFrac16	f16ControllerPIrAWDOut	**output**	Required output of the d-axis ControllerPIrAW.
tFrac16	f16ControllerPIrAWQOut	**output**	Required output of the q-axis ControllerPIrAW.
AMCLIB_CURRENT_LOOP_T_F16 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_CurrentLoop state.

CAUTION: *Set the parameters in the structure pointed to by pCtrl before calling this function.*

2.3.7.4.3 Code Example

```
#include "amclib.h"

AMCLIB_CURRENT_LOOP_T_F16 CurrentLoop;
SWLIBS_2Syst_F16 IDQReq;       // required dq currents
SWLIBS_2Syst_F16 IDQFbck;      // feedback dq currents
SWLIBS_2Syst_F16 UDQReq;       // required dq voltages
tFrac16 f16VelocityReq;        // required velocity
tFrac16 f16VelocityFbck;       // actual velocity
tFrac16 f16UDcBus;             // DC bus voltage

void main (void)
{
    tFrac16 f16ControllerPIrAWDOut;
    tFrac16 f16ControllerPIrAWQOut;

    // Initialize the parameters and pointers in CurrentLoop
    CurrentLoop.pPIrAWD.f16CC1sc      = (tFrac16)FRAC16(0.1345);
    CurrentLoop.pPIrAWD.f16CC2sc      = (tFrac16)FRAC16(0.3498);
    CurrentLoop.pPIrAWD.u16NShift     = 1u;
    CurrentLoop.pPIrAWQ.f16CC1sc      = (tFrac16)FRAC16(0.6432);
    CurrentLoop.pPIrAWQ.f16CC2sc      = (tFrac16)FRAC16(0.2735);
    CurrentLoop.pPIrAWQ.u16NShift     = 1u;
    CurrentLoop.pIDQReq = &IDQReq;
    CurrentLoop.pIDQFbck = &IDQFbck;

    // Clear AMCLIB_CurrentLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopInit_F16(&CurrentLoop);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopInit(&CurrentLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_CurrentLoopInit(&CurrentLoop);

    // Initialize the AMCLIB_CurrentLoop state variables to predefined values
    // Warning: Parameters in CurrentLoop must be already initialized.
    f16ControllerPIrAWDOut = (tFrac16)123;
    f16ControllerPIrAWQOut = (tFrac16)123;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopSetState_F16(f16ControllerPIrAWDOut,
      f16ControllerPIrAWQOut, &CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopSetState(f16ControllerPIrAWDOut,
      f16ControllerPIrAWQOut, &CurrentLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_CurrentLoopSetState(f16ControllerPIrAWDOut,
      f16ControllerPIrAWQOut, &CurrentLoop);

    f16VelocityReq = (tFrac16)100;

    while(1);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of IDQFbck, f16VelocityFbck, measure f16UDcBus
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoop_F16(f16UDcBus, &UDQReq, &CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoop(f16UDcBus, &UDQReq, &CurrentLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_CurrentLoop(f16UDcBus, &UDQReq, &CurrentLoop);

    // Calculate new PWM values from UDQReq
    // (...)
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Calculate new values of IDQReq
    // (...)
}
```

2.3.7.5 Function AMCLIB_CurrentLoopSetState_FLT

2.3.7.5.1 Declaration

```
void AMCLIB_CurrentLoopSetState_FLT(tFloat
fltControllerPIrAWDOut, tFloat fltControllerPIrAWQOut,
AMCLIB_CURRENT_LOOP_T_FLT *pCtrl);
```

2.3.7.5.2 Arguments

Table 24. AMCLIB_CurrentLoopSetState_FLT arguments

Type	Name	Direction	Description
tFloat	fltControllerPIrAWDOut	**output**	Required output of the d-axis ControllerPIrAW.
tFloat	fltControllerPIrAWQOut	**output**	Required output of the q-axis ControllerPIrAW.
AMCLIB_CURRENT_LOOP_T_FLT *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_CurrentLoop state.

CAUTION: *Set the parameters in the structure pointed to by pCtrl before calling this function.*

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.3.7.5.3 Code Example

```
#include "amclib.h"

AMCLIB_CURRENT_LOOP_T_FLT CurrentLoop;
SWLIBS_2Syst_FLT IDQReq;       // required dq currents
SWLIBS_2Syst_FLT IDQFbck;      // feedback dq currents
SWLIBS_2Syst_FLT UDQReq;       // required dq voltages
tFloat fltVelocityReq;         // required velocity
tFloat fltVelocityFbck;        // actual velocity
tFloat fltUDcBus;              // DC bus voltage

void main (void)
{
    tFloat fltControllerPIrAWDOut;
    tFloat fltControllerPIrAWQOut;

    // Initialize the parameters and pointers in CurrentLoop
    CurrentLoop.pPIrAWD.fltCC1sc      = (tFloat)0.1345;
    CurrentLoop.pPIrAWD.fltCC2sc      = (tFloat)0.3498;
    CurrentLoop.pPIrAWQ.fltCC1sc      = (tFloat)0.6432;
    CurrentLoop.pPIrAWQ.fltCC2sc      = (tFloat)0.2735;
    CurrentLoop.pIDQReq = &IDQReq;
    CurrentLoop.pIDQFbck = &IDQFbck;

    // Clear AMCLIB_CurrentLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    AMCLIB_CurrentLoopInit_FLT(&CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopInit(&CurrentLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if floating point implementation is selected as default.
    AMCLIB_CurrentLoopInit(&CurrentLoop);

    // Initialize the AMCLIB_CurrentLoop state variables to predefined values
    // Warning: Parameters in CurrentLoop must be already initialized.
    fltControllerPIrAWDOut = 123.0F;
    fltControllerPIrAWQOut = 123.0F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopSetState_FLT(fltControllerPIrAWDOut,
      fltControllerPIrAWQOut, &CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoopSetState(fltControllerPIrAWDOut,
      fltControllerPIrAWQOut, &CurrentLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if floating point implementation is selected as default.
    AMCLIB_CurrentLoopSetState(fltControllerPIrAWDOut,
      fltControllerPIrAWQOut, &CurrentLoop);

    fltVelocityReq = 123.0F;
    while(1);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of IDQFbck, fltVelocityFbck, measure fltUDcBus
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_CurrentLoop_FLT(fltUDcBus, &UDQReq, &CurrentLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_CurrentLoop(fltUDcBus, &UDQReq, &CurrentLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if floating point implementation is selected as default.
    AMCLIB_CurrentLoop(fltUDcBus, &UDQReq, &CurrentLoop);

    // Calculate new PWM values from UDQReq
    // (...)
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
{
    // Calculate new values of IDQReq
    // (...)
}
```

2.4 Function AMCLIB_CurrentLoopPIp

This library function implements the current control loop which is an integral part of FOC and represents the most inner loop in the cascade control structure. The current control loop contains two PI controllers in parallel form.

2.4.1 Description

Current control is essential in a variable-frequency drive control methods known as field-oriented control (FOC). The transformation of the stator phase currents into the orthogonal rotational two-phase system results into two independent current components. The direct (d-axis) current component is known as flux-producing component and the orthogonal (q-axis) component to the magnetic flux is known as torque-producing component. The current control loop is used to control the torque and magnetic flux of the 3-phase motor with high accuracy, dynamics and bandwidth. To achieve this, PI controllers are typically used to control measured current components to their reference values. These reference values are usually given by an outer loop speed and field-weakening controllers.

AMCLIB_CurrentLoopPIp implements two current PI controllers in parallel form to control both components of the current vector separately, as highlighted in Figure 45 .

Figure 45. Principal Field Oriented Control scheme with AMCLIB_CurrentLoopPIp

Before using the FOC with a particular motor, the user needs to provide a set of coefficients through the `pCtrl` input pointer. The controller coefficient values can be calculated from motor parameters. A method for measuring the motor parameters is described in PMSM Electrical Parameters Measurement (document AN4680).

Refer to the following resources to find out how the NXP motor control tuning and debugging tools for NXP microcontrollers can help you deploy the FOC in your application:

- AN4642 - Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM
- FREEMASTER - FreeMASTER Run-Time Debugging Tool

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.4.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.4.3 Function AMCLIB_CurrentLoopPIp_F32

2.4.3.1 Declaration

```
void AMCLIB_CurrentLoopPIp_F32(tFrac32 f32UDcBus,
SWLIBS_2Syst_F32 *const pUDQReq, AMCLIB_CURRENT_LOOP_PIP_T_F32
*pCtrl);
```

2.4.3.2 Arguments

Table 25. AMCLIB_CurrentLoopPIp_F32 arguments

Type	Name	Direction	Description
tFrac32	f32UDcBus	**input**	DC bus voltage.
SWLIBS_2Syst_F32 *const	pUDQReq	**output**	Pointer to the structure with the required stator voltages in the two-phase rotational orthogonal system (d-q).
AMCLIB_CURRENT_LOOP_PIP_T_F32 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_CurrentLoopPIp state.

2.4.3.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_CurrentLoopPIp_F32.

Figure 46. Functions and data structures in AMCLIB_CurrentLoopPIp_F32

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. All inputs and outputs of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

Table 26. Scaling constants

Scaling constant	Symbol	Calculation
Maximum stator phase voltage [V]	U_{MAX}	$U_{MAX}=U_{DC_Bus_Max}$
Maximum phase current [A]	I_{MAX}	Maximum current of the inverter or nominal current of the motor (whichever is lower).

The current loop contains two controllers (GFLIB_ControllerPIpAW_F32). The proportional gain K_P and integral gain K_I of these controllers can be calculated as follows:

$$K_{pD} = 2 \cdot \xi \cdot \omega_0 \cdot L_D - R_S$$
$$K_{iD} = \omega_0^2 \cdot L_D$$
$$K_{pQ} = 2 \cdot \xi \cdot \omega_0 \cdot L_Q - R_S$$
$$K_{iQ} = \omega_0^2 \cdot L_Q$$

Equation AMCLIB_CurrentLoopPIp_F32_Eq1

where ξ is the current loop attenuation, and ω_0 is the current loop natural frequency [rad/s], R_S is the phase resistance [ohm], L_D and L_Q are phase inductances [H] in the d and q axes, respectively.

The continuous-time controller parameters K_P and K_I are then discretized using the bilinear transform. The resulting difference equation coefficients must be scaled so that the input and output values of the controllers fit in the fractional range.

$$K_{pD_{SC}} = K_{pD} \cdot \frac{I_{MAX}}{U_{MAX}}$$
$$K_{iD_{SC}} = \frac{K_{iD} \cdot T_S}{2} \cdot \frac{I_{MAX}}{U_{MAX}}$$
$$K_{pQ_{SC}} = K_{pQ} \cdot \frac{I_{MAX}}{U_{MAX}}$$
$$K_{iQ_{SC}} = \frac{K_{iQ} \cdot T_S}{2} \cdot \frac{I_{MAX}}{U_{MAX}}$$

Equation AMCLIB_CurrentLoopPIp_F32_Eq2

T_S is the sampling period [s]. In case the coefficients $K_{pD\ SC}$, $K_{iD\ SC}$, $K_{pQ\ SC}$ or $K_{piQ\ SC}$ exceed the fractional range [-1, 1), or they are extremely small (which could lead to loss of precision), their value must be bit-shifted by following coefficients

$$pPIpAWDs16PropGainShift = \lceil \log_2(|K_{pD_{SC}}|) \rceil$$
$$pPIpAWDs16IntegGainShift = \lceil \log_2(|K_{iD_{SC}}|) \rceil$$
$$pPIpAWQs16PropGainShift = \lceil \log_2(|K_{pQ_{SC}}|) \rceil$$
$$pPIpAWQs16IntegGainShift = \lceil \log_2(|K_{iQ_{SC}}|) \rceil$$

Equation AMCLIB_CurrentLoopPIp_F32_Eq3

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 31st power of two (corresponds to the Q1.31 encoding):

$$pPIpAWD.f32PropGain = \left(K_{pD_{SC}} \cdot 2^{-pPIpAWD.s16PropGainShift}\right) \cdot 2^{31}$$

$$pPIpAWD.f32IntegGain = \left(K_{iD_{SC}} \cdot 2^{-pPIpAWD.s16IntegGainShift}\right) \cdot 2^{31}$$

$$pPIpAWQ.f32PropGain = \left(K_{pQ_{SC}} \cdot 2^{-pPIpAWQ.s16PropGainShift}\right) \cdot 2^{31}$$

$$pPIpAWQ.f32IntegGain = \left(K_{iQ_{SC}} \cdot 2^{-pPIpAWQ.s16IntegGainShift}\right) \cdot 2^{31}$$

Equation AMCLIB_CurrentLoopPIp_F32_Eq4

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

2.4.3.4 Code Example

```
#include "AMCLIB_CurrentLoopPIp.h"

AMCLIB_CURRENT_LOOP_PIP_T_F32 CurrentLoop;
SWLIBS_2Syst_F32 IDQReq;       // required dq currents
SWLIBS_2Syst_F32 IDQFbck;      // feedback dq currents
SWLIBS_2Syst_F32 UDQReq;       // required dq voltages
tFrac32 f32UDcBus;             // DC bus voltage

// Initialize the parameters and pointers in CurrentLoop
CurrentLoop.pPIpAWD.f32PropGain       = FRAC32(0.1345);
CurrentLoop.pPIpAWD.f32IntegGain      = FRAC32(0.3498);
CurrentLoop.pPIpAWD.s16PropGainShift  = (tS16)1;
CurrentLoop.pPIpAWD.s16IntegGainShift = (tS16)2;
CurrentLoop.pPIpAWQ.f32PropGain       = FRAC32(0.6432);
CurrentLoop.pPIpAWQ.f32IntegGain      = FRAC32(0.2735);
CurrentLoop.pPIpAWQ.s16PropGainShift  = (tS16)1;
CurrentLoop.pPIpAWQ.s16IntegGainShift = (tS16)2;
CurrentLoop.pIDQReq                   = &IDQReq;
CurrentLoop.pIDQFbck                  = &IDQFbck;

// Alternative 1: API call with postfix
// (only one alternative shall be used).
AMCLIB_CurrentLoopPIp_F32(f32UDcBus, &UDQReq, &CurrentLoop);

// Alternative 2: API call with implementation parameter
// (only one alternative shall be used).
AMCLIB_CurrentLoopPIp(f32UDcBus, &UDQReq, &CurrentLoop, F32);

// Alternative 3: API call with global configuration of implementation
// (only one alternative shall be used). This alternative is available
// only if 32-bit fractional implementation is selected as default.
AMCLIB_CurrentLoopPIp(f32UDcBus, &UDQReq, &CurrentLoop);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.4.4 Function AMCLIB_CurrentLoopPIp_F16

2.4.4.1 Declaration

```
void AMCLIB_CurrentLoopPIp_F16(tFrac16 f16UDcBus,
SWLIBS_2Syst_F16 *const pUDQReq, AMCLIB_CURRENT_LOOP_PIP_T_F16
*pCtrl);
```

2.4.4.2 Arguments

Table 27. AMCLIB_CurrentLoopPIp_F16 arguments

Type	Name	Direction	Description
tFrac16	f16UDcBus	**input**	DC bus voltage.
SWLIBS_2Syst_F16 *const	pUDQReq	**output**	Pointer to the structure with the required stator voltages in the two-phase rotational orthogonal system (d-q).
AMCLIB_CURRENT_LOOP_PIP_T_F16 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_CurrentLoopPIp state.

2.4.4.3 Return

Add description, remove if void.

2.4.4.4 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_CurrentLoopPIp_F16.

Figure 47. Functions and data structures in AMCLIB_CurrentLoopPIp_F16

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. All inputs and outputs of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

Table 28. Scaling constants

Scaling constant	Symbol	Calculation
Maximum stator phase voltage [V]	U_{MAX}	$U_{MAX}=U_{DC_Bus_Max}$
Maximum phase current [A]	I_{MAX}	Maximum current of the inverter or nominal current of the motor (whichever is lower).

The current loop contains two controllers (GFLIB_ControllerPIpAW_F16). The proportional gain K_P and integral gain K_I of these controllers can be calculated as follows:

$$K_{pD} = 2 \cdot \xi \cdot \omega_0 \cdot L_D - R_S$$
$$K_{iD} = \omega_0^2 \cdot L_D$$
$$K_{pQ} = 2 \cdot \xi \cdot \omega_0 \cdot L_Q - R_S$$
$$K_{iQ} = \omega_0^2 \cdot L_Q$$

Equation AMCLIB_CurrentLoopPIp_F16_Eq1

where ξ is the current loop attenuation, and ω_0 is the current loop natural frequency [rad/s], R_S is the phase resistance [ohm], L_D and L_Q are phase inductances [H] in the d and q axes, respectively.

The continuous-time controller parameters K_P and K_I are then discretized using the bilinear transform. The resulting difference equation coefficients must be scaled so that the input and output values of the controllers fit in the fractional range.

$$K_{pD_{SC}} = K_{pD} \cdot \frac{I_{MAX}}{U_{MAX}}$$
$$K_{iD_{SC}} = \frac{K_{iD} \cdot T_S}{2} \cdot \frac{I_{MAX}}{U_{MAX}}$$
$$K_{pQ_{SC}} = K_{pQ} \cdot \frac{I_{MAX}}{U_{MAX}}$$
$$K_{iQ_{SC}} = \frac{K_{iQ} \cdot T_S}{2} \cdot \frac{I_{MAX}}{U_{MAX}}$$

Equation AMCLIB_CurrentLoopPIp_F16_Eq2

T_S is the sampling period [s]. In case the coefficients $K_{pD\ SC}$, $K_{iD\ SC}$, $K_{pQ\ SC}$ or $K_{piQ\ SC}$ exceed the fractional range [-1, 1), or they are extremely small (which could lead to loss of precision), their value must be bit-shifted by following coefficients

$$pPIpAWDs16PropGainShift = \lceil \log_2(|K_{pD_{SC}}|) \rceil$$
$$pPIpAWDs16IntegGainShift = \lceil \log_2(|K_{iD_{SC}}|) \rceil$$
$$pPIpAWQs16PropGainShift = \lceil \log_2(|K_{pQ_{SC}}|) \rceil$$
$$pPIpAWQs16IntegGainShift = \lceil \log_2(|K_{iQ_{SC}}|) \rceil$$

Equation AMCLIB_CurrentLoopPIp_F16_Eq3

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 15th power of two (corresponds to the Q1.15 encoding):

$$pPIpAWD.f16PropGain = \left(K_{pD_{SC}} \cdot 2^{-pPIpAWD.s16PropGainShift}\right) \cdot 2^{15}$$

$$pPIpAWD.f16IntegGain = \left(K_{iD_{SC}} \cdot 2^{-pPIpAWD.s16IntegGainShift}\right) \cdot 2^{15}$$

$$pPIpAWQ.f16PropGain = \left(K_{pQ_{SC}} \cdot 2^{-pPIpAWQ.s16PropGainShift}\right) \cdot 2^{15}$$

$$pPIpAWQ.f16IntegGain = \left(K_{iQ_{SC}} \cdot 2^{-pPIpAWQ.s16IntegGainShift}\right) \cdot 2^{15}$$

Equation AMCLIB_CurrentLoopPIp_F16_Eq4

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

2.4.4.5 Code Example

```
#include "AMCLIB_CurrentLoopPIp.h"

AMCLIB_CURRENT_LOOP_PIP_T_F16 CurrentLoop;
SWLIBS_2Syst_F16 IDQReq;       // required dq currents
SWLIBS_2Syst_F16 IDQFbck;      // feedback dq currents
SWLIBS_2Syst_F16 UDQReq;       // required dq voltages
tFrac16 f16UDcBus;             // DC bus voltage

// Initialize the parameters and pointers in CurrentLoop
CurrentLoop.pPIpAWD.f16PropGain       = FRAC16(0.1345);
CurrentLoop.pPIpAWD.f16IntegGain      = FRAC16(0.3498);
CurrentLoop.pPIpAWD.s16PropGainShift  = (tS16)1;
CurrentLoop.pPIpAWD.s16IntegGainShift = (tS16)2;
CurrentLoop.pPIpAWQ.f16PropGain       = FRAC16(0.6432);
CurrentLoop.pPIpAWQ.f16IntegGain      = FRAC16(0.2735);
CurrentLoop.pPIpAWQ.s16PropGainShift  = (tS16)1;
CurrentLoop.pPIpAWQ.s16IntegGainShift = (tS16)2;
CurrentLoop.pIDQReq                   = &IDQReq;
CurrentLoop.pIDQFbck                  = &IDQFbck;

// Alternative 1: API call with postfix
// (only one alternative shall be used).
AMCLIB_CurrentLoopPIp_F16(f16UDcBus, &UDQReq, &CurrentLoop);

// Alternative 2: API call with implementation parameter
// (only one alternative shall be used).
AMCLIB_CurrentLoopPIp(f16UDcBus, &UDQReq, &CurrentLoop, F16);

// Alternative 3: API call with global configuration of implementation
// (only one alternative shall be used). This alternative is available
// only if 16-bit fractional implementation is selected as default.
AMCLIB_CurrentLoopPIp(f16UDcBus, &UDQReq, &CurrentLoop);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.4.5 Function AMCLIB_CurrentLoopPIp_FLT

2.4.5.1 Declaration

```
void AMCLIB_CurrentLoopPIp_FLT(tFloat fltUDcBus, SWLIBS_2Syst_FLT
*const pUDQReq, AMCLIB_CURRENT_LOOP_PIP_T_FLT *pCtrl);
```

2.4.5.2 Arguments

Table 29. AMCLIB_CurrentLoopPIp_FLT arguments

Type	Name	Direction	Description
tFloat	fltUDcBus	**input**	DC bus voltage.
SWLIBS_2Syst_FLT *const	pUDQReq	**output**	Pointer to the structure with the required stator voltages in the two-phase rotational orthogonal system (d-q).
AMCLIB_CURRENT_LOOP_PIP_T_FLT *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_CurrentLoopPIp state.

2.4.5.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_CurrentLoop_FLT.

Figure 48. Functions and data structures in AMCLIB_CurrentLoop_FLT

The current loop contains two controllers (GFLIB_ControllerPIpAW_FLT). The proportional gain K_P and integral gain K_I of these controllers can be calculated as follows:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$K_{pD} = 2 \cdot \xi \cdot \omega_0 \cdot L_D - R_S$$
$$K_{iD} = \omega_0^2 \cdot L_D$$
$$K_{pQ} = 2 \cdot \xi \cdot \omega_0 \cdot L_Q - R_S$$
$$K_{iQ} = \omega_0^2 \cdot L_Q$$

Equation AMCLIB_CurrentLoop_FLT_Eq1

where ξ is the current loop attenuation, and ω_0 is the current loop natural frequency [rad/s], R_S is the phase resistance [ohm], L_D and L_Q are phase inductances [H] in the d and q axes, respectively.

The continuous-time controller parameters K_P and K_I are then discretized using the bilinear transform.

$$pPIrAWD.fltCC1sc = K_{pD} + \frac{K_{iD} T_S}{2}$$
$$pPIrAWD.fltCC2sc = -K_{pD} + \frac{K_{iD} T_S}{2}$$
$$pPIrAWQ.fltCC1sc = K_{pQ} + \frac{K_{iQ} T_S}{2}$$
$$pPIrAWQ.fltCC2sc = -K_{pQ} + \frac{K_{iQ} T_S}{2}$$

Equation AMCLIB_CurrentLoop_FLT_Eq2

T_S is the sampling period [s].

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.4.5.4 Code Example

```
#include "AMCLIB_CurrentLoopPIp.h"

AMCLIB_CURRENT_LOOP_PIP_T_FLT CurrentLoop;
SWLIBS_2Syst_FLT IDQReq;       // required dq currents
SWLIBS_2Syst_FLT IDQFbck;      // feedback dq currents
SWLIBS_2Syst_FLT UDQReq;       // required dq voltages
tFloat fltUDcBus;              // DC bus voltage

// Initialize the parameters and pointers in CurrentLoop
CurrentLoop.pPIpAWD.fltPropGain       = 0.1345f;
CurrentLoop.pPIpAWD.fltIntegGain      = 0.3498f;
CurrentLoop.pPIpAWQ.fltPropGain       = 0.6432f;
CurrentLoop.pPIpAWQ.fltIntegGain      = 0.2735f;
CurrentLoop.pIDQReq                   = &IDQReq;
CurrentLoop.pIDQFbck                  = &IDQFbck;

// Alternative 1: API call with postfix
// (only one alternative shall be used).
AMCLIB_CurrentLoopPIp_FLT(fltUDcBus, &UDQReq, &CurrentLoop);

// Alternative 2: API call with implementation parameter
// (only one alternative shall be used).
AMCLIB_CurrentLoopPIp(fltUDcBus, &UDQReq, &CurrentLoop, FLT);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
// Alternative 3: API call with global configuration of implementation
// (only one alternative shall be used). This alternative is available
// only if floating point implementation is selected as default.
AMCLIB_CurrentLoopPIp(fltUDcBus, &UDQReq, &CurrentLoop);
```

2.4.6 Function AMCLIB_CurrentLoopPIpInit

2.4.6.1 Description

This function clears the AMCLIB_CurrentLoopPIp state variables.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.4.6.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.4.6.3 Function AMCLIB_CurrentLoopPIpInit_F32

2.4.6.3.1 Declaration

```
void AMCLIB_CurrentLoopPIpInit_F32(AMCLIB_CURRENT_LOOP_PIP_T_F32
*const pCtrl);
```

2.4.6.3.2 Arguments

Table 30. AMCLIB_CurrentLoopPIpInit_F32 arguments

Type	Name	Direction	Description
AMCLIB_CURRENT_LOOP_PIP_T_F32 *const	pCtrl	**input, output**	Pointer to the structure with AMCLIB_CurrentLoopPIp state.

2.4.6.3.3 Code Example

```
#include "AMCLIB_CurrentLoopPIp.h"

AMCLIB_CURRENT_LOOP_PIP_T_F32 CurrentLoop;

// Alternative 1: API call with postfix
// (only one alternative shall be used).
AMCLIB_CurrentLoopPIpInit_F32(&CurrentLoop);

// Alternative 2: API call with implementation parameter
// (only one alternative shall be used).
AMCLIB_CurrentLoopPIpInit(&CurrentLoop, F32);

// Alternative 3: API call with global configuration of implementation
// (only one alternative shall be used). This alternative is available
// only if 32-bit fractional implementation is selected as default.
AMCLIB_CurrentLoopPIpInit(&CurrentLoop);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.4.6.4 Function AMCLIB_CurrentLoopPIpInit_F16

2.4.6.4.1 Declaration

```
void AMCLIB_CurrentLoopPIpInit_F16(AMCLIB_CURRENT_LOOP_PIP_T_F16
*const pCtrl);
```

2.4.6.4.2 Arguments

Table 31. AMCLIB_CurrentLoopPIpInit_F16 arguments

Type	Name	Direction	Description
AMCLIB_CURRENT_LOOP_PIP_T_F16 *const	pCtrl	**input, output**	Pointer to the structure with AMCLIB_CurrentLoopPIp state.

2.4.6.4.3 Code Example

```
#include "AMCLIB_CurrentLoopPIp.h"

AMCLIB_CURRENT_LOOP_PIP_T_F16 CurrentLoop;

// Alternative 1: API call with postfix
// (only one alternative shall be used).
AMCLIB_CurrentLoopPIpInit_F16(&CurrentLoop);

// Alternative 2: API call with implementation parameter
// (only one alternative shall be used).
AMCLIB_CurrentLoopPIpInit(&CurrentLoop, F16);

// Alternative 3: API call with global configuration of implementation
// (only one alternative shall be used). This alternative is available
// only if 16-bit fractional implementation is selected as default.
AMCLIB_CurrentLoopPIpInit(&CurrentLoop);
```

2.4.6.5 Function AMCLIB_CurrentLoopPIpInit_FLT

2.4.6.5.1 Declaration

```
void AMCLIB_CurrentLoopPIpInit_FLT(AMCLIB_CURRENT_LOOP_PIP_T_FLT
*const pCtrl);
```

2.4.6.5.2 Arguments

Table 32. AMCLIB_CurrentLoopPIpInit_FLT arguments

Type	Name	Direction	Description
AMCLIB_CURRENT_LOOP_PIP_T_FLT *const	pCtrl	**input, output**	Pointer to the structure with AMCLIB_CurrentLoopPIp state.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.4.6.5.3 Code Example

```
#include "AMCLIB_CurrentLoopPIp.h"

AMCLIB_CURRENT_LOOP_PIP_T_FLT CurrentLoop;

// Alternative 1: API call with postfix
// (only one alternative shall be used).
AMCLIB_CurrentLoopPIpInit_FLT(&CurrentLoop);

// Alternative 2: API call with implementation parameter
// (only one alternative shall be used).
AMCLIB_CurrentLoopPIpInit(&CurrentLoop, FLT);

// Alternative 3: API call with global configuration of implementation
// (only one alternative shall be used). This alternative is available
// only if floating point implementation is selected as default.
AMCLIB_CurrentLoopPIpInit(&CurrentLoop);
```

2.4.7 Function AMCLIB_CurrentLoopPIpSetState

2.4.7.1 Description

This function initializes the AMCLIB_CurrentLoopPIp state variables to achieve the required output values.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.4.7.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.4.7.3 Function AMCLIB_CurrentLoopPIpSetState_F32

2.4.7.3.1 Declaration

```
void AMCLIB_CurrentLoopPIpSetState_F32(tFrac32
f32ControllerPIpAWDOut, tFrac32 f32ControllerPIpAWQOut,
AMCLIB_CURRENT_LOOP_PIP_T_F32 *pCtrl);
```

2.4.7.3.2 Arguments

Table 33. AMCLIB_CurrentLoopPIpSetState_F32 arguments

Type	Name	Direction	Description
tFrac32	f32ControllerPIpAWDOut	**output**	Required output of the d-axis ControllerPIpAW.
tFrac32	f32ControllerPIpAWQOut	**output**	Required output of the q-axis ControllerPIpAW.
AMCLIB_CURRENT_LOOP_PIP_T_F32 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_CurrentLoopPIp state.

2.4.7.3.3 Code Example

```
#include "AMCLIB_CurrentLoopPIp.h"
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
AMCLIB_CURRENT_LOOP_PIP_T_F32 CurrentLoop;
tFrac32 f32ControllerPIpAWDOut;
tFrac32 f32ControllerPIpAWQOut;

f32ControllerPIrAWDOut = (tFrac32)123;
f32ControllerPIrAWQOut = (tFrac32)456;

// Alternative 1: API call with postfix
// (only one alternative shall be used).
AMCLIB_CurrentLoopPIpSetState_F32(f32ControllerPIpAWDOut, f32ControllerPIpAWQOut, &CurrentLoop);

// Alternative 2: API call with implementation parameter
// (only one alternative shall be used).
AMCLIB_CurrentLoopPIpSetState(f32ControllerPIpAWDOut, f32ControllerPIpAWQOut, &CurrentLoop, F32);

// Alternative 3: API call with global configuration of implementation
// (only one alternative shall be used). This alternative is available
// only if 32-bit fractional implementation is selected as default.
AMCLIB_CurrentLoopPIpSetState(f32ControllerPIpAWDOut, f32ControllerPIpAWQOut, &CurrentLoop);
}
```

2.4.7.4 Function AMCLIB_CurrentLoopPIpSetState_F16

2.4.7.4.1 Declaration

```
void AMCLIB_CurrentLoopPIpSetState_F16(tFrac16
f16ControllerPIpAWDOut, tFrac16 f16ControllerPIpAWQOut,
AMCLIB_CURRENT_LOOP_PIP_T_F16 *pCtrl);
```

2.4.7.4.2 Arguments

Table 34. AMCLIB_CurrentLoopPIpSetState_F16 arguments

Type	Name	Direction	Description
Commented parameter f32ControllerPIpAWDOut does not exist in function AMCLIB_CurrentLoopPIpSetState_F16.	f32ControllerPIpAWDOut	**output**	Required output of the d-axis ControllerPIpAW.
Commented parameter f32ControllerPIpAWQOut does not exist in function AMCLIB_CurrentLoopPIpSetState_F16.	f32ControllerPIpAWQOut	**output**	Required output of the q-axis ControllerPIpAW.
AMCLIB_CURRENT_LOOP_PIP_T_F16 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_CurrentLoopPIp state.

2.4.7.4.3 Code Example

```
#include "AMCLIB_CurrentLoopPIp.h"

AMCLIB_CURRENT_LOOP_PIP_T_F32 CurrentLoop;
tFrac32 f32ControllerPIpAWDOut;
tFrac32 f32ControllerPIpAWQOut;

f32ControllerPIrAWDOut = (tFrac32)123;
f32ControllerPIrAWQOut = (tFrac32)456;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
// Alternative 1: API call with postfix
// (only one alternative shall be used).
AMCLIB_CurrentLoopPIpSetState_F16(f32ControllerPIpAWDOut, f32ControllerPIpAWQOut, &CurrentLoop);

// Alternative 2: API call with implementation parameter
// (only one alternative shall be used).
AMCLIB_CurrentLoopPIpSetState(f32ControllerPIpAWDOut, f32ControllerPIpAWQOut, &CurrentLoop, F16);

// Alternative 3: API call with global configuration of implementation
// (only one alternative shall be used). This alternative is available
// only if 16-bit fractional implementation is selected as default.
AMCLIB_CurrentLoopPIpSetState(f32ControllerPIpAWDOut, f32ControllerPIpAWQOut, &CurrentLoop);
}
```

2.4.7.5 Function AMCLIB_CurrentLoopPIpSetState_FLT

2.4.7.5.1 Declaration

```
void AMCLIB_CurrentLoopPIpSetState_FLT(tFloat
fltControllerPIpAWDOut, tFloat fltControllerPIpAWQOut,
AMCLIB_CURRENT_LOOP_PIP_T_FLT *pCtrl);
```

2.4.7.5.2 Arguments

Table 35. AMCLIB_CurrentLoopPIpSetState_FLT arguments

Type	Name	Direction	Description
tFloat	fltControllerPIpAWDOut	**input**	Required output of the d-axis ControllerPIpAW.
tFloat	fltControllerPIpAWQOut	**input**	Required output of the q-axis ControllerPIpAW.
AMCLIB_CURRENT_LOOP_PIP_T_FLT *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_CurrentLoopPIp state.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.4.7.5.3 Code Example

```
#include "AMCLIB_CurrentLoopPIp.h"

AMCLIB_CURRENT_LOOP_PIP_T_FLT CurrentLoop;
tFloat fltControllerPIpAWDOut;
tFloat fltControllerPIpAWQOut;

fltControllerPIrAWDOut = 0.123f;
fltControllerPIrAWQOut = 0.456f;

// Alternative 1: API call with postfix
// (only one alternative shall be used).
AMCLIB_CurrentLoopPIpSetState_FLT(fltControllerPIpAWDOut, fltControllerPIpAWQOut, &CurrentLoop);

// Alternative 2: API call with implementation parameter
// (only one alternative shall be used).
AMCLIB_CurrentLoopPIpSetState(fltControllerPIpAWDOut, fltControllerPIpAWQOut, &CurrentLoop, FLT);

// Alternative 3: API call with global configuration of implementation
// (only one alternative shall be used). This alternative is available
// only if floating point implementation is selected as default.
AMCLIB_CurrentLoopPIpSetState(fltControllerPIpAWDOut, fltControllerPIpAWQOut, &CurrentLoop);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.5 Function AMCLIB_EstimBJ_FLT

This function performs offline estimation of the mechanical parameters of a permanent magnet synchronous motor.

2.5.1 Declaration

```
AMCLIB_ESTIMBJ_RET_T AMCLIB_EstimBJ_FLT(SWLIBS_2Syst_FLT *const
pUAlBeReq, const SWLIBS_2Syst_FLT *const pIAlBeFbck, tFloat
fltUDcBus, AMCLIB_ESTIMBJ_T_FLT *const pCtrl);
```

2.5.2 Arguments

Table 36. AMCLIB_EstimBJ_FLT arguments

Type	Name	Direction	Description
SWLIBS_2Syst_FLT *const	pUAlBeReq	**output**	Pointer to the structure containing calculated output voltages in the 2-phase stationary orthogonal system (α-β) [V].
const SWLIBS_2Syst_FLT *const	pIAlBeFbck	**input**	Pointer to the structure containing the measured phase currents in the 2-phase stationary orthogonal system (α-β) [A].
tFloat	fltUDcBus	**input**	Measured DC bus voltage [V].
AMCLIB_ESTIMBJ_T_FLT *const	pCtrl	**input, output**	Pointer to the structure with parameters and state variables.

2.5.3 Return

The returned value indicates the current stage of the algorithm.

2.5.4 Description

Sensorless control of a 3-phase permanent magnet synchronous motor (PMSM) requires knowledge of the motor's electrical and mechanical parameters. Some of these parameters are usually provided by the vendor (e.g. nominal current, number of pole pairs, nominal speed), others need to be obtained by measurement. The electrical parameters can be estimated with the function AMCLIB_EstimRL_FLT. This library function performs offline estimation of the PMSM's mechanical parameters in the target application's hardware setup using the existing three-phase power supply inverter and shunt resistors to obtain three-phase stator currents. In "basic" mode the function can estimate the following motor parameters:

- Torque constant, k_t [N·m/A]
- Moment of inertia, J [kg·m^2]

In "advanced" mode, when two measurement points are defined, the function can estimate in addition the following parameters:

- Static load torque component independent of speed, A [N·m]
- Viscous friction torque component proportional to speed, B [N·m·s/rad]

The relation of the estimated torque parameters to the motor electrical torque T can be expressed as

e

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$T_e = J\frac{d\omega_r}{dt} + A + B\omega_r + C\omega_r^2 + D\omega_r^3$$

Equation AMCLIB_EstimBJ_Eq1

where ω_r is the angular velocity and the constants A, B, C, and D represent the sum of all load torque components proportional to the respective powers of the angular velocity. Note that the components C and D are not estimated by this library function.

In order to be able to estimate the mechanical parameters, the motor must be brought up to speed and the feedback currents must be measured. The following diagram depicts the measurement setup which is needed to use this library function.

Figure 49. Measurement setup for the estimation of the mechanical parameters

The process of estimation of the mechanical parameters assumes that the motor can be driven by a field-oriented control (FOC) algorithm. Since the proper FOC setup requires prior knowledge of these parameters, the FOC controller is operating without all the necessary information. This may lead to unsuccessful measurement in case the motor parameters are far off the expected range. The function will indicate whether the estimation procedure was successful or not. It is recommended to start the measurement with the default "basic" mode (using the ESTIMBJ_INIT_DEFAULT_FLT macro), which requires minimal user interaction. Only if the measurement is unsuccessful, the user should consider configuring various internal parameters of the control algorithm manually (the "advanced" mode). Only the "basic" mode is intended to be used without the supervision of NXP support experts.

Function AMCLIB_EstimBJInit_FLT must be called before the first use of AMCLIB_EstimBJ_FLT to initialize the state variables and set the algorithm parameters. The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.5.5 Code Example

```
#include "amclib.h"

#define F_SAMPLING  10000
#define I_NOMINAL   6
#define W_NOMINAL   3000
#define Ld          0.00023
#define Lq          0.00025
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
#define Rs          0.28
#define DtVDrop     0.1657

SWLIBS_2Syst_FLT         fltUAlBeReq;
SWLIBS_2Syst_FLT         fltUDqReqLim;
SWLIBS_3Syst_FLT         fltUabcPwm;
tU32                     u32Sector;
SWLIBS_3Syst_FLT         fltIabcFbck;
SWLIBS_2Syst_FLT         fltIAlBeFbck;
tFloat                   fltUDcBus;
AMCLIB_ESTIMBJ_T_FLT     estimBJStruct;
AMCLIB_ESTIMBJINIT_RET_T retValInit;
AMCLIB_ESTIMBJ_RET_T     retVal;
GMCLIB_ELIMDCBUSRIP_T_FLT elimDcParStruct;

void main (void)
{
    // Initialize AMCLIB_EstimBJ
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    retValInit = AMCLIB_EstimBJInit_FLT(&estimBJStruct,
      ESTIMBJ_INIT_DEFAULT_FLT(F_SAMPLING, I_NOMINAL, W_NOMINAL, Ld, Lq, Rs, DtVDrop));

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    retValInit = AMCLIB_EstimBJInit(&estimBJStruct,
      ESTIMBJ_INIT_DEFAULT_FLT(F_SAMPLING, I_NOMINAL, W_NOMINAL, Ld, Lq, Rs, DtVDrop), FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    retValInit = AMCLIB_EstimBJInit(F_SAMPLING, &estimBJStruct,
      ESTIMBJ_INIT_DEFAULT(F_SAMPLING, I_NOMINAL, W_NOMINAL, Ld, Lq, Rs, DtVDrop));

    while(1);
}

// Periodical function or interrupt
void ISR(void)
{
    // Measure the 3-phase currents fltIabcFbck and convert them
    // into the 2-phase stationary orthogonal system alpha-beta
    GMCLIB_Clark_FLT(&fltIAlBeFbck, &fltIabcFbck)

    // Measure the DC bus voltage
    fltUDcBus = (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    retVal = AMCLIB_EstimBJ_FLT(&fltUAlBeReq, &fltIAlBeFbck, fltUDcBus, &estimBJStruct);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    retVal = AMCLIB_EstimBJ(&fltUAlBeReq, &fltIAlBeFbck, fltUDcBus, &estimBJStruct, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    retVal = AMCLIB_EstimBJ(&fltUAlBeReq, &fltIAlBeFbck, fltUDcBus, &estimBJStruct);

    if(RET_DONE == retVal)
    {
      // finish the measurement
    }

    // Convert the calculated voltages into PWM duty cycles
    elimDcParStruct.fltArgDcBusMsr = fltUDcBus;
    GMCLIB_ElimDcBusRip_FLT(&fltUDqReqLim, &fltUAlBeReq, &elimDcParStruct);
    u32Sector = GMCLIB_SvmStd_FLT(&fltUabcPwm, &fltUDqReqLim);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.5.6 Reentrancy

The function is reentrant for a different pCtrl.

2.5.7 Function AMCLIB_EstimBJInit_FLT

2.5.7.1 Declaration

```
AMCLIB_ESTIMBJINIT_RET_T
AMCLIB_EstimBJInit_FLT(AMCLIB_ESTIMBJ_T_FLT *const pCtrl,
AMCLIB_ESTIMBJ_INIT_T_FLT pParam);
```

2.5.7.2 Arguments

Table 37. AMCLIB_EstimBJInit_FLT arguments

Type	Name	Direction	Description
AMCLIB_ESTIMBJ_T_FLT *const	pCtrl	**input, output**	Pointer to the structure containing state variables and estimated parameters.
AMCLIB_ESTIMBJ_INIT_T_FLT	pParam	**input**	Structure of configuration parameters. In "basic" mode, it is intended to be filled with the predefined macro ESTIMBJ_INIT_DEFAULT_FLT.

2.5.7.3 Return

The returned value indicates whether the initialization was successful.

2.5.7.4 Implementation details

This function initializes all state variables and parameters of the PMSM mechanical parameters estimator. The initialization function must be called first before starting the estimation process. Refer to AMCLIB_EstimBJ_FLT for the description of available configuration options. The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.5.7.5 Code Example

```
#include "amclib.h"

#define F_SAMPLING  10000
#define I_NOMINAL   6
#define W_NOMINAL   3000
#define Ld          0.00023
#define Lq          0.00025
#define Rs          0.28
#define DtVDrop     0.1657

SWLIBS_2Syst_FLT          fltUAlBeReq;
SWLIBS_2Syst_FLT          fltUDqReqLim;
SWLIBS_3Syst_FLT          fltUabcPwm;
tU32                      u32Sector;
SWLIBS_3Syst_FLT          fltIabcFbck;
SWLIBS_2Syst_FLT          fltIAlBeFbck;
tFloat                    fltUDcBus;
AMCLIB_ESTIMBJ_T_FLT      estimBJStruct;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
AMCLIB_ESTIMBJINIT_RET_T retValInit;
AMCLIB_ESTIMBJ_RET_T     retVal;
GMCLIB_ELIMDCBUSRIP_T_FLT elimDcParStruct;

void main (void)
{
    // Initialize AMCLIB_EstimBJ
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    retValInit = AMCLIB_EstimBJInit_FLT(&estimBJStruct,
      ESTIMBJ_INIT_DEFAULT_FLT(F_SAMPLING, I_NOMINAL, W_NOMINAL, Ld, Lq, Rs, DtVDrop));

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    retValInit = AMCLIB_EstimBJInit(&estimBJStruct,
      ESTIMBJ_INIT_DEFAULT_FLT(F_SAMPLING, I_NOMINAL, W_NOMINAL, Ld, Lq, Rs, DtVDrop), FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    retValInit = AMCLIB_EstimBJInit(F_SAMPLING, &estimBJStruct,
      ESTIMBJ_INIT_DEFAULT(F_SAMPLING, I_NOMINAL, W_NOMINAL, Ld, Lq, Rs, DtVDrop));

    while(1);
}

// Periodical function or interrupt
void ISR(void)
{
    // Measure the 3-phase currents fltIabcFbck and convert them
    // into the 2-phase stationary orthogonal system alpha-beta
    GMCLIB_Clark_FLT(&fltIAlBeFbck, &fltIabcFbck)

    // Measure the DC bus voltage
    fltUDcBus = (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    retVal = AMCLIB_EstimBJ_FLT(&fltUAlBeReq, &fltIAlBeFbck, fltUDcBus, &estimBJStruct);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    retVal = AMCLIB_EstimBJ(&fltUAlBeReq, &fltIAlBeFbck, fltUDcBus, &estimBJStruct, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    retVal = AMCLIB_EstimBJ(&fltUAlBeReq, &fltIAlBeFbck, fltUDcBus, &estimBJStruct);

    if(RET_DONE == retVal)
    {
      // finish the measurement
    }

    // Convert the calculated voltages into PWM duty cycles
    elimDcParStruct.fltArgDcBusMsr = fltUDcBus;
    GMCLIB_ElimDcBusRip_FLT(&fltUDqReqLim, &fltUAlBeReq, &elimDcParStruct);
    u32Sector = GMCLIB_SvmStd_FLT(&fltUabcPwm, &fltUDqReqLim);
}
```

2.5.7.6 Reentrancy

The function is reentrant for a different pCtrl.

2.6 Function AMCLIB_EstimRL

This function performs offline estimation of the electrical parameters of a permanent magnet synchronous motor.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.6.1 Description

Sensorless control of a 3-phase permanent magnet synchronous motor (PMSM) requires knowledge of the motor's electrical and mechanical parameters. Some of these parameters are usually provided by the vendor (e.g. nominal current, number of pole pairs, nominal speed), others need to be obtained by measurement. Refer to AMCLIB_EstimBJ for the estimation of mechanical parameters. This library function performs offline estimation of the PMSM's electrical parameters in the target application's hardware setup using the existing three-phase power supply inverter and shunt resistors to obtain three-phase stator currents. The function can estimate the following motor parameters:

- R_s, i.e. the resistance of one stator phase [Ω],
- L_d and L_q, i.e. the d-axis and q-axis inductances [H] in the two-phase (d-q) rotational orthogonal coordinate system,
- U_{dt}, i.e. the voltage drop due to the power supply inverter dead time (the interval when both transistors in the given phase are switched off) [V].

To use this function, the rotor must be either allowed to freely rotate or it must be mechanically fixed in alignment with the first phase. The function will inject magnetization DC current that will force the rotor in the correct position (in the free rotating case). The inductances are measured by injecting a low-amplitude high-frequency AC current. Depending on the mode of operation, the funciton will inject DC current either in the d-axis (i.e. I_{DCd}) or the q-axis (i.e. I_{DCq}) or both.

The function can be configured for either automatic or manual measurement. The mode of operation is determined from the preset applied to the input struct `pParam`. The presets are available as predefined macros (separate for the AMCLIB_EstimRLInit function and the AMCLIB_EstimRL function):

- ESTIMRL_INIT_DEFAULT_<impl>, ESTIMRL_RUN_DEFAULT_<impl> - this preset will measure d-axis inductance for the case I_{DCd} = 0 and the q-axis inductance for the case I_{DCd} = 0. This preset keeps I_{DCq} = 0. Both inductances and the R_s and U_{dt} values are provided in the structure pointed to by `pCtrl`. The rotor must be allowed to freely rotate.
- ESTIMRL_INIT_CFG1_<impl>, ESTIMRL_RUN_CFG1_<impl> - this preset will perform automatic measurement of the inductances for a user-specified number of different DC current levels, using positive values of the DC current. The estimated inductances are provided in a table pointed to by `pLdqTable`. The function will also output L_d for I_{DCd} = 0, L_q for I_{DCd} = `fltIDcLq`, R_s, and U_{dt} in the structure pointed to by `pCtrl` This preset keeps I_{DCq} = 0. The rotor must be allowed to freely rotate.
- ESTIMRL_INIT_CFG2_<impl>, ESTIMRL_RUN_CFG2_<impl> - this preset will perform automatic measurement of the inductances for a user-specified number of different DC current levels using both positive and negative values of the DC current. The q-axis inductance measurements use DC current injection in the q-axis. The estimated inductances are provided in a table pointed to by `pLdqTable`. The function will also output L_d for I_{DCd} = `fltIDcLd`, L_q for I_{DCd} = `fltIDcLq`, R_s, and U_{dt} in the structure pointed to by `pCtrl`. The rotor must be mechanically fixed in alignment with the first phase.
- ESTIMRL_INIT_CFG3_<impl>, ESTIMRL_RUN_CFG3_<impl> - this preset serves for manual measurement. All parameters can be changed during runtime. The function will not calculate R_s and U_{dt} and it will not de-noise the measured inductances. The rotor must be mechanically fixed in alignment with the first phase.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The following table summarizes the current injection patterns used with the available presets.

Table 38. AMCLIB_EstimRL presets

Preset	I_{DC} stepping	Rotor fixed	L_d measurement	L_q measurement
DEFAULT	Auto	Free	$I_d = I_{DCd} + I_{AC}$, $I_q = 0$	$I_d = I_{DCd}$, $I_q = I_{AC}$
CFG1	Auto	Free	$I_d = I_{DCd} + I_{AC}$, $I_q = 0$	$I_d = I_{DCd}$, $I_q = I_{AC}$
CFG2	Auto	Fixed	$I_d = I_{DCd} + I_{AC}$, $I_q = 0$	$I_d = 0$, $I_q = I_{DCq} + I_{AC}$
CFG3	Manual	Fixed	$I_d = I_{DCd} + I_{AC}$, $I_q = I_{DCq}$	$I_d = I_{DCd}$, $I_q = I_{DCq} + I_{AC}$

The motor inductances cannot be measured accurately when I_{DC} is close to zero due to the dead time inserted in the switching pattern of the power stage transistors. If the function is configured for automatic measurement, it will calculate the correct inductances by interpolating two other measurements taken with nonzero I_{DC}. Note that the required I_{DC} will be automatically reduced if the available DC bus voltage is insufficient to achieve such current.

The estimation process can take several seconds, depending on the amount of noise in the measured currents and the number of required measurement points. The function AMCLIB_EstimRL must be called periodically with the specified sampling frequency until it returns RET_DONE (or RET_ERROR). If the function returns RET_ERROR, the measurement cannot continue and the function must be reinitialized before another measurement can be attempted.

The input feedback currents and the output phase voltages are provided in the two-phase orthogonal coordinate system (alpha/beta). Use the GMCLIB_Clark function to transform the measured three-phase currents into the two-phase system required by AMCLIB_EstimRL. The output voltages calculated by AMCLIB_EstimRL shall be processed by the GMCLIB_ElimDcBusRip function, followed by the GMCLIB_SvmStd function. It is possible to use other PWM generating functions as well (e.g. GMCLIB_PwmIct, GMCLIB_SvmSci, etc.), however, AMCLIB_EstimRL will correctly detect the DC bus voltage limitation only if GMCLIB_SvmStd is used.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.6.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.6.3 Function AMCLIB_EstimRL_FLT

2.6.3.1 Declaration

```
AMCLIB_ESTIMRL_RET_T AMCLIB_EstimRL_FLT(SWLIBS_2Syst_FLT
*const pUAlBeReq, const SWLIBS_2Syst_FLT *const pIAlBeFbck,
tFloat fltUDcBus, AMCLIB_ESTIMRL_T_FLT *const pCtrl,
AMCLIB_ESTIMRL_RUN_T_FLT pParam);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.6.3.2 Arguments

Table 39. AMCLIB_EstimRL_FLT arguments

Type	Name	Direction	Description
SWLIBS_2Syst_FLT *const	pUAlBeReq	**output**	Pointer to the structure containing calculated output voltages in the 2-phase stationary orthogonal system (α-β) [V].
const SWLIBS_2Syst_FLT *const	pIAlBeFbck	**input**	Pointer to the structure containing the measured phase currents in the 2-phase stationary orthogonal system (α-β) [A].
tFloat	fltUDcBus	**input**	Measured DC bus voltage [V].
AMCLIB_ESTIMRL_T_FLT *const	pCtrl	**input, output**	Pointer to the structure with parameters and state variables.
AMCLIB_ESTIMRL_RUN_T_FLT	pParam	**input**	Structure of configuration parameters. It is intended to be filled with one of the predefined macros: ESTIMRL_RUN_DEFAULT_FLT, ESTIMRL_RUN_CFG1_FLT, ESTIMRL_RUN_CFG2_FLT, or ESTIMRL_RUN_CFG3_FLT.

2.6.3.3 Return

The returned value indicates the current stage of the algorithm.

2.6.3.4 Implementation details

Function AMCLIB_EstimRLInit_FLT must be called before the first use of AMCLIB_EstimRL_FLT to initialize the state variables and set the parameters.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.6.3.5 Code Example

```
#include "amclib.h"

#define F_SAMPLING  10000
#define I_NOMINAL  10.0f

SWLIBS_2Syst_FLT          fltUAlBeReq;
SWLIBS_2Syst_FLT          fltUDqReqLim;
SWLIBS_3Syst_FLT          fltUabcPwm;
tU32                      u32Sector;
SWLIBS_3Syst_FLT          fltIabcFbck;
SWLIBS_2Syst_FLT          fltIAlBeFbck;
tFloat                    fltUDcBus;
AMCLIB_ESTIMRL_T_FLT      estimRLStruct;
AMCLIB_ESTIMRLINIT_RET_T  retValInit;
AMCLIB_ESTIMRL_RET_T      retVal;
GMCLIB_ELIMDCBUSRIP_T_FLT elimDcParStruct;

void main (void)
{
    // Initialize AMCLIB_EstimRL
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    retValInit = AMCLIB_EstimRLInit_FLT(F_SAMPLING, &estimRLStruct,
      ESTIMRL_INIT_DEFAULT_FLT(I_NOMINAL));

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    retValInit = AMCLIB_EstimRLInit(F_SAMPLING, &estimRLStruct,
      ESTIMRL_INIT_DEFAULT_FLT(I_NOMINAL), FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    retValInit = AMCLIB_EstimRLInit(F_SAMPLING, &estimRLStruct,
      ESTIMRL_INIT_DEFAULT_FLT(I_NOMINAL));

    while(1);
}

// Periodical function or interrupt
void ISR(void)
{
    // Measure the 3-phase currents fltIabcFbck and convert them
    // into the 2-phase stationary orthogonal system alpha-beta
    GMCLIB_Clark_FLT(&fltIAlBeFbck, &fltIabcFbck)

    // Measure the DC bus voltage
    fltUDcBus = (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    retVal = AMCLIB_EstimRL_FLT(&fltUAlBeReq, &fltIAlBeFbck, fltUDcBus,
      &estimRLStruct, ESTIMRL_RUN_DEFAULT_FLT);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    retVal = AMCLIB_EstimRL(&fltUAlBeReq, &fltIAlBeFbck, fltUDcBus,
      &estimRLStruct, ESTIMRL_RUN_DEFAULT_FLT, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    retVal = AMCLIB_EstimRL(&fltUAlBeReq, &fltIAlBeFbck, fltUDcBus,
      &estimRLStruct, ESTIMRL_RUN_DEFAULT_FLT);

    if(RET_DONE == retVal)
    {
      // finish the measurement
    }

    // Convert the calculated voltages into PWM duty cycles
    // (assuming the rotor is aligned with the first phase)
    elimDcParStruct.fltArgDcBusMsr = fltUDcBus;
    GMCLIB_ElimDcBusRip_FLT(&fltUDqReqLim, &fltUAlBeReq, &elimDcParStruct);
    u32Sector = GMCLIB_SvmStd_FLT(&fltUabcPwm, &fltUDqReqLim);
}
```

2.6.4 Function AMCLIB_EstimRLInit

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.6.4.1 Description

This function initializes all state variables and parameters of the PMSM electrical parameters estimator. The initialization function must be called first before starting the estimation process. Refer to AMCLIB_EstimRL for the description of available configuration options.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.6.4.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.6.4.3 Function AMCLIB_EstimRLInit_FLT

2.6.4.3.1 Declaration

```
AMCLIB_ESTIMRLINIT_RET_T AMCLIB_EstimRLInit_FLT(tU32
u32SamplingFreq, AMCLIB_ESTIMRL_T_FLT *const pCtrl,
AMCLIB_ESTIMRL_INIT_T_FLT pParam);
```

2.6.4.3.2 Arguments

Table 40. AMCLIB_EstimRLInit_FLT arguments

Type	Name	Direction	Description
tU32	u32SamplingFreq	**input**	ADC/DAC Sampling frequency, control loop frequency [Hz]. The recommended frequency is 10000 Hz. The sampling frequency must be at least four times higher than the injected AC current frequency (the default AC current frequency is 500 Hz).
AMCLIB_ESTIMRL_T_FLT *const	pCtrl	**input, output**	Pointer to the structure with parameters and state variables.
AMCLIB_ESTIMRL_INIT_T_FLT	pParam	**input**	Structure of configuration parameters. It is intended to be filled with one of the predefined macros: ESTIMRL_INIT_DEFAULT_FLT, ESTIMRL_INIT_CFG1_FLT, ESTIMRL_INIT_CFG2_FLT, or ESTIMRL_INIT_CFG3_FLT.

2.6.4.3.3 Return

The returned value indicates whether the initialization was successful.

2.6.4.3.4 Implementation details

Function AMCLIB_EstimRLInit_FLT must be called before the first use of AMCLIB_EstimRL_FLT to initialize the state variables and set the parameters.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.6.4.3.5 Code Example

```
#include "amclib.h"
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
#define F_SAMPLING  10000
#define I_NOMINAL  10.0f

SWLIBS_2Syst_FLT          fltUAlBeReq;
SWLIBS_2Syst_FLT          fltUDqReqLim;
SWLIBS_3Syst_FLT          fltUabcPwm;
tU32                      u32Sector;
SWLIBS_3Syst_FLT          fltIabcFbck;
SWLIBS_2Syst_FLT          fltIAlBeFbck;
tFloat                    fltUDcBus;
AMCLIB_ESTIMRL_T_FLT      estimRLStruct;
AMCLIB_ESTIMRLINIT_RET_T  retValInit;
AMCLIB_ESTIMRL_RET_T      retVal;
GMCLIB_ELIMDCBUSRIP_T_FLT elimDcParStruct;

void main (void)
{
    // Initialize AMCLIB_EstimRL
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    retValInit = AMCLIB_EstimRLInit_FLT(F_SAMPLING, &estimRLStruct,
      ESTIMRL_INIT_DEFAULT_FLT(I_NOMINAL));

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    retValInit = AMCLIB_EstimRLInit(F_SAMPLING, &estimRLStruct,
      ESTIMRL_INIT_DEFAULT_FLT(I_NOMINAL), FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    retValInit = AMCLIB_EstimRLInit(F_SAMPLING, &estimRLStruct,
      ESTIMRL_INIT_DEFAULT_FLT(I_NOMINAL));

    while(1);
}

// Periodical function or interrupt
void ISR(void)
{
    // Measure the 3-phase currents fltIabcFbck and convert them
    // into the 2-phase stationary orthogonal system alpha-beta
    GMCLIB_Clark_FLT(&fltIAlBeFbck, &fltIabcFbck)

    // Measure the DC bus voltage
    fltUDcBus = (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    retVal = AMCLIB_EstimRL_FLT(&fltUAlBeReq, &fltIAlBeFbck, fltUDcBus,
      &estimRLStruct, ESTIMRL_RUN_DEFAULT_FLT);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    retVal = AMCLIB_EstimRL(&fltUAlBeReq, &fltIAlBeFbck, fltUDcBus,
      &estimRLStruct, ESTIMRL_RUN_DEFAULT_FLT, FLT);

    // Alternative 3: API call with global configuration of implementation
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    retVal = AMCLIB_EstimRL(&fltUAlBeReq, &fltIAlBeFbck, fltUDcBus,
      &estimRLStruct, ESTIMRL_RUN_DEFAULT_FLT);

    if(RET_DONE == retVal)
    {
      // finish the measurement
    }

    // Convert the calculated voltages into PWM duty cycles
    // (assuming the rotor is aligned with the first phase)
    elimDcParStruct.fltArgDcBusMsr = fltUDcBus;
    GMCLIB_ElimDcBusRip_FLT(&fltUDqReqLim, &fltUAlBeReq, &elimDcParStruct);
    u32Sector = GMCLIB_SvmStd_FLT(&fltUabcPwm, &fltUDqReqLim);
}
```

2.7 Function AMCLIB_FluxEstimAB

This function calculates the PMSM rotor position by estimating the rotor flux in the α/β reference frame.

2.7.1 Description

The Rotor Flux Estimator calculates the Permanent Magnet Synchronous Motor (PMSM) rotor position by estimating the rotor flux in the α/β reference frame. The rotor position is needed for sensorless motor control applications with Field-Oriented Control (FOC).

Figure 50. Principal Field Oriented Control scheme with AMCLIB_FluxEstimAB

The estimator fits the input voltages and currents to a mathematical model of the motor according to the following equation:

$$\Psi_\alpha = \frac{1}{\omega_{el}}\left(u_\beta - R_S i_\beta\right) - L_S i_\alpha$$

$$\Psi_\beta = \frac{1}{\omega_{el}}\left(-u_\alpha + R_S i_\alpha\right) - L_S i_\beta$$

Equation AMCLIB_FluxEstimAB_Eq1

where

© NXP B.V. 2023. All rights reserved.

- i_α and u_β are actual values of current in α/β reference frame [A],
- u_α and u_β are actual values of voltage in α/β reference frame [V],
- R_s is the stator resistance [Ω],
- L_S is the stator inductance [H],
- ω_{el} is the electrical angular velocity of the rotor [rad/s],

Stator inductance L_S can be calculated by following equation:

$$L_S = \frac{L_d + L_q}{2}$$

Equation AMCLIB_FluxEstimAB_Eq2

where L_d and L_q are the d-axis and q-axis inductances.

For more robust sensorless application, AMCLIB_BemfObsrvDQ function can be used instead of AMCLIB_FluxEstimAB. In addition you can tune the dynamics of the closed loop system with AMCLIB_BemfObsrvDQ but this algorithm requires more computing resources.

Before using the estimator with a particular motor, the user needs to provide a set of motor parameters through the `pParam` input pointer. A method for measuring the motor parameters is described in PMSM Electrical Parameters Measurement (document AN4680) or the library function AMCLIB_EstimRL can be used.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.7.2 Re-entrancy

The function is re-entrant for a different set of input pointers.

2.7.3 Function AMCLIB_FluxEstimAB_F32

2.7.3.1 Declaration

```
tFrac32 AMCLIB_FluxEstimAB_F32(tFrac32 f32Velocity, const
SWLIBS_2Syst_F32 *const pIAB, const SWLIBS_2Syst_F32 *const pUAB,
const AMCLIB_FLUXESTIMAB_T_F32 *const pParam);
```

2.7.3.2 Arguments

Table 41. AMCLIB_FluxEstimAB_F32 arguments

Type	Name	Direction	Description
tFrac32	f32Velocity	**input**	Electrical angular velocity.
const SWLIBS_2Syst_F32 *const	pIAB	**input**	Pointer to the structure with Alpha/Beta current components.
const SWLIBS_2Syst_F32 *const	pUAB	**input**	Pointer to the structure with Alpha/Beta voltage components.
const AMCLIB_FLUXESTIMAB_T_F32 *const	pParam	**input**	Pointer to the structure containing motor parameters.

2.7.3.3 Return

Estimated rotor electrical phase.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.7.3.4 Implementation details

Prior to calculating the flux estimator coefficients, it is necessary to set the scaling constants. All inputs and outputs of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation. There are two different scaling systems involved, one for the FOC part of the motor control algorithm, and another for the flux estimator. Scaling constants must be positive values equal to or greater than the expected maxima of the corresponding physical quantities. The following scaling constants are applied to the flux estimator coefficients:

Table 42. Scaling constants

Scaling constant	Symbol	Calculation
Maximum stator phase voltage [V]	U_{MAX}	$U_{MAX}=U_{DC_Bus_Max}$
Maximum phase current [A]	I_{MAX}	Maximum current of the inverter or nominal current of the motor (whichever is lower).
Maximum speed [rad/s]	Ω_{MAX}	Maximum application required speed, at least the motor electrical rated speed.

Parameters R_S and L_S can be calculated as follows:

$$Rs_{SC} = Rs\frac{I_{MAX}}{U_{MAX}}$$
$$Ls_{SC} = Ls\frac{I_{MAX}\cdot\Omega_{MAX}}{U_{MAX}}$$

Equation AMCLIB_FluxEstimAB_F32_Eq1

In case the coefficients exceed the fractional range, their value must be right-shifted by *NShift*. The binary shift amount corresponds to

$$u16NShift = \max\left(0, \left\lceil \log_2\left(\max\left(Rs_{SC}, Ls_{SC}\right)\right)\right\rceil\right)$$

Equation AMCLIB_FluxEstimAB_F32_Eq2

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 31st power of two (corresponds to the Q1.31 encoding):

$$pParam.f32KRs = \left(Rs_{SC}\cdot 2^{-u16NShift}\right)\cdot 2^{31}$$
$$pParam.f32KLs = \left(Ls_{SC}\cdot 2^{-u16NShift}\right)\cdot 2^{31}$$

Equation AMCLIB_FluxEstimAB_F32_Eq3

2.7.3.5 Code Example

```
#include "AMCLIB_FluxEstimAB.h"

tFrac32 f32Velocity;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
SWLIBS_2Syst_F32 pIAB;
SWLIBS_2Syst_F32 pUAB;
AMCLIB_FLUXESTIMAB_T_F32 pParam;
tFrac32 f32Phase;

// Alternative 1: API call with postfix
// (only one alternative shall be used).
f32Phase = AMCLIB_FluxEstimAB_F32(f32Velocity, pIAB, pUAB, pParam);

// Alternative 2: API call with implementation parameter
// (only one alternative shall be used).
f32Phase = AMCLIB_FluxEstimAB(f32Velocity, pIAB, pUAB, pParam, F32);

// Alternative 3: API call with global configuration of implementation
// (only one alternative shall be used). This alternative is available
// only if 32-bit fractional implementation is selected as default.
f32Phase = AMCLIB_FluxEstimAB(f32Velocity, pIAB, pUAB, pParam);
```

2.7.4 Function AMCLIB_FluxEstimAB_F16

2.7.4.1 Declaration

```
tFrac16 AMCLIB_FluxEstimAB_F16(tFrac16 f16Velocity, const
SWLIBS_2Syst_F16 *const pIAB, const SWLIBS_2Syst_F16 *const pUAB,
const AMCLIB_FLUXESTIMAB_T_F16 *const pParam);
```

2.7.4.2 Arguments

Table 43. AMCLIB_FluxEstimAB_F16 arguments

Type	Name	Direction	Description
tFrac16	f16Velocity	**input**	Electrical angular velocity.
const SWLIBS_2Syst_F16 *const	pIAB	**input**	Pointer to the structure with Alpha/Beta current components.
const SWLIBS_2Syst_F16 *const	pUAB	**input**	Pointer to the structure with Alpha/Beta voltage components.
const AMCLIB_FLUXESTIMAB_T_F16 *const	pParam	**input**	Pointer to the structure containing motor parameters.

2.7.4.3 Return

Estimated rotor electrical phase.

2.7.4.4 Implementation details

Prior to calculating the flux estimator coefficients, it is necessary to set the scaling constants. All inputs and outputs of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation. There are two different scaling systems involved, one for the FOC part of the motor control algorithm, and another for the flux estimator. Scaling constants must be positive values equal to or greater than the expected maxima of the corresponding physical quantities. The following scaling constants are applied to the flux estimator coefficients:

Table 44. Scaling constants

Scaling constant	Symbol	Calculation

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 44. Scaling constants...*continued*

Maximum stator phase voltage [V]	U_{MAX}	$U_{MAX}=U_{DC_Bus_Max}$
Maximum phase current [A]	I_{MAX}	Maximum current of the inverter or nominal current of the motor (whichever is lower).
Maximum speed [rad/s]	Ω_{MAX}	Maximum application required speed, at least the motor electrical rated speed.

Parameters R_S and L_S can be calculated as follows:

$$Rs_{SC} = Rs\frac{I_{MAX}}{U_{MAX}}$$
$$Ls_{SC} = Ls\frac{I_{MAX} \cdot \Omega_{MAX}}{U_{MAX}}$$

Equation AMCLIB_FluxEstimAB_F16_Eq1

In case the coefficients exceed the fractional range, their value must be right-shifted by *NShift*. The binary shift amount corresponds to

$$u16NShift = \max\left(0, \left\lceil \log_2\left(\max\left(Rs_{SC}, Ls_{SC}\right)\right)\right\rceil\right)$$

Equation AMCLIB_FluxEstimAB_F16_Eq2

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 15st power of two (corresponds to the Q1.15 encoding):

$$pParam.f16KRs = \left(Rs_{SC} \cdot 2^{-u16NShift}\right) \cdot 2^{15}$$
$$pParam.f16KLs = \left(Ls_{SC} \cdot 2^{-u16NShift}\right) \cdot 2^{15}$$

Equation AMCLIB_FluxEstimAB_F16_Eq3

2.7.4.5 Code Example

```
#include "AMCLIB_FluxEstimAB.h"

tFrac16 f16Velocity;
SWLIBS_2Syst_F16 pIAB;
SWLIBS_2Syst_F16 pUAB;
AMCLIB_FLUXESTIMAB_T_F16 pParam;
tFrac16 f16Phase;

// Alternative 1: API call with postfix
// (only one alternative shall be used).
f16Phase = AMCLIB_FluxEstimAB_F16(f16Velocity, pIAB, pUAB, pParam);

// Alternative 2: API call with implementation parameter
// (only one alternative shall be used).
f16Phase = AMCLIB_FluxEstimAB(f16Velocity, pIAB, pUAB, pParam, F16);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
// Alternative 3: API call with global configuration of implementation
// (only one alternative shall be used). This alternative is available
// only if 16-bit fractional implementation is selected as default.
f16Phase = AMCLIB_FluxEstimAB(f16Velocity, pIAB, pUAB, pParam);
```

2.7.5 Function AMCLIB_FluxEstimAB_FLT

2.7.5.1 Declaration

```
tFloat AMCLIB_FluxEstimAB_FLT(tFloat fltVelocity, const
SWLIBS_2Syst_FLT *const pIAB, const SWLIBS_2Syst_FLT *const pUAB,
const AMCLIB_FLUXESTIMAB_T_FLT *const pParam);
```

2.7.5.2 Arguments

Table 45. AMCLIB_FluxEstimAB_FLT arguments

Type	Name	Direction	Description
tFloat	fltVelocity	**input**	Electrical angular velocity.
const SWLIBS_2Syst_FLT *const	pIAB	**input**	Pointer to the structure with Alpha/Beta current components.
const SWLIBS_2Syst_FLT *const	pUAB	**input**	Pointer to the structure with Alpha/Beta voltage components.
const AMCLIB_FLUXESTIMAB_T_FLT *const	pParam	**input**	Pointer to the structure containing motor parameters.

2.7.5.3 Return

Estimated rotor electrical phase.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.7.5.4 Code Example

```
#include "AMCLIB_FluxEstimAB.h"

tFloat fltVelocity;
SWLIBS_2Syst_FLT pIAB;
SWLIBS_2Syst_FLT pUAB;
AMCLIB_FLUXESTIMAB_T_FLT pParam;
tFloat fltPhase;

// Alternative 1: API call with postfix
// (only one alternative shall be used).
fltPhase = AMCLIB_FluxEstimAB_FLT(fltVelocity, pIAB, pUAB, pParam);

// Alternative 2: API call with implementation parameter
// (only one alternative shall be used).
fltPhase = AMCLIB_FluxEstimAB(fltVelocity, pIAB, pUAB, pParam, FLT);

// Alternative 3: API call with global configuration of implementation
// (only one alternative shall be used). This alternative is available
// only if floating point implementation is selected as default.
fltPhase = AMCLIB_FluxEstimAB(fltVelocity, pIAB, pUAB, pParam);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.8 Function AMCLIB_FW

This library function implements the field-weakening algorithm for permanent magnet synchronous motors.

2.8.1 Description

Field weakening represents an advanced control approach to run the electric motor beyond a base speed. The back electromotive force (EMF) is proportional to the rotor speed and counteracts the motor supply voltage. If a given speed is to be reached, the terminal voltage must be increased to match the increased stator back-EMF. A sufficient voltage is available from the inverter in the operation up to the base speed. Beyond the base speed, motor voltages u_d and u_q are limited and cannot be increased because of the ceiling voltage of a given inverter. As the difference between the induced back-EMF and the supply voltage decreases, the phase current flow is limited, hence the currents i_d and i_q cannot be controlled sufficiently. Further increase of speed would eventually result in back-EMF voltage equal to the limited stator voltage, which means a complete loss of current control. The only way to retain the current control even beyond the base speed is to lower the generated back-EMF by weakening the flux that links the stator winding.

Base speed splits the whole speed motor operation into two regions: constant torque and constant power, see Figure 51 .

Figure 51. Constant torque/power operational regions

Operation in constant torque region means that maximal torque can be constantly developed while the output power increases with the rotor speed. The phase voltage increases linearly with the speed and the current is controlled to its reference. The operation in constant power region is characterized by a rapid decrease in developed torque while the output power remains constant. The phase voltage is at its limit while the phase current and the stator flux decrease proportionally with the rotor speed, see Figure 52 .

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 52. Constant flux/voltage operational regions

Direct field weakening is possible only in drives with separate excitation. Nevertheless, the same effect can be achieved in drives with permanent magnets using an appropriate stator current control technique as outlined in the following Figure 53 .

Figure 53. Steady-state phasor diagram of PMSM, a) operation up to base speed, b) operation above base speed

The left diagram in Figure 53 depicts the normal operation when the stator current space phasor I_S is aligned with the q-axis. Consequently, the entire phase current is utilized for torque production, i.e. $I_Q = I_S$ and $I_D = 0$. The rotor speed is ω_e and the terminal stator voltage is V_{S1}. R_S and jX_S represent the stator resistance and reactance, respectively, j is the imaginary unit.

The right diagram in Figure 53 corresponds to the field weakening operation. The displacement of the current space phasor I_S yields to the production of the nonzero current component I_D along the negative d-axis of the rotor reference frame. The motor spins at the same speed ω_e; however, the terminal voltage V_{S2} is now lower than the previous V_{S1}. This is achieved thanks to the negative flux component $L_D I_D$ which counteracts the flux of the permanent magnets.

AMCLIB_FW implements the field weakening controller in the outer control loop highlighted in Figure 54 . AMCLIB_FW is intended to be used in combination with AMCLIB_SpeedLoop. The code can be simplified further by utilizing AMCLIB_FWSpeedLoop which combines the speed controller with the field weakening controller in one library function. AMCLIB_FW does not allow debugging of all internal variables. Use AMCLIB_FWDebug for debugging purposes and replace it with AMCLIB_FW once the debugging is finished.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 54. Principal Field Oriented Control scheme with AMCLIB_FW

Field weakening technique employed in AMCLIB_FW extends the speed range of PMSM beyond the base speed value by reducing the linkage magnetic flux. The algorithm implemented in the library brings the following key advantages:

- Fully utilizes the drive capabilities (speed range, load torque).
- Reduces the total linkage flux only when necessary.
- Achieves smooth transition between the normal and field weakening operating speed range, regains full control when recovering from voltage saturation.
- The algorithm is very robust - as a result, the PMSM behaves as a separately excited wound field synchronous motor drive.
- Allows maximum torque optimal control.

This algorithm is protected by US Patent No. US 2011/0050152 A1.

CAUTION:

1. *A motor operated at speeds over the base speed region generates higher back-EMF voltage than the supply stator voltage. By applying the field weakening technique, the back-EMF is actively kept under control, i.e. lower than the stator voltage. It is dangerous to break the control loop during the field weakening operation. A loss of control (e.g. due to a fault state or turning the application off while running) may result in damage of the electronics and hardware due to the excessive back-EMF which would no longer be suppressed by the control algorithm.*
2. *The field is weakened by applying a negative current d-component. Too high negative current can cause magnet damage by its demagnetization. Make sure to set a correct lower limit of the field weakening PI controller to prevent damage to the motor.*

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.8.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.8.3 Function AMCLIB_FW_F32

2.8.3.1 Declaration

```
void AMCLIB_FW_F32(tFrac32 f32IDQReqAmp, tFrac32 f32VelocityFbck,
SWLIBS_2Syst_F32 *const pIDQReq, AMCLIB_FW_T_F32 *pCtrl);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.8.3.2 Arguments

Table 46. AMCLIB_FW_F32 arguments

Type	Name	Direction	Description
tFrac32	f32IDQReqAmp	**input**	Required amplitude of the currents Id and Iq in the two-phase rotational orthogonal system (d-q).
tFrac32	f32VelocityFbck	**input**	Actual electrical angular velocity from the feedback.
SWLIBS_2Syst_F32 *const	pIDQReq	**input, output**	Pointer to the structure with the required stator currents in the two-phase rotational orthogonal system (d-q).
AMCLIB_FW_T_F32 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FW state.

2.8.3.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_FW_F32.

Figure 55. Functions and data structures in AMCLIB_FW_F32

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. All inputs and outputs of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

Table 47. Scaling constants

Scaling constant	Symbol	Calculation
Maximum stator phase voltage [V]	U_{MAX}	$U_{MAX}=U_{DC_Bus_Max}$
Maximum phase current [A]	I_{MAX}	Maximum current of the inverter or nominal current of the motor (whichever is lower).
Maximum speed [rad/s]	Ω_{MAX}	Maximum application required speed, at least the motor electrical rated speed.

An initial estimate of the field weakening PI controller parameters can be taken from the speed PI controller, see AMCLIB_SpeedLoop_F32_Eq4. The upper limit of the controller output shall be set to zero, i.e. pPIpAWFW.f32UpperLimit = 0. The lower limit shall be set to an adequate value from the interval of fractional values [-0.5, 0] which corresponds to the Q1.31-encoded integer range [-1073741824, 0]. It is important to set the lower limit correctly to prevent motor damage by irreversible demagnetization of the permanent magnets. Let I_{D_MAX} be the maximum permitted negative d-axis current for field weakening. Then, to represent the lower limit in integer format expected by the C

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

language, simply shift the radix point by multiplying the lower limit value by the 31st power of two (corresponds to the Q1.31 encoding):

$$pPIpAWFW.f32LowerLimit = \left(\frac{1}{\pi} \cdot \cos^{-1}\left(\frac{|I_{D_MAX}|}{I_{MAX}}\right) - 0.5\right) \cdot 2^{31}$$

Equation AMCLIB_FW_F32_Eq1

Further refinement of the field weakening PI controller parameters should be done interactively based on the observed performance. This can be easily achieved with the FREEMASTER real-time debugging tool. The following points should be considered:

- The field weakening PI controller tweaks the angle of the current space vector I_S to produce the negative flux-producing current component I_D and sets the limits of the torque-producing current component I_Q.
- The magnitude of I_D depends also on the output of the speed PI controller.
- There are two different paths producing the PIpAW input error. The weight of each depends on the motor operation mode.

The performance of the field weakening controller can be affected by measurement noise. The noise immunity can be improved by tweaking the smoothing factor of the moving average filter GDFLIB_FilterMA_F32 which preprocesses the input to the field weakening controller.

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

2.8.3.4 Code Example

```
#include "amclib.h"

AMCLIB_FW_T_F32 FWState;
SWLIBS_2Syst_F32 IDQReq;       // required dq currents
SWLIBS_2Syst_F32 f32IDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_F32 f32UDQReq;    // required dq voltages
tFrac32 f32VelocityReq;        // required velocity
tFrac32 f32VelocityFbck;       // actual velocity
AMCLIB_CURRENT_LOOP_T_F32 CurrentLoop;

void main (void)
{
    tFrac32 f32FilterMAFWOut;
    tFrac32 f32ControllerPIpAWFWOut;

    // Initialize the parameters and pointers in FWState
    FW.pFilterFW.u16NSamples = 3u;
    FW.pPIpAWFW.f32PropGain = (tFrac32)FRAC32(0.2348);
    FW.pPIpAWFW.f32IntegGain = (tFrac32)FRAC32(0.3457);
    FW.pPIpAWFW.s16PropGainShift = 1;
    FW.pPIpAWFW.s16IntegGainShift = 1;
    FW.pPIpAWFW.f32LowerLimit = (tFrac32)-1073741824L;
    FW.pPIpAWFW.f32UpperLimit = (tFrac32)0;
    FW.pIQFbck = &f32IDQFbck.f32Arg2;
    FW.pUQReq = &f32UDQReq.f32Arg2;
    FW.pUQLim = &CurrentLoop.pPIrAWQ.f32UpperLimit;

    // Clear AMCLIB_FW state variables
    // Alternative 1: API call with postfix
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // (only one alternative shall be used).
    AMCLIB_FWInit_F32(&FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWInit(&FWState, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWInit(&FWState);

    // Initialize the AMCLIB_FW state variables to predefined values
    // Warning: Parameters in FWState must be already initialized.
    f32FilterMAFWOut = (tFrac32)123L;
    f32ControllerPIpAWFWOut = (tFrac32)123L;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSetState_F32(f32FilterMAFWOut, f32ControllerPIpAWFWOut,
      &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSetState(f32FilterMAFWOut, f32ControllerPIpAWFWOut,
      &FWState, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWSetState(f32FilterMAFWOut, f32ControllerPIpAWFWOut,
      &FWState);

    f32VelocityReq = (tFrac32)100L;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    tFrac32 f32IDQReqAmp;

    // Calculate f32IDQReqAmp from f32VelocityReq and f32VelocityFbck
    // using AMCLIB_SpeedLoop
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FW_F32(f32IDQReqAmp, f32VelocityFbck, &IDQReq, &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FW(f32IDQReqAmp, f32VelocityFbck, &IDQReq, &FWState, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FW(f32IDQReqAmp, f32VelocityFbck, &IDQReq, &FWState);
}

// Periodical function or interrupt - current control loop
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
void FastLoop(void)
{
    // Calculate new values of f32VelocityFbck, f32IDQFbck, f32UDQReq
    // (...)
}
```

2.8.4 Function AMCLIB_FW_F16

2.8.4.1 Declaration

```
void AMCLIB_FW_F16(tFrac16 f16IDQReqAmp, tFrac16 f16VelocityFbck,
SWLIBS_2Syst_F16 *const pIDQReq, AMCLIB_FW_T_F16 *pCtrl);
```

2.8.4.2 Arguments

Table 48. AMCLIB_FW_F16 arguments

Type	Name	Direction	Description
tFrac16	f16IDQReqAmp	**input**	Required amplitude of the currents Id and Iq in the two-phase rotational orthogonal system (d-q).
tFrac16	f16VelocityFbck	**input**	Actual electrical angular velocity from the feedback.
SWLIBS_2Syst_F16 *const	pIDQReq	**input, output**	Pointer to the structure with the required stator currents in the two-phase rotational orthogonal system (d-q).
AMCLIB_FW_T_F16 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FW state.

2.8.4.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_FW_F16.

Figure 56. Functions and data structures in AMCLIB_FW_F16

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. All inputs and outputs of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

Table 49. Scaling constants

Scaling constant	Symbol	Calculation
Maximum stator phase voltage [V]	U_{MAX}	$U_{MAX}=U_{DC_Bus_Max}$

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 49. Scaling constants...*continued*

Maximum phase current [A]	I_{MAX}	Maximum current of the inverter or nominal current of the motor (whichever is lower).
Maximum speed [rad/s]	Ω_{MAX}	Maximum application required speed, at least the motor electrical rated speed.

An initial estimate of the field weakening PI controller parameters can be taken from the speed PI controller, see AMCLIB_SpeedLoop_F16_Eq4. The upper limit of the controller output shall be set to zero, i.e. pPIpAWFW.f16UpperLimit = 0. The lower limit shall be set to an adequate value from the interval of fractional values [-0.5, 0] which corresponds to the Q1.15-encoded integer range [-16384, 0]. It is important to set the lower limit correctly to prevent motor damage by irreversible demagnetization of the permanent magnets. Let I_{D_MAX} be the maximum permitted negative d-axis current for field weakening. Then, to represent the lower limit in integer format expected by the C language, simply shift the radix point by multiplying the lower limit value by the 15th power of two (corresponds to the Q1.15 encoding):

$$pPIpAWFW.f16LowerLimit = \left(\frac{1}{\pi}\cdot\cos^{-1}\left(\frac{|I_{D_MAX}|}{I_{MAX}}\right) - 0.5\right)\cdot 2^{15}$$

Equation AMCLIB_FW_F16_Eq1

Further refinement of the field weakening PI controller parameters should be done interactively based on the observed performance. This can be easily achieved with the FREEMASTER real-time debugging tool. The following points should be considered:

- The field weakening PI controller tweaks the angle of the current space vector I_S to produce the negative flux-producing current component I_D and sets the limits of the torque-producing current component I_Q.
- The magnitude of I_D depends also on the output of the speed PI controller.
- There are two different paths producing the PIpAW input error. The weight of each depends on the motor operation mode.

The performance of the field weakening controller can be affected by measurement noise. The noise immunity can be improved by tweaking the smoothing factor of the moving average filter GDFLIB_FilterMA_F16 which preprocesses the input to the field weakening controller.

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

2.8.4.4 Code Example

```
#include "amclib.h"

AMCLIB_FW_T_F16 FWState;
SWLIBS_2Syst_F16 IDQReq;       // required dq currents
SWLIBS_2Syst_F16 f16IDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_F16 f16UDQReq;    // required dq voltages
tFrac16 f16VelocityReq;        // required velocity
tFrac16 f16VelocityFbck;       // actual velocity
AMCLIB_CURRENT_LOOP_T_F16 CurrentLoop;

void main (void)
{
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    tFrac16 f16FilterMAFWOut;
    tFrac16 f16ControllerPIpAWFWOut;

    // Initialize the parameters and pointers in FWState
    FW.pFilterFW.u16NSamples = 3u;
    FW.pPIpAWFW.f16PropGain = (tFrac16)FRAC16(0.2348);
    FW.pPIpAWFW.f16IntegGain = (tFrac16)FRAC16(0.3457);
    FW.pPIpAWFW.s16PropGainShift = 1;
    FW.pPIpAWFW.s16IntegGainShift = 1;
    FW.pPIpAWFW.f16LowerLimit = (tFrac16)-16384;
    FW.pPIpAWFW.f16UpperLimit = (tFrac16)0;
    FW.pIQFbck = &f16IDQFbck.f16Arg2;
    FW.pUQReq = &f16UDQReq.f16Arg2;
    FW.pUQLim = &CurrentLoop.pPIrAWQ.f16UpperLimit;

    // Clear AMCLIB_FW state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWInit_F16(&FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWInit(&FWState, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWInit(&FWState);

    // Initialize the AMCLIB_FW state variables to predefined values
    // Warning: Parameters in FWState must be already initialized.
    f16FilterMAFWOut = (tFrac16)123;
    f16ControllerPIpAWFWOut = (tFrac16)123;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSetState_F16(f16FilterMAFWOut, f16ControllerPIpAWFWOut,
      &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSetState(f16FilterMAFWOut, f16ControllerPIpAWFWOut,
      &FWState, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWSetState(f16FilterMAFWOut, f16ControllerPIpAWFWOut,
      &FWState);

    f16VelocityReq = (tFrac16)100;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    tFrac16 f16IDQReqAmp;

    // Calculate f16IDQReqAmp from f16VelocityReq and f16VelocityFbck
    // using AMCLIB_SpeedLoop
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FW_F16(f16IDQReqAmp, f16VelocityFbck, &IDQReq, &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FW(f16IDQReqAmp, f16VelocityFbck, &IDQReq, &FWState, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FW(f16IDQReqAmp, f16VelocityFbck, &IDQReq, &FWState);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of f16VelocityFbck, f16IDQFbck, f16UDQReq
    // (...)
}
```

2.8.5 Function AMCLIB_FW_FLT

2.8.5.1 Declaration

```
void AMCLIB_FW_FLT(tFloat fltIDQReqAmp, tFloat fltVelocityFbck,
SWLIBS_2Syst_FLT *const pIDQReq, AMCLIB_FW_T_FLT *pCtrl);
```

2.8.5.2 Arguments

Table 50. AMCLIB_FW_FLT arguments

Type	Name	Direction	Description
tFloat	fltIDQReqAmp	**input**	Required amplitude of the currents Id and Iq in the two-phase rotational orthogonal system (d-q).
tFloat	fltVelocityFbck	**input**	Actual electrical angular velocity from the feedback.
SWLIBS_2Syst_FLT *const	pIDQReq	**input, output**	Pointer to the structure with the required stator currents in the two-phase rotational orthogonal system (d-q).
AMCLIB_FW_T_FLT *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FW state.

2.8.5.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_FW_FLT.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 57. Functions and data structures in AMCLIB_FW_FLT

An initial estimate of the field weakening PI controller parameters can be taken from the speed PI controller, see AMCLIB_SpeedLoop_FLT_Eq1. The upper limit of the controller output shall be set to zero, i.e. pPIpAWFW.fltUpperLimit = 0. The lower limit shall be set to an adequate value from the interval [-π/2, 0]. It is important to set the lower limit correctly to prevent motor damage by irreversible demagnetization of the permanent magnets. Let I_{D_MAX} be the maximum permitted negative d-axis current for field weakening, then

$$pPIpAWFW.fltLowerLimit = \cos^{-1}\left(\frac{|I_{D_MAX}|}{I_{MAX}}\right) - \frac{\pi}{2}$$

Equation AMCLIB_FW_FLT_Eq1

Further refinement of the field weakening PI controller parameters should be done interactively based on the observed performance. This can be easily achieved with the FREEMASTER real-time debugging tool. The following points should be considered:

- The field weakening PI controller tweaks the angle of the current space vector I_S to produce the negative flux-producing current component I_D and sets the limits of the torque-producing current component I_Q.
- The magnitude of I_D depends also on the output of the speed PI controller.
- There are two different paths producing the PIpAW input error. The weight of each depends on the motor operation mode.

The performance of the field weakening controller can be affected by measurement noise. The noise immunity can be improved by tweaking the smoothing factor of the moving average filter GDFLIB_FilterMA_FLT which preprocesses the input to the field weakening controller.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.8.5.4 Code Example

```
#include "amclib.h"

AMCLIB_FW_T_FLT FWState;
SWLIBS_2Syst_FLT IDQReq;       // required dq currents
SWLIBS_2Syst_FLT fltIDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_FLT fltUDQReq;    // required dq voltages
tFloat fltVelocityReq;         // required velocity
tFloat fltVelocityFbck;        // actual velocity
AMCLIB_CURRENT_LOOP_T_FLT CurrentLoop;

void main (void)
{
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    tFloat fltFilterMAFWOut;
    tFloat fltControllerPIpAWFWOut;

    // Initialize the parameters and pointers in FWState
    FW.pFilterFW.fltLambda = (tFloat)0.6;
    FW.pPIpAWFW.fltPropGain = (tFloat)0.2348;
    FW.pPIpAWFW.fltIntegGain = (tFloat)0.3457;
    FW.pPIpAWFW.fltLowerLimit = (tFloat)-1.570796326794897;
    FW.pPIpAWFW.fltUpperLimit = (tFloat)0.0;
    FW.pIQFbck = &fltIDQFbck.fltArg2;
    FW.pUQReq = &fltUDQReq.fltArg2;
    FW.pUQLim = &CurrentLoop.pPIrAWQ.fltUpperLimit;
    FW.fltUmaxDivImax = (tFloat)1.0;

    // Clear AMCLIB_FW state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWInit_FLT(&FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWInit(&FWState, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWInit(&FWState);

    // Initialize the AMCLIB_FW state variables to predefined values
    // Warning: Parameters in FWState must be already initialized.
    fltFilterMAFWOut = 123.0F;
    fltControllerPIpAWFWOut = 123.0F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSetState_FLT(fltFilterMAFWOut, fltControllerPIpAWFWOut,
      &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSetState(fltFilterMAFWOut, fltControllerPIpAWFWOut,
      &FWState, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWSetState(fltFilterMAFWOut, fltControllerPIpAWFWOut,
      &FWState);

    fltVelocityReq = 100.0F;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    tFloat fltIDQReqAmp;

    // Calculate fltIDQReqAmp from fltVelocityReq and fltVelocityFbck
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // using AMCLIB_SpeedLoop
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FW_FLT(fltIDQReqAmp, fltVelocityFbck, &IDQReq, &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FW(fltIDQReqAmp, fltVelocityFbck, &IDQReq, &FWState, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FW(fltIDQReqAmp, fltVelocityFbck, &IDQReq, &FWState);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of fltVelocityFbck, fltIDQFbck, fltUDQReq
    // (...)
}
```

2.8.6 Function AMCLIB_FWInit

2.8.6.1 Description

This function clears the AMCLIB_FW state variables.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.8.6.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.8.6.3 Function AMCLIB_FWInit_F32

2.8.6.3.1 Declaration

```
void AMCLIB_FWInit_F32(AMCLIB_FW_T_F32 *const pCtrl);
```

2.8.6.3.2 Arguments

Table 51. AMCLIB_FWInit_F32 arguments

Type	Name	Direction	Description
AMCLIB_FW_T_F32 *const	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FW state.

Note: *If pCtrl points to a structure of type AMCLIB_FW_DEBUG_T_F32, it must be recasted to AMCLIB_FW_T_F32 *.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.8.6.3.3 Code Example

```
#include "amclib.h"

AMCLIB_FW_T_F32 FWState;
SWLIBS_2Syst_F32 IDQReq;       // required dq currents
SWLIBS_2Syst_F32 f32IDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_F32 f32UDQReq;    // required dq voltages
tFrac32 f32VelocityReq;        // required velocity
tFrac32 f32VelocityFbck;       // actual velocity
AMCLIB_CURRENT_LOOP_T_F32 CurrentLoop;

void main (void)
{
    tFrac32 f32FilterMAFWOut;
    tFrac32 f32ControllerPIpAWFWOut;

    // Initialize the parameters and pointers in FWState
    FW.pFilterFW.u16NSamples = 3u;
    FW.pPIpAWFW.f32PropGain = (tFrac32)FRAC32(0.2348);
    FW.pPIpAWFW.f32IntegGain = (tFrac32)FRAC32(0.3457);
    FW.pPIpAWFW.s16PropGainShift = 1;
    FW.pPIpAWFW.s16IntegGainShift = 1;
    FW.pPIpAWFW.f32LowerLimit = (tFrac32)-1073741824L;
    FW.pPIpAWFW.f32UpperLimit = (tFrac32)0;
    FW.pIQFbck = &f32IDQFbck.f32Arg2;
    FW.pUQReq = &f32UDQReq.f32Arg2;
    FW.pUQLim = &CurrentLoop.pPIrAWQ.f32UpperLimit;

    // Clear AMCLIB_FW state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWInit_F32(&FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWInit(&FWState, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWInit(&FWState);

    // Initialize the AMCLIB_FW state variables to predefined values
    // Warning: Parameters in FWState must be already initialized.
    f32FilterMAFWOut = (tFrac32)123L;
    f32ControllerPIpAWFWOut = (tFrac32)123L;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSetState_F32(f32FilterMAFWOut, f32ControllerPIpAWFWOut,
      &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSetState(f32FilterMAFWOut, f32ControllerPIpAWFWOut,
      &FWState, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    AMCLIB_FWSetState(f32FilterMAFWOut, f32ControllerPIpAWFWOut,
      &FWState);

    f32VelocityReq = (tFrac32)100L;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    tFrac32 f32IDQReqAmp;

    // Calculate f32IDQReqAmp from f32VelocityReq and f32VelocityFbck
    // using AMCLIB_SpeedLoop
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FW_F32(f32IDQReqAmp, f32VelocityFbck, &IDQReq, &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FW(f32IDQReqAmp, f32VelocityFbck, &IDQReq, &FWState, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FW(f32IDQReqAmp, f32VelocityFbck, &IDQReq, &FWState);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of f32VelocityFbck, f32IDQFbck, f32UDQReq
    // (...)
}
```

2.8.6.4 Function AMCLIB_FWInit_F16

2.8.6.4.1 Declaration

```
void AMCLIB_FWInit_F16(AMCLIB_FW_T_F16 *const pCtrl);
```

2.8.6.4.2 Arguments

Table 52. AMCLIB_FWInit_F16 arguments

Type	Name	Direction	Description
AMCLIB_FW_T_F16 *const	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FW state.

Note: *If pCtrl points to a structure of type AMCLIB_FW_DEBUG_T_F16, it must be recasted to AMCLIB_FW_T_F16 *.*

2.8.6.4.3 Code Example

```
#include "amclib.h"

AMCLIB_FW_T_F16 FWState;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
SWLIBS_2Syst_F16 IDQReq;       // required dq currents
SWLIBS_2Syst_F16 f16IDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_F16 f16UDQReq;    // required dq voltages
tFrac16 f16VelocityReq;        // required velocity
tFrac16 f16VelocityFbck;       // actual velocity
AMCLIB_CURRENT_LOOP_T_F16 CurrentLoop;

void main (void)
{
    tFrac16 f16FilterMAFWOut;
    tFrac16 f16ControllerPIpAWFWOut;

    // Initialize the parameters and pointers in FWState
    FW.pFilterFW.u16NSamples = 3u;
    FW.pPIpAWFW.f16PropGain = (tFrac16)FRAC16(0.2348);
    FW.pPIpAWFW.f16IntegGain = (tFrac16)FRAC16(0.3457);
    FW.pPIpAWFW.s16PropGainShift = 1;
    FW.pPIpAWFW.s16IntegGainShift = 1;
    FW.pPIpAWFW.f16LowerLimit = (tFrac16)-16384;
    FW.pPIpAWFW.f16UpperLimit = (tFrac16)0;
    FW.pIQFbck = &f16IDQFbck.f16Arg2;
    FW.pUQReq = &f16UDQReq.f16Arg2;
    FW.pUQLim = &CurrentLoop.pPIrAWQ.f16UpperLimit;

    // Clear AMCLIB_FW state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWInit_F16(&FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWInit(&FWState, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWInit(&FWState);

    // Initialize the AMCLIB_FW state variables to predefined values
    // Warning: Parameters in FWState must be already initialized.
    f16FilterMAFWOut = (tFrac16)123;
    f16ControllerPIpAWFWOut = (tFrac16)123;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSetState_F16(f16FilterMAFWOut, f16ControllerPIpAWFWOut,
      &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSetState(f16FilterMAFWOut, f16ControllerPIpAWFWOut,
      &FWState, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWSetState(f16FilterMAFWOut, f16ControllerPIpAWFWOut,
      &FWState);

    f16VelocityReq = (tFrac16)100;
    while(1);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    tFrac16 f16IDQReqAmp;

    // Calculate f16IDQReqAmp from f16VelocityReq and f16VelocityFbck
    // using AMCLIB_SpeedLoop
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FW_F16(f16IDQReqAmp, f16VelocityFbck, &IDQReq, &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FW(f16IDQReqAmp, f16VelocityFbck, &IDQReq, &FWState, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FW(f16IDQReqAmp, f16VelocityFbck, &IDQReq, &FWState);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of f16VelocityFbck, f16IDQFbck, f16UDQReq
    // (...)
}
```

2.8.6.5 Function AMCLIB_FWInit_FLT

2.8.6.5.1 Declaration

```
void AMCLIB_FWInit_FLT(AMCLIB_FW_T_FLT *const pCtrl);
```

2.8.6.5.2 Arguments

Table 53. AMCLIB_FWInit_FLT arguments

Type	Name	Direction	Description
AMCLIB_FW_T_FLT *const	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FW state.

Note: *If pCtrl points to a structure of type AMCLIB_FW_DEBUG_T_FLT, it must be recasted to AMCLIB_FW_T_FLT *.*

The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.

2.8.6.5.3 Code Example

```
#include "amclib.h"

AMCLIB_FW_T_FLT FWState;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
SWLIBS_2Syst_FLT IDQReq;       // required dq currents
SWLIBS_2Syst_FLT fltIDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_FLT fltUDQReq;    // required dq voltages
tFloat fltVelocityReq;         // required velocity
tFloat fltVelocityFbck;        // actual velocity
AMCLIB_CURRENT_LOOP_T_FLT CurrentLoop;

void main (void)
{
    tFloat fltFilterMAFWOut;
    tFloat fltControllerPIpAWFWOut;

    // Initialize the parameters and pointers in FWState
    FW.pFilterFW.fltLambda = (tFloat)0.6;
    FW.pPIpAWFW.fltPropGain = (tFloat)0.2348;
    FW.pPIpAWFW.fltIntegGain = (tFloat)0.3457;
    FW.pPIpAWFW.fltLowerLimit = (tFloat)-1.570796326794897;
    FW.pPIpAWFW.fltUpperLimit = (tFloat)0.0;
    FW.pIQFbck = &fltIDQFbck.fltArg2;
    FW.pUQReq = &fltUDQReq.fltArg2;
    FW.pUQLim = &CurrentLoop.pPIrAWQ.fltUpperLimit;
    FW.fltUmaxDivImax = (tFloat)1.0;

    // Clear AMCLIB_FW state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWInit_FLT(&FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWInit(&FWState, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWInit(&FWState);

    // Initialize the AMCLIB_FW state variables to predefined values
    // Warning: Parameters in FWState must be already initialized.
    fltFilterMAFWOut = 123.0F;
    fltControllerPIpAWFWOut = 123.0F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSetState_FLT(fltFilterMAFWOut, fltControllerPIpAWFWOut,
      &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSetState(fltFilterMAFWOut, fltControllerPIpAWFWOut,
      &FWState, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWSetState(fltFilterMAFWOut, fltControllerPIpAWFWOut,
      &FWState);

    fltVelocityReq = 100.0F;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    tFloat fltIDQReqAmp;

    // Calculate fltIDQReqAmp from fltVelocityReq and fltVelocityFbck
    // using AMCLIB_SpeedLoop
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FW_FLT(fltIDQReqAmp, fltVelocityFbck, &IDQReq, &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FW(fltIDQReqAmp, fltVelocityFbck, &IDQReq, &FWState, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FW(fltIDQReqAmp, fltVelocityFbck, &IDQReq, &FWState);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of fltVelocityFbck, fltIDQFbck, fltUDQReq
    // (...)
}
```

2.8.7 Function AMCLIB_FWSetState

2.8.7.1 Description

This function initializes the AMCLIB_FW state variables to achieve the required output values.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.8.7.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.8.7.3 Function AMCLIB_FWSetState_F32

2.8.7.3.1 Declaration

```
void AMCLIB_FWSetState_F32(tFrac32 f32FilterMAFWOut, tFrac32
f32ControllerPIpAWFWOut, AMCLIB_FW_T_F32 *pCtrl);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.8.7.3.2 Arguments

Table 54. AMCLIB_FWSetState_F32 arguments

Type	Name	Direction	Description
tFrac32	f32FilterMAFWOut	**input**	Required output of the FilterMA.
tFrac32	f32ControllerPIpAWFWOut	**input**	Required output of the ControllerPIpAW.
AMCLIB_FW_T_F32 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FW state.

CAUTION: *Set the parameters in the structure pointed to by pCtrl before calling this function.*

Note: *If pCtrl points to a structure of type AMCLIB_FW_DEBUG_T_F32, it must be recasted to AMCLIB_FW_T_F32 *.*

2.8.7.3.3 Code Example

```
#include "amclib.h"

AMCLIB_FW_T_F32 FWState;
SWLIBS_2Syst_F32 IDQReq;       // required dq currents
SWLIBS_2Syst_F32 f32IDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_F32 f32UDQReq;    // required dq voltages
tFrac32 f32VelocityReq;        // required velocity
tFrac32 f32VelocityFbck;       // actual velocity
AMCLIB_CURRENT_LOOP_T_F32 CurrentLoop;

void main (void)
{
    tFrac32 f32FilterMAFWOut;
    tFrac32 f32ControllerPIpAWFWOut;

    // Initialize the parameters and pointers in FWState
    FW.pFilterFW.u16NSamples = 3u;
    FW.pPIpAWFW.f32PropGain = (tFrac32)FRAC32(0.2348);
    FW.pPIpAWFW.f32IntegGain = (tFrac32)FRAC32(0.3457);
    FW.pPIpAWFW.s16PropGainShift = 1;
    FW.pPIpAWFW.s16IntegGainShift = 1;
    FW.pPIpAWFW.f32LowerLimit = (tFrac32)-1073741824L;
    FW.pPIpAWFW.f32UpperLimit = (tFrac32)0;
    FW.pIQFbck = &f32IDQFbck.f32Arg2;
    FW.pUQReq = &f32UDQReq.f32Arg2;
    FW.pUQLim = &CurrentLoop.pPIrAWQ.f32UpperLimit;

    // Clear AMCLIB_FW state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWInit_F32(&FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWInit(&FWState, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWInit(&FWState);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Initialize the AMCLIB_FW state variables to predefined values
    // Warning: Parameters in FWState must be already initialized.
    f32FilterMAFWOut = (tFrac32)123L;
    f32ControllerPIpAWFWOut = (tFrac32)123L;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSetState_F32(f32FilterMAFWOut, f32ControllerPIpAWFWOut,
      &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSetState(f32FilterMAFWOut, f32ControllerPIpAWFWOut,
      &FWState, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWSetState(f32FilterMAFWOut, f32ControllerPIpAWFWOut,
      &FWState);

    f32VelocityReq = (tFrac32)100L;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    tFrac32 f32IDQReqAmp;

    // Calculate f32IDQReqAmp from f32VelocityReq and f32VelocityFbck
    // using AMCLIB_SpeedLoop
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FW_F32(f32IDQReqAmp, f32VelocityFbck, &IDQReq, &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FW(f32IDQReqAmp, f32VelocityFbck, &IDQReq, &FWState, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FW(f32IDQReqAmp, f32VelocityFbck, &IDQReq, &FWState);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of f32VelocityFbck, f32IDQFbck, f32UDQReq
    // (...)
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.8.7.4 Function AMCLIB_FWSetState_F16

2.8.7.4.1 Declaration

```
void AMCLIB_FWSetState_F16(tFrac16 f16FilterMAFWOut, tFrac16
f16ControllerPIpAWFWOut, AMCLIB_FW_T_F16 *pCtrl);
```

2.8.7.4.2 Arguments

Table 55. AMCLIB_FWSetState_F16 arguments

Type	Name	Direction	Description
tFrac16	f16FilterMAFWOut	**input**	Required output of the FilterMA.
tFrac16	f16ControllerPIpAWFWOut	**input**	Required output of the ControllerPIpAW.
AMCLIB_FW_T_F16 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FW state.

CAUTION: *Set the parameters in the structure pointed to by pCtrl before calling this function.*

Note: *If pCtrl points to a structure of type* AMCLIB_FW_DEBUG_T_F16*, it must be recasted to* AMCLIB_FW_T_F16 *.*

2.8.7.4.3 Code Example

```
#include "amclib.h"

AMCLIB_FW_T_F16 FWState;
SWLIBS_2Syst_F16 IDQReq;       // required dq currents
SWLIBS_2Syst_F16 f16IDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_F16 f16UDQReq;    // required dq voltages
tFrac16 f16VelocityReq;        // required velocity
tFrac16 f16VelocityFbck;       // actual velocity
AMCLIB_CURRENT_LOOP_T_F16 CurrentLoop;

void main (void)
{
    tFrac16 f16FilterMAFWOut;
    tFrac16 f16ControllerPIpAWFWOut;

    // Initialize the parameters and pointers in FWState
    FW.pFilterFW.u16NSamples = 3u;
    FW.pPIpAWFW.f16PropGain = (tFrac16)FRAC16(0.2348);
    FW.pPIpAWFW.f16IntegGain = (tFrac16)FRAC16(0.3457);
    FW.pPIpAWFW.s16PropGainShift = 1;
    FW.pPIpAWFW.s16IntegGainShift = 1;
    FW.pPIpAWFW.f16LowerLimit = (tFrac16)-16384;
    FW.pPIpAWFW.f16UpperLimit = (tFrac16)0;
    FW.pIQFbck = &f16IDQFbck.f16Arg2;
    FW.pUQReq = &f16UDQReq.f16Arg2;
    FW.pUQLim = &CurrentLoop.pPIrAWQ.f16UpperLimit;

    // Clear AMCLIB_FW state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWInit_F16(&FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    AMCLIB_FWInit(&FWState, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWInit(&FWState);

    // Initialize the AMCLIB_FW state variables to predefined values
    // Warning: Parameters in FWState must be already initialized.
    f16FilterMAFWOut = (tFrac16)123;
    f16ControllerPIpAWFWOut = (tFrac16)123;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSetState_F16(f16FilterMAFWOut, f16ControllerPIpAWFWOut,
      &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSetState(f16FilterMAFWOut, f16ControllerPIpAWFWOut,
      &FWState, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWSetState(f16FilterMAFWOut, f16ControllerPIpAWFWOut,
      &FWState);

    f16VelocityReq = (tFrac16)100;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    tFrac16 f16IDQReqAmp;

    // Calculate f16IDQReqAmp from f16VelocityReq and f16VelocityFbck
    // using AMCLIB_SpeedLoop
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FW_F16(f16IDQReqAmp, f16VelocityFbck, &IDQReq, &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FW(f16IDQReqAmp, f16VelocityFbck, &IDQReq, &FWState, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FW(f16IDQReqAmp, f16VelocityFbck, &IDQReq, &FWState);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of f16VelocityFbck, f16IDQFbck, f16UDQReq
    // (...)
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.8.7.5 Function AMCLIB_FWSetState_FLT

2.8.7.5.1 Declaration

```
void AMCLIB_FWSetState_FLT(tFloat fltFilterMAFWOut, tFloat
fltControllerPIpAWFWOut, AMCLIB_FW_T_FLT *pCtrl);
```

2.8.7.5.2 Arguments

Table 56. AMCLIB_FWSetState_FLT arguments

Type	Name	Direction	Description
tFloat	fltFilterMAFWOut	**input**	Required output of the FilterMA.
tFloat	fltControllerPIpAWFWOut	**input**	Required output of the ControllerPIpAW.
AMCLIB_FW_T_FLT *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FW state.

CAUTION: *Set the parameters in the structure pointed to by pCtrl before calling this function.*

Note: *If pCtrl points to a structure of type AMCLIB_FW_DEBUG_T_FLT, it must be recasted to AMCLIB_FW_T_FLT *.*

The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.

2.8.7.5.3 Code Example

```
#include "amclib.h"

AMCLIB_FW_T_FLT FWState;
SWLIBS_2Syst_FLT IDQReq;       // required dq currents
SWLIBS_2Syst_FLT fltIDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_FLT fltUDQReq;    // required dq voltages
tFloat fltVelocityReq;         // required velocity
tFloat fltVelocityFbck;        // actual velocity
AMCLIB_CURRENT_LOOP_T_FLT CurrentLoop;

void main (void)
{
    tFloat fltFilterMAFWOut;
    tFloat fltControllerPIpAWFWOut;

    // Initialize the parameters and pointers in FWState
    FW.pFilterFW.fltLambda = (tFloat)0.6;
    FW.pPIpAWFW.fltPropGain = (tFloat)0.2348;
    FW.pPIpAWFW.fltIntegGain = (tFloat)0.3457;
    FW.pPIpAWFW.fltLowerLimit = (tFloat)-1.570796326794897;
    FW.pPIpAWFW.fltUpperLimit = (tFloat)0.0;
    FW.pIQFbck = &fltIDQFbck.fltArg2;
    FW.pUQReq = &fltUDQReq.fltArg2;
    FW.pUQLim = &CurrentLoop.pPIrAWQ.fltUpperLimit;
    FW.fltUmaxDivImax = (tFloat)1.0;

    // Clear AMCLIB_FW state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    AMCLIB_FWInit_FLT(&FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWInit(&FWState, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWInit(&FWState);

    // Initialize the AMCLIB_FW state variables to predefined values
    // Warning: Parameters in FWState must be already initialized.
    fltFilterMAFWOut = 123.0F;
    fltControllerPIpAWFWOut = 123.0F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSetState_FLT(fltFilterMAFWOut, fltControllerPIpAWFWOut,
      &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSetState(fltFilterMAFWOut, fltControllerPIpAWFWOut,
      &FWState, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWSetState(fltFilterMAFWOut, fltControllerPIpAWFWOut,
      &FWState);

    fltVelocityReq = 100.0F;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    tFloat fltIDQReqAmp;

    // Calculate fltIDQReqAmp from fltVelocityReq and fltVelocityFbck
    // using AMCLIB_SpeedLoop
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FW_FLT(fltIDQReqAmp, fltVelocityFbck, &IDQReq, &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FW(fltIDQReqAmp, fltVelocityFbck, &IDQReq, &FWState, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FW(fltIDQReqAmp, fltVelocityFbck, &IDQReq, &FWState);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of fltVelocityFbck, fltIDQFbck, fltUDQReq
    // (...)
}
```

2.8.8 Function AMCLIB_FWDebug

This function implements the PMSM Field Weakening controller. Debugging information is provided.

2.8.8.1 Description

AMCLIB_FWDebug implements the field weakening controller in the outer control loop highlighted in Figure 58 . AMCLIB_FWDebug is intended to be used in combination with AMCLIB_SpeedLoopDebug. The code can be simplified further by utilizing AMCLIB_FWSpeedLoopDebug which combines the speed controller with the field weakening controller in one library function. AMCLIB_FWDebug provides the same functionality as AMCLIB_FW. Additionally,this function allows debugging of all internal variables. The debugging outputs are provided in the structure pointed to by `pCtrl`. Replace AMCLIB_FWDebug by AMCLIB_FW once the debugging is finished.

Figure 58. Principal Field Oriented Control scheme with AMCLIB_FW

Field weakening technique employed in AMCLIB_FWDebug extends the speed range of PMSM beyond the motor base speed by reducing the linkage magnetic flux through negative current id_req. See AMCLIB_FW for more details.

2.8.8.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.8.8.3 Function AMCLIB_FWDebug_F32

2.8.8.3.1 Declaration

```
void AMCLIB_FWDebug_F32(tFrac32 f32IDQReqAmp, tFrac32
f32VelocityFbck, SWLIBS_2Syst_F32 *const pIDQReq,
AMCLIB_FW_DEBUG_T_F32 *pCtrl);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.8.8.3.2 Arguments

Table 57. AMCLIB_FWDebug_F32 arguments

Type	Name	Direction	Description
tFrac32	f32IDQReqAmp	**input**	Required amplitude of the currents Id and Iq in the two-phase rotational orthogonal system (d-q).
tFrac32	f32VelocityFbck	**input**	Actual electrical angular velocity from the feedback.
SWLIBS_2Syst_F32 *const	pIDQReq	**input, output**	Pointer to the structure with the required stator currents in the two-phase rotational orthogonal system (d-q).
AMCLIB_FW_DEBUG_T_F32 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FW state and debugging information.

2.8.8.3.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_FWDebug_F32.

Figure 59. Functions and data structures in AMCLIB_FWDebug_F32

Refer to the description of AMCLIB_FW_F32 function on how to set up the controller parameters.

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

2.8.8.3.4 Code Example

```
#include "amclib.h"

AMCLIB_FW_DEBUG_T_F32 FWState;
SWLIBS_2Syst_F32 IDQReq;        // required dq currents
SWLIBS_2Syst_F32 f32IDQFbck;    // calculated dq currents from the feedback
SWLIBS_2Syst_F32 f32UDQReq;     // required dq voltages
tFrac32 f32VelocityReq;         // required velocity
tFrac32 f32VelocityFbck;        // actual velocity
AMCLIB_CURRENT_LOOP_T_F32 CurrentLoop;

void main (void)
{
    tFrac32 f32FilterMAFWOut;
    tFrac32 f32ControllerPIpAWFWOut;

    // Initialize the parameters and pointers in FWState
    FW.pFilterFW.u16NSamples = 3u;
    FW.pPIpAWFW.f32PropGain = (tFrac32)FRAC32(0.2348);
    FW.pPIpAWFW.f32IntegGain = (tFrac32)FRAC32(0.3457);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    FW.pPIpAWFW.s16PropGainShift = 1;
    FW.pPIpAWFW.s16IntegGainShift = 1;
    FW.pPIpAWFW.f32LowerLimit = (tFrac32)-1073741824L;
    FW.pPIpAWFW.f32UpperLimit = (tFrac32)0;
    FW.pIQFbck = &f32IDQFbck.f32Arg2;
    FW.pUQReq = &f32UDQReq.f32Arg2;
    FW.pUQLim = &CurrentLoop.pPIrAWQ.f32UpperLimit;

    // Clear AMCLIB_FW state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWInit_F32((AMCLIB_FW_T_F32 *)&FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWInit((AMCLIB_FW_T_F32 *)&FWState, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWInit((AMCLIB_FW_T_F32 *)&FWState);

    // Initialize the AMCLIB_FW state variables to predefined values
    // Warning: Parameters in FWState must be already initialized.
    f32FilterMAFWOut = (tFrac32)123L;
    f32ControllerPIpAWFWOut = (tFrac32)123L;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSetState_F32(f32FilterMAFWOut, f32ControllerPIpAWFWOut,
      (AMCLIB_FW_T_F32 *)&FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSetState(f32FilterMAFWOut, f32ControllerPIpAWFWOut,
      (AMCLIB_FW_T_F32 *)&FWState, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWSetState(f32FilterMAFWOut, f32ControllerPIpAWFWOut,
      (AMCLIB_FW_T_F32 *)&FWState);

    f32VelocityReq = (tFrac32)100L;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    tFrac32 f32IDQReqAmp;

    // Calculate f32IDQReqAmp from f32VelocityReq and f32VelocityFbck
    // using AMCLIB_SpeedLoop
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWDebug_F32(f32IDQReqAmp, f32VelocityFbck, &IDQReq, &FWState);

    // Alternative 2: API call with implementation parameter
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // (only one alternative shall be used).
    AMCLIB_FWDebug(f32IDQReqAmp, f32VelocityFbck, &IDQReq, &FWState, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWDebug(f32IDQReqAmp, f32VelocityFbck, &IDQReq, &FWState);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of f32VelocityFbck, f32IDQFbck, f32UDQReq
    // (...)
}
```

2.8.8.4 Function AMCLIB_FWDebug_F16

2.8.8.4.1 Declaration

```
void AMCLIB_FWDebug_F16(tFrac16 f16IDQReqAmp, tFrac16
f16VelocityFbck, SWLIBS_2Syst_F16 *const pIDQReq,
AMCLIB_FW_DEBUG_T_F16 *pCtrl);
```

2.8.8.4.2 Arguments

Table 58. AMCLIB_FWDebug_F16 arguments

Type	Name	Direction	Description
tFrac16	f16IDQReqAmp	**input**	Required amplitude of the currents Id and Iq in the two-phase rotational orthogonal system (d-q).
tFrac16	f16VelocityFbck	**input**	Actual electrical angular velocity from the feedback.
SWLIBS_2Syst_F16 *const	pIDQReq	**input, output**	Pointer to the structure with the required stator currents in the two-phase rotational orthogonal system (d-q).
AMCLIB_FW_DEBUG_T_F16 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FW state and debugging information.

2.8.8.4.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_FWDebug_F16.

Figure 60. Functions and data structures in AMCLIB_FWDebug_F16

Refer to the description of AMCLIB_FW_F16 function on how to set up the controller parameters.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

2.8.8.4.4 Code Example

```
#include "amclib.h"

AMCLIB_FW_DEBUG_T_F16 FWState;
SWLIBS_2Syst_F16 IDQReq;       // required dq currents
SWLIBS_2Syst_F16 f16IDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_F16 f16UDQReq;    // required dq voltages
tFrac16 f16VelocityReq;        // required velocity
tFrac16 f16VelocityFbck;       // actual velocity
AMCLIB_CURRENT_LOOP_T_F16 CurrentLoop;

void main (void)
{
    tFrac16 f16FilterMAFWOut;
    tFrac16 f16ControllerPIpAWFWOut;

    // Initialize the parameters and pointers in FWState
    FW.pFilterFW.u16NSamples = 3u;
    FW.pPIpAWFW.f16PropGain = (tFrac16)FRAC16(0.2348);
    FW.pPIpAWFW.f16IntegGain = (tFrac16)FRAC16(0.3457);
    FW.pPIpAWFW.s16PropGainShift = 1;
    FW.pPIpAWFW.s16IntegGainShift = 1;
    FW.pPIpAWFW.f16LowerLimit = (tFrac16)-16384;
    FW.pPIpAWFW.f16UpperLimit = (tFrac16)0;
    FW.pIQFbck = &f16IDQFbck.f16Arg2;
    FW.pUQReq = &f16UDQReq.f16Arg2;
    FW.pUQLim = &CurrentLoop.pPIrAWQ.f16UpperLimit;

    // Clear AMCLIB_FW state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWInit_F16((AMCLIB_FW_T_F16 *)&FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWInit((AMCLIB_FW_T_F16 *)&FWState, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWInit((AMCLIB_FW_T_F16 *)&FWState);

    // Initialize the AMCLIB_FW state variables to predefined values
    // Warning: Parameters in FWState must be already initialized.
    f16FilterMAFWOut = (tFrac16)123;
    f16ControllerPIpAWFWOut = (tFrac16)123;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSetState_F16(f16FilterMAFWOut, f16ControllerPIpAWFWOut,
      (AMCLIB_FW_T_F16 *)&FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSetState(f16FilterMAFWOut, f16ControllerPIpAWFWOut,
      (AMCLIB_FW_T_F16 *)&FWState, F16);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWSetState(f16FilterMAFWOut, f16ControllerPIpAWFWOut,
      (AMCLIB_FW_T_F16 *)&FWState);

    f16VelocityReq = (tFrac16)100;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    tFrac16 f16IDQReqAmp;

    // Calculate f16IDQReqAmp from f16VelocityReq and f16VelocityFbck
    // using AMCLIB_SpeedLoop
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWDebug_F16(f16IDQReqAmp, f16VelocityFbck, &IDQReq, &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWDebug(f16IDQReqAmp, f16VelocityFbck, &IDQReq, &FWState, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWDebug(f16IDQReqAmp, f16VelocityFbck, &IDQReq, &FWState);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of f16VelocityFbck, f16IDQFbck, f16UDQReq
    // (...)
}
```

2.8.8.5 Function AMCLIB_FWDebug_FLT

2.8.8.5.1 Declaration

```
void AMCLIB_FWDebug_FLT(tFloat fltIDQReqAmp, tFloat
fltVelocityFbck, SWLIBS_2Syst_FLT *const pIDQReq,
AMCLIB_FW_DEBUG_T_FLT *pCtrl);
```

2.8.8.5.2 Arguments

Table 59. AMCLIB_FWDebug_FLT arguments

Type	Name	Direction	Description
tFloat	fltIDQReqAmp	**input**	Required amplitude of the currents Id and Iq in the two-phase rotational orthogonal system (d-q).
tFloat	fltVelocityFbck	**input**	Actual electrical angular velocity from the feedback.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 59. AMCLIB_FWDebug_FLT arguments...*continued*

Type	Name	Direction	Description
SWLIBS_2Syst_FLT *const	pIDQReq	**input, output**	Pointer to the structure with the required stator currents in the two-phase rotational orthogonal system (d-q).
AMCLIB_FW_DEBUG_T_FLT *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FW state and debugging information.

2.8.8.5.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_FWDebug_FLT.

Figure 61. Functions and data structures in AMCLIB_FWDebug_FLT

Refer to the description of AMCLIB_FW_FLT function on how to set up the controller parameters.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.8.8.5.4 Code Example

```
#include "amclib.h"

AMCLIB_FW_DEBUG_T_FLT FWState;
SWLIBS_2Syst_FLT IDQReq;       // required dq currents
SWLIBS_2Syst_FLT fltIDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_FLT fltUDQReq;    // required dq voltages
tFloat fltVelocityReq;         // required velocity
tFloat fltVelocityFbck;        // actual velocity
AMCLIB_CURRENT_LOOP_T_FLT CurrentLoop;

void main (void)
{
    tFloat fltFilterMAFWOut;
    tFloat fltControllerPIpAWFWOut;

    // Initialize the parameters and pointers in FWState
    FW.pFilterFW.fltLambda = (tFloat)0.6;
    FW.pPIpAWFW.fltPropGain = (tFloat)0.2348;
    FW.pPIpAWFW.fltIntegGain = (tFloat)0.3457;
    FW.pPIpAWFW.fltLowerLimit = (tFloat)-1.570796326794897;
    FW.pPIpAWFW.fltUpperLimit = (tFloat)0.0;
    FW.pIQFbck = &fltIDQFbck.fltArg2;
    FW.pUQReq = &fltUDQReq.fltArg2;
    FW.pUQLim = &CurrentLoop.pPIrAWQ.fltUpperLimit;
    FW.fltUmaxDivImax = (tFloat)1.0;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Clear AMCLIB_FW state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWInit_FLT((AMCLIB_FW_T_FLT *)&FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWInit((AMCLIB_FW_T_FLT *)&FWState, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWInit((AMCLIB_FW_T_FLT *)&FWState);

    // Initialize the AMCLIB_FW state variables to predefined values
    // Warning: Parameters in FWState must be already initialized.
    fltFilterMAFWOut = 123.0F;
    fltControllerPIpAWFWOut = 123.0F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSetState_FLT(fltFilterMAFWOut, fltControllerPIpAWFWOut,
      (AMCLIB_FW_T_FLT *)&FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSetState(fltFilterMAFWOut, fltControllerPIpAWFWOut,
      (AMCLIB_FW_T_FLT *)&FWState, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWSetState(fltFilterMAFWOut, fltControllerPIpAWFWOut,
      (AMCLIB_FW_T_FLT *)&FWState);

    fltVelocityReq = 100.0F;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    tFloat fltIDQReqAmp;

    // Calculate fltIDQReqAmp from fltVelocityReq and fltVelocityFbck
    // using AMCLIB_SpeedLoop
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWDebug_FLT(fltIDQReqAmp, fltVelocityFbck, &IDQReq, &FWState);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWDebug(fltIDQReqAmp, fltVelocityFbck, &IDQReq, &FWState, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWDebug(fltIDQReqAmp, fltVelocityFbck, &IDQReq, &FWState);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of fltVelocityFbck, fltIDQFbck, fltUDQReq
    // (...)
}
```

2.9 Function AMCLIB_FWSpeedLoop

This function implements the speed PI controller in the FOC outer control loop and the field-weakening algorithm for permanent magnet synchronous motors.

2.9.1 Description

This library function implements a portion of Field Oriented Control (FOC) algorithm. FOC (also called vector control) is a widely used control strategy for Permanent Magnet Synchronous Motors (PMSM). FOC is characterized by smooth rotation over the entire speed range of the motor, full torque control at zero speed, and fast acceleration/deceleration. FOC consists of a hierarchical cascade of inner current loop and outer speed loop. PI controllers within these closed loops maintain the required speed and torque based on feedback measurements.

AMCLIB_FWSpeedLoop implements the speed PI controller and the field weakening controller in the outer control loop highlighted in Figure 62 . AMCLIB_FWSpeedLoop combines the functionalities of AMCLIB_FW and AMCLIB_SpeedLoop in a more integrated form to simplify the application code and improve execution speed. AMCLIB_FWSpeedLoop does not allow debugging of all internal variables. Use AMCLIB_FWSpeedLoopDebug for debugging purposes and replace it with AMCLIB_FWSpeedLoop once the debugging is finished.

Figure 62. Principal Field Oriented Control scheme with AMCLIB_FWSpeedLoop

Field weakening technique employed in AMCLIB_FWSpeedLoop extends the speed range of PMSM beyond the motor base speed by reducing the linkage magnetic flux through negative current id_req. See AMCLIB_FW for more details.

This algorithm is protected by US Patent No. US 2011/0050152 A1.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Before using the FOC with a particular motor, the user needs to provide a set of coefficients through the `pCtrl` input pointer. The controller coefficient values can be calculated from motor parameters.

Refer to the following resources to find out how the NXP motor control tuning and debugging tools for NXP microcontrollers can help you deploy the FOC in your application:

- AN4642 - Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM
- FREEMASTER - FreeMASTER Run-Time Debugging Tool

CAUTION:

1. *A motor operated at speeds over the base speed region generates higher back-EMF voltage than the supply stator voltage. By applying the field weakening technique, the back-EMF is actively kept under control, i.e. lower than the stator voltage. It is dangerous to break the control loop during the field weakening operation. A loss of control (e.g. due to a fault state or turning the application off while running) may result in damage of the electronics and hardware due to the excessive back-EMF which would no longer be suppressed by the control algorithm.*
2. *The field is weakened by applying a negative current d-component. Too high negative current can cause magnet damage by its demagnetization. Make sure to set a correct lower limit of the field weakening PI controller to prevent damage to the motor.*

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.9.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.9.3 Function AMCLIB_FWSpeedLoop_F32

2.9.3.1 Declaration

```
void AMCLIB_FWSpeedLoop_F32(tFrac32 f32VelocityReq,
tFrac32 f32VelocityFbck, SWLIBS_2Syst_F32 *const pIDQReq,
AMCLIB_FW_SPEED_LOOP_T_F32 *pCtrl);
```

2.9.3.2 Arguments

Table 60. AMCLIB_FWSpeedLoop_F32 arguments

Type	Name	Direction	Description
tFrac32	f32VelocityReq	**input**	Required electrical angular velocity (setpoint).
tFrac32	f32VelocityFbck	**input**	Actual electrical angular velocity from the feedback.
SWLIBS_2Syst_F32 *const	pIDQReq	**input, output**	Pointer to the structure with the required stator currents in the two-phase rotational orthogonal system (d-q).
AMCLIB_FW_SPEED_LOOP_T_F32 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FWSpeedLoop state.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.9.3.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_FWSpeedLoop_F32.

Figure 63. Functions and data structures in AMCLIB_FWSpeedLoop_F32

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. All inputs and outputs of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

Table 61. Scaling constants

Scaling constant	Symbol	Calculation
Maximum stator phase voltage [V]	U_{MAX}	$U_{MAX}=U_{DC_Bus_Max}$
Maximum phase current [A]	I_{MAX}	Maximum current of the inverter or nominal current of the motor (whichever is lower).
Maximum speed [rad/s]	Ω_{MAX}	Maximum application required speed, at least the motor electrical rated speed.

The proportional gain K_P and integral gain K_I of the speed controller (GFLIB_ControllerPIpAW_F32) can be calculated as follows:

$$K_P = 2 \cdot \xi \cdot \omega_0 \cdot \frac{J}{K_T}$$

$$K_I = \omega_0^2 \cdot \frac{J}{K_T}$$

Equation AMCLIB_FWSpeedLoop_F32_Eq1

where ξ is the speed loop attenuation, ω_0 is the speed loop natural frequency [rad/s], J is the moment of inertia [kg·m^2], K_T is the motor torque constant [N·m·A^{-1}].

The continuous-time controller parameters K_P and K_I are then discretized using the bilinear transform. The resulting difference equation coefficients must be scaled so that the input and output values of the controller fit in the fractional range.

$$K_{P_SC} = K_P \cdot \frac{\Omega_{MAX}}{I_{MAX}}$$

$$K_{I_SC} = \frac{K_I T_S}{2} \cdot \frac{\Omega_{MAX}}{I_{MAX}}$$

Equation AMCLIB_FWSpeedLoop_F32_Eq2

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

T_S is the sampling period [s]. In case the coefficients K_{P_SC} and K_{I_SC} exceed the fractional range [-1, 1) or they are extremely small (which could lead to loss of precision), their value must be bit-shifted by *NShiftP* and *NShiftI*, respectively. The shift amount can be calculated as

$$NShiftP = \lceil \log_2(|K_{P_SC}|) \rceil$$
$$NShiftI = \lceil \log_2(|K_{I_SC}|) \rceil$$

Equation AMCLIB_FWSpeedLoop_F32_Eq3

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 31st power of two (corresponds to the Q1.31 encoding):

$$pPIpAWQ.f32PropGain = \left(K_{P_SC} \cdot 2^{-NShiftP}\right) \cdot 2^{31}$$
$$pPIpAWQ.f32IntegGain = \left(K_{I_SC} \cdot 2^{-NShiftI}\right) \cdot 2^{31}$$
$$pPIpAWQ.s16PropGainShift = NShiftP$$
$$pPIpAWQ.s16IntegGainShift = NShiftI$$
$$pPIpAWQ.f32UpperLimit = 2^{31} - 1$$
$$pPIpAWQ.f32LowerLimit = -2^{31}$$

Equation AMCLIB_FWSpeedLoop_F32_Eq4

The smoothing factor of the GDFLIB_FilterMA_F32 and the slope of the GFLIB_Ramp_F32 on the input of the speed controller should reflect the achievable dynamics of the drive. The AN4642 (Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM) can help with tuning of these parameters.

An initial estimate of the field weakening PI controller parameters can be taken from the speed PI controller as defined in the above AMCLIB_FWSpeedLoop_F32_Eq4. The upper limit of the field weakening controller output shall be set to zero, i.e. pPIpAWFW.f32UpperLimit = 0. The lower limit shall be set to an adequate value from the interval of fractional values [-0.5, 0] which corresponds to the Q1.31-encoded integer range [-1073741824, 0]. It is important to set the lower limit correctly to prevent motor damage by irreversible demagnetization of the permanent magnets. Let I_{D_MAX} be the maximum permitted negative d-axis current for field weakening. Then, to represent the lower limit in integer format expected by the C language, simply shift the radix point by multiplying the lower limit value by the 31st power of two (corresponds to the Q1.31 encoding):

$$pPIpAWFW.f32LowerLimit = \left(\frac{1}{\pi} \cdot \cos^{-1}\left(\frac{|I_{D_MAX}|}{I_{MAX}}\right) - 0.5\right) \cdot 2^{31}$$

Equation AMCLIB_FWSpeedLoop_F32_Eq5

Further refinement of the field weakening controller parameters should be done interactively based on the observed performance. This can be easily achieved with the FREEMASTER real-time debugging tool. The following points should be considered:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

- The field weakening PI controller tweaks the angle of the current space vector I_S to produce the negative flux-producing current component I_D and sets the limits of the torque-producing current component I_Q.
- The magnitude of I_D depends also on the output of the speed PI controller.
- There are two different paths producing the PI input error. The weight of each depends on the motor operation mode.

The performance of the field weakening controller can be affected by measurement noise. The noise immunity can be improved by tweaking the smoothing factor of the moving average filter GDFLIB_FilterMA_F32 which preprocesses the input to the field weakening controller.

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

2.9.3.4 Code Example

```
#include "amclib.h"

AMCLIB_FW_SPEED_LOOP_T_F32 FWSpeedLoop;
SWLIBS_2Syst_F32 IDQReq;       // required dq currents
SWLIBS_2Syst_F32 f32IDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_F32 f32UDQReq;    // required dq voltages
tFrac32 f32VelocityReq;        // required velocity
tFrac32 f32VelocityFbck;       // actual velocity
AMCLIB_CURRENT_LOOP_T_F32 CurrentLoop;

void main (void)
{
    tFrac32 f32FilterMAWOut;
    tFrac32 f32FilterMAFWOut;
    tFrac32 f32ControllerPIpAWQOut;
    tFrac32 f32ControllerPIpAWFWOut;
    tFrac32 f32RampOut;

    // Initialize the parameters and pointers in FWSpeedLoop
    FWSpeedLoop.pFilterW.u16NSamples = 3u;
    FWSpeedLoop.pFilterFW.u16NSamples = 3u;
    FWSpeedLoop.pPIpAWQ.f32PropGain = (tFrac32)FRAC32(0.1234);
    FWSpeedLoop.pPIpAWQ.f32IntegGain = (tFrac32)FRAC32(0.1675);
    FWSpeedLoop.pPIpAWQ.s16PropGainShift = 1;
    FWSpeedLoop.pPIpAWQ.s16IntegGainShift = 1;
    FWSpeedLoop.pPIpAWQ.f32LowerLimit = (tFrac32)-2006823469L;
    FWSpeedLoop.pPIpAWQ.f32UpperLimit = (tFrac32)2101527497L;
    FWSpeedLoop.pPIpAWFW.f32PropGain = (tFrac32)FRAC32(0.2348);
    FWSpeedLoop.pPIpAWFW.f32IntegGain = (tFrac32)FRAC32(0.3457);
    FWSpeedLoop.pPIpAWFW.s16PropGainShift = 1;
    FWSpeedLoop.pPIpAWFW.s16IntegGainShift = 1;
    FWSpeedLoop.pPIpAWFW.f32LowerLimit = (tFrac32)-1073741824L;
    FWSpeedLoop.pPIpAWFW.f32UpperLimit = (tFrac32)0;
    FWSpeedLoop.pRamp.f32RampUp = (tFrac32)FRAC32(0.4768);
    FWSpeedLoop.pRamp.f32RampDown = (tFrac32)FRAC32(0.3754);
    FWSpeedLoop.pIQFbck = &f32IDQFbck.f32Arg2;
    FWSpeedLoop.pUQReq = &f32UDQReq.f32Arg2;
    FWSpeedLoop.pUQLim = &CurrentLoop.pPIrAWQ.f32UpperLimit;

    // Clear AMCLIB_FWSpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    AMCLIB_FWSpeedLoopInit_F32(&FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit(&FWSpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoopInit(&FWSpeedLoop);

    // Initialize the AMCLIB_FWSpeedLoop state variables to predefined values
    // Warning: Parameters in FWSpeedLoop must be already initialized.
    f32FilterMAWOut = (tFrac32)123L;
    f32FilterMAFWOut = (tFrac32)123L;
    f32ControllerPIpAWQOut = (tFrac32)123L;
    f32ControllerPIpAWFWOut = (tFrac32)123L;
    f32RampOut = (tFrac32)123L;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState_F32(f32FilterMAWOut, f32FilterMAFWOut,
      f32ControllerPIpAWQOut, f32ControllerPIpAWFWOut,
      f32RampOut, &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState(f32FilterMAWOut, f32FilterMAFWOut,
      f32ControllerPIpAWQOut, f32ControllerPIpAWFWOut,
      f32RampOut, &FWSpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoopSetState(f32FilterMAWOut, f32FilterMAFWOut,
      f32ControllerPIpAWQOut, f32ControllerPIpAWFWOut,
      f32RampOut, &FWSpeedLoop);

    f32VelocityReq = (tFrac32)100L;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoop_F32(f32VelocityReq, f32VelocityFbck, &IDQReq,
      &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoop(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &FWSpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoop(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &FWSpeedLoop);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of f32VelocityFbck, f32IDQFbck, f32UDQReq
    // (...)
}
```

2.9.4 Function AMCLIB_FWSpeedLoop_F16

2.9.4.1 Declaration

```
void AMCLIB_FWSpeedLoop_F16(tFrac16 f16VelocityReq,
tFrac16 f16VelocityFbck, SWLIBS_2Syst_F16 *const pIDQReq,
AMCLIB_FW_SPEED_LOOP_T_F16 *pCtrl);
```

2.9.4.2 Arguments

Table 62. AMCLIB_FWSpeedLoop_F16 arguments

Type	Name	Direction	Description
tFrac16	f16VelocityReq	**input**	Required electrical angular velocity (setpoint).
tFrac16	f16VelocityFbck	**input**	Actual electrical angular velocity from the feedback.
SWLIBS_2Syst_F16 *const	pIDQReq	**input, output**	Pointer to the structure with the required stator currents in the two-phase rotational orthogonal system (d-q).
AMCLIB_FW_SPEED_LOOP_T_F16 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FWSpeedLoop state.

2.9.4.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_FWSpeedLoop_F16.

Figure 64. Functions and data structures in AMCLIB_FWSpeedLoop_F16

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. All inputs and outputs of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

Table 63. Scaling constants

Scaling constant	Symbol	Calculation

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 63. Scaling constants...*continued*

Maximum stator phase voltage [V]	U_{MAX}	$U_{MAX}=U_{DC_Bus_Max}$
Maximum phase current [A]	I_{MAX}	Maximum current of the inverter or nominal current of the motor (whichever is lower).
Maximum speed [rad/s]	Ω_{MAX}	Maximum application required speed, at least the motor electrical rated speed.

The proportional gain K_P and integral gain K_I of the speed controller (GFLIB_ControllerPIpAW_F16) can be calculated as follows:

$$K_P = 2 \cdot \xi \cdot \omega_0 \cdot \frac{J}{K_T}$$

$$K_I = \omega_0{}^2 \cdot \frac{J}{K_T}$$

Equation AMCLIB_FWSpeedLoop_F16_Eq1

where ξ is the speed loop attenuation, ω_0 is the speed loop natural frequency [rad/s], J is the moment of inertia [kg·m^2], K_T is the motor torque constant [N·m·A^{-1}].

The continuous-time controller parameters K_P and K_I are then discretized using the bilinear transform. The resulting difference equation coefficients must be scaled so that the input and output values of the controller fit in the fractional range.

$$K_{P_SC} = K_P \cdot \frac{\Omega_{MAX}}{I_{MAX}}$$

$$K_{I_SC} = \frac{K_I T_S}{2} \cdot \frac{\Omega_{MAX}}{I_{MAX}}$$

Equation AMCLIB_FWSpeedLoop_F16_Eq2

T_S is the sampling period [s]. In case the coefficients K_{P_SC} and K_{I_SC} exceed the fractional range [-1, 1) or they are extremely small (which could lead to loss of precision), their value must be bit-shifted by *NShiftP* and *NShiftI*, respectively. The shift amount can be calculated as

$$NShiftP = \lceil \log_2(|K_{P_SC}|) \rceil$$

$$NShiftI = \lceil \log_2(|K_{I_SC}|) \rceil$$

Equation AMCLIB_FWSpeedLoop_F16_Eq3

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 15th power of two (corresponds to the Q1.15 encoding):

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$pPIpAWQ.f16PropGain = \left(K_{P_SC} \cdot 2^{-NShiftP}\right) \cdot 2^{15}$$
$$pPIpAWQ.f16IntegGain = \left(K_{I_SC} \cdot 2^{-NShiftI}\right) \cdot 2^{15}$$
$$pPIpAWQ.s16PropGainShift = NShiftP$$
$$pPIpAWQ.s16IntegGainShift = NShiftI$$
$$pPIpAWQ.f16UpperLimit = 2^{15} - 1$$
$$pPIpAWQ.f16LowerLimit = -2^{15}$$

Equation AMCLIB_FWSpeedLoop_F16_Eq4

The smoothing factor of the GDFLIB_FilterMA_F16 and the slope of the GFLIB_Ramp_F16 on the input of the speed controller should reflect the achievable dynamics of the drive. The AN4642 (Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM) can help with tuning of these parameters.

An initial estimate of the field weakening PI controller parameters can be taken from the speed PI controller as defined in the above AMCLIB_FWSpeedLoop_F16_Eq4. The upper limit of the field weakening controller output shall be set to zero, i.e. pPIpAWFW.f16UpperLimit = 0. The lower limit shall be set to an adequate value from the interval of fractional values [-0.5, 0] which corresponds to the Q1.15-encoded integer range [-16384, 0]. It is important to set the lower limit correctly to prevent motor damage by irreversible demagnetization of the permanent magnets. Let I_{D_MAX} be the maximum permitted negative d-axis current for field weakening. Then, to represent the lower limit in integer format expected by the C language, simply shift the radix point by multiplying the lower limit value by the 15th power of two (corresponds to the Q1.15 encoding):

$$pPIpAWFW.f16LowerLimit = \left(\frac{1}{\pi} \cdot \cos^{-1}\left(\frac{|I_{D_MAX}|}{I_{MAX}}\right) - 0.5\right) \cdot 2^{15}$$

Equation AMCLIB_FWSpeedLoop_F16_Eq5

Further refinement of the field weakening controller parameters should be done interactively based on the observed performance. This can be easily achieved with the FREEMASTER real-time debugging tool. The following points should be considered:

- The field weakening PI controller tweaks the angle of the current space vector I_S to produce the negative flux-producing current component I_D and sets the limits of the torque-producing current component I_Q.
- The magnitude of I_D depends also on the output of the speed PI controller.
- There are two different paths producing the PI input error. The weight of each depends on the motor operation mode.

The performance of the field weakening controller can be affected by measurement noise. The noise immunity can be improved by tweaking the smoothing factor of the moving average filter GDFLIB_FilterMA_F16 which preprocesses the input to the field weakening controller.

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

2.9.4.4 Code Example

```
#include "amclib.h"

AMCLIB_FW_SPEED_LOOP_T_F16 FWSpeedLoop;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
SWLIBS_2Syst_F16 IDQReq;       // required dq currents
SWLIBS_2Syst_F16 f16IDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_F16 f16UDQReq;    // required dq voltages
tFrac16 f16VelocityReq;        // required velocity
tFrac16 f16VelocityFbck;       // actual velocity
AMCLIB_CURRENT_LOOP_T_F16 CurrentLoop;

void main (void)
{
    tFrac16 f16FilterMAWOut;
    tFrac16 f16FilterMAFWOut;
    tFrac16 f16ControllerPIpAWQOut;
    tFrac16 f16ControllerPIpAWFWOut;
    tFrac32 f32RampOut;

    // Initialize the parameters and pointers in FWSpeedLoop
    FWSpeedLoop.pFilterW.u16NSamples = 3u;
    FWSpeedLoop.pFilterFW.u16NSamples = 3u;
    FWSpeedLoop.pPIpAWQ.f16PropGain = (tFrac16)FRAC16(0.1234);
    FWSpeedLoop.pPIpAWQ.f16IntegGain = (tFrac16)FRAC16(0.1675);
    FWSpeedLoop.pPIpAWQ.s16PropGainShift = 1;
    FWSpeedLoop.pPIpAWQ.s16IntegGainShift = 1;
    FWSpeedLoop.pPIpAWQ.f16LowerLimit = (tFrac16)-30621;
    FWSpeedLoop.pPIpAWQ.f16UpperLimit = (tFrac16)32066;
    FWSpeedLoop.pPIpAWFW.f16PropGain = (tFrac16)FRAC16(0.2348);
    FWSpeedLoop.pPIpAWFW.f16IntegGain = (tFrac16)FRAC16(0.3457);
    FWSpeedLoop.pPIpAWFW.s16PropGainShift = 1;
    FWSpeedLoop.pPIpAWFW.s16IntegGainShift = 1;
    FWSpeedLoop.pPIpAWFW.f16LowerLimit = (tFrac16)-16384;
    FWSpeedLoop.pPIpAWFW.f16UpperLimit = (tFrac16)0;
    FWSpeedLoop.pRamp.f32RampUp = (tFrac32)FRAC32(0.4768);
    FWSpeedLoop.pRamp.f32RampDown = (tFrac32)FRAC32(0.3754);
    FWSpeedLoop.pIQFbck = &f16IDQFbck.f16Arg2;
    FWSpeedLoop.pUQReq = &f16UDQReq.f16Arg2;
    FWSpeedLoop.pUQLim = &CurrentLoop.pPIrAWQ.f16UpperLimit;

    // Clear AMCLIB_FWSpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit_F16(&FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit(&FWSpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoopInit(&FWSpeedLoop);

    // Initialize the AMCLIB_FWSpeedLoop state variables to predefined values
    // Warning: Parameters in FWSpeedLoop must be already initialized.
    f16FilterMAWOut = (tFrac16)123;
    f16FilterMAFWOut = (tFrac16)123;
    f16ControllerPIpAWQOut = (tFrac16)123;
    f16ControllerPIpAWFWOut = (tFrac16)123;
    f32RampOut = (tFrac32)123;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState_F16(f16FilterMAWOut, f16FilterMAFWOut,
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
      f16ControllerPIpAWQOut, f16ControllerPIpAWFWOut,
      f32RampOut, &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState(f16FilterMAWOut, f16FilterMAFWOut,
      f16ControllerPIpAWQOut, f16ControllerPIpAWFWOut,
      f32RampOut, &FWSpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoopSetState(f16FilterMAWOut, f16FilterMAFWOut,
      f16ControllerPIpAWQOut, f16ControllerPIpAWFWOut,
      f32RampOut, &FWSpeedLoop);

    f16VelocityReq = (tFrac16)100;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoop_F16(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoop(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &FWSpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoop(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &FWSpeedLoop);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of f16VelocityFbck, f16IDQFbck, f16UDQReq
    // (...)
}
```

2.9.5 Function AMCLIB_FWSpeedLoop_FLT

2.9.5.1 Declaration

```
void AMCLIB_FWSpeedLoop_FLT(tFloat fltVelocityReq,
tFloat fltVelocityFbck, SWLIBS_2Syst_FLT *const pIDQReq,
AMCLIB_FW_SPEED_LOOP_T_FLT *pCtrl);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.9.5.2 Arguments

Table 64. AMCLIB_FWSpeedLoop_FLT arguments

Type	Name	Direction	Description
tFloat	fltVelocityReq	**input**	Required electrical angular velocity (setpoint).
tFloat	fltVelocityFbck	**input**	Actual electrical angular velocity from the feedback.
SWLIBS_2Syst_FLT *const	pIDQReq	**input, output**	Pointer to the structure with the required stator currents in the two-phase rotational orthogonal system (d-q).
AMCLIB_FW_SPEED_LOOP_T_FLT *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FWSpeedLoop state.

2.9.5.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_FWSpeedLoop_FLT.

Figure 65. Functions and data structures in AMCLIB_FWSpeedLoop_FLT

Parameters of the speed PI controller (GFLIB_ControllerPIpAW_FLT) can be calculated using the following equations:

$$pPIpAWQ.fltPropGain = 2 \cdot \xi \cdot \omega_0 \cdot \frac{J}{K_T}$$

$$pPIpAWQ.fltIntegGain = \omega_0^2 \cdot \frac{J}{K_T} \cdot \frac{T_S}{2}$$

$$pPIpAWQ.fltUpperLimit = I_{MAX}$$

$$pPIpAWQ.fltLowerLimit = -I_{MAX}$$

Equation AMCLIB_FWSpeedLoop_FLT_Eq1

where ξ is the speed loop attenuation, ω_0 is the speed loop natural frequency [rad/s], J is the moment of inertia [kg·m^2], K_T is the motor torque constant [N·m·A^{-1}], T_S is the sampling period [s], and I_{MAX} is the maximum phase current [A].

The smoothing factor of the GDFLIB_FilterMA_FLT and the slope of the GFLIB_Ramp_FLT on the input of the speed controller should reflect the achievable dynamics of the drive. The AN4642 (Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM) can help with tuning of these parameters.

An initial estimate of the field weakening PI controller parameters can be taken from the speed PI controller as defined in the above AMCLIB_FWSpeedLoop_FLT_Eq1. The upper limit of the field weakening controller output shall be set to zero, i.e. pPIpAWFW.fltUpperLimit = 0. The lower limit shall be set to an adequate value from the

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

interval [-π/2, 0]. It is important to set the lower limit correctly to prevent motor damage by irreversible demagnetization of the permanent magnets. Let I_{D_MAX} be the maximum permitted negative d-axis current for field weakening, then

$$pPIpAWFW.fltLowerLimit = \cos^{-1}\left(\frac{|I_{D_MAX}|}{I_{MAX}}\right) - \frac{\pi}{2}$$

Equation AMCLIB_FWSpeedLoop_FLT_Eq2

Further refinement of the field weakening controller parameters should be done interactively based on the observed performance. This can be easily achieved with the FREEMASTER real-time debugging tool. The following points should be considered:

- The field weakening PI controller tweaks the angle of the current space vector I_S to produce the negative flux-producing current component I_D and sets the limits of the torque-producing current component I_Q.
- The magnitude of I_D depends also on the output of the speed PI controller.
- There are two different paths producing the PI input error. The weight of each depends on the motor operation mode.

The performance of the field weakening controller can be affected by measurement noise. The noise immunity can be improved by tweaking the smoothing factor of the moving average filter GDFLIB_FilterMA_FLT which preprocesses the input to the field weakening controller.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.9.5.4 Code Example

```
#include "amclib.h"

AMCLIB_FW_SPEED_LOOP_T_FLT FWSpeedLoop;
SWLIBS_2Syst_FLT IDQReq;       // required dq currents
SWLIBS_2Syst_FLT fltIDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_FLT fltUDQReq;    // required dq voltages
tFloat fltVelocityReq;         // required velocity
tFloat fltVelocityFbck;        // actual velocity
AMCLIB_CURRENT_LOOP_T_FLT CurrentLoop;

void main (void)
{
    tFloat fltFilterMAWOut;
    tFloat fltFilterMAFWOut;
    tFloat fltControllerPIpAWQOut;
    tFloat fltControllerPIpAWFWOut;
    tFloat fltRampOut;

    // Initialize the parameters and pointers in FWSpeedLoop
    FWSpeedLoop.pFilterW.fltLambda = (tFloat)0.5;
    FWSpeedLoop.pFilterFW.fltLambda = (tFloat)0.6;
    FWSpeedLoop.pPIpAWQ.fltPropGain = (tFloat)0.1234;
    FWSpeedLoop.pPIpAWQ.fltIntegGain = (tFloat)0.1675;
    FWSpeedLoop.pPIpAWQ.fltLowerLimit = (tFloat)-0.934499979019165;
    FWSpeedLoop.pPIpAWQ.fltUpperLimit = (tFloat)0.978600025177002;
    FWSpeedLoop.pPIpAWFW.fltPropGain = (tFloat)0.2348;
    FWSpeedLoop.pPIpAWFW.fltIntegGain = (tFloat)0.3457;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    FWSpeedLoop.pPIpAWFW.fltLowerLimit = (tFloat)-1.570796326794897;
    FWSpeedLoop.pPIpAWFW.fltUpperLimit = (tFloat)0.0;
    FWSpeedLoop.pRamp.fltRampUp = (tFloat)0.4768;
    FWSpeedLoop.pRamp.fltRampDown = (tFloat)0.3754;
    FWSpeedLoop.pIQFbck = &fltIDQFbck.fltArg2;
    FWSpeedLoop.pUQReq = &fltUDQReq.fltArg2;
    FWSpeedLoop.pUQLim = &CurrentLoop.pPIrAWQ.fltUpperLimit;
    FWSpeedLoop.fltUmaxDivImax = (tFloat)1.0;

    // Clear AMCLIB_FWSpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit_FLT(&FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit(&FWSpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWSpeedLoopInit(&FWSpeedLoop);

    // Initialize the AMCLIB_FWSpeedLoop state variables to predefined values
    // Warning: Parameters in FWSpeedLoop must be already initialized.
    fltFilterMAWOut = 123.0F;
    fltFilterMAFWOut = 123.0F;
    fltControllerPIpAWQOut = 123.0F;
    fltControllerPIpAWFWOut = 123.0F;
    fltRampOut = 123.0F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState_FLT(fltFilterMAWOut, fltFilterMAFWOut,
      fltControllerPIpAWQOut, fltControllerPIpAWFWOut,
      fltRampOut, &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState(fltFilterMAWOut, fltFilterMAFWOut,
      fltControllerPIpAWQOut, fltControllerPIpAWFWOut,
      fltRampOut, &FWSpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWSpeedLoopSetState(fltFilterMAWOut, fltFilterMAFWOut,
      fltControllerPIpAWQOut, fltControllerPIpAWFWOut,
      fltRampOut, &FWSpeedLoop);

    fltVelocityReq = 100.0F;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    AMCLIB_FWSpeedLoop_FLT(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoop(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &FWSpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWSpeedLoop(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &FWSpeedLoop);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of fltVelocityFbck, fltIDQFbck, fltUDQReq
    // (...)
}
```

2.9.6 Function AMCLIB_FWSpeedLoopInit

2.9.6.1 Description

This function clears the AMCLIB_FWSpeedLoop state variables.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.9.6.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.9.6.3 Function AMCLIB_FWSpeedLoopInit_F32

2.9.6.3.1 Declaration

```
void AMCLIB_FWSpeedLoopInit_F32(AMCLIB_FW_SPEED_LOOP_T_F32 *const
pCtrl);
```

2.9.6.3.2 Arguments

Table 65. AMCLIB_FWSpeedLoopInit_F32 arguments

Type	Name	Direction	Description
AMCLIB_FW_SPEED_LOOP_T_F32 *const	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FWSpeedLoop state.

Note: *If pCtrl points to a structure of type AMCLIB_FW_SPEED_LOOP_DEBUG_T_F32, it must be recasted to AMCLIB_FW_SPEED_LOOP_T_F32 *.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.9.6.3.3 Code Example

```
#include "amclib.h"

AMCLIB_FW_SPEED_LOOP_T_F32 FWSpeedLoop;
SWLIBS_2Syst_F32 IDQReq;       // required dq currents
SWLIBS_2Syst_F32 f32IDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_F32 f32UDQReq;    // required dq voltages
tFrac32 f32VelocityReq;        // required velocity
tFrac32 f32VelocityFbck;       // actual velocity
AMCLIB_CURRENT_LOOP_T_F32 CurrentLoop;

void main (void)
{
    tFrac32 f32FilterMAWOut;
    tFrac32 f32FilterMAFWOut;
    tFrac32 f32ControllerPIpAWQOut;
    tFrac32 f32ControllerPIpAWFWOut;
    tFrac32 f32RampOut;

    // Initialize the parameters and pointers in FWSpeedLoop
    FWSpeedLoop.pFilterW.u16NSamples = 3u;
    FWSpeedLoop.pFilterFW.u16NSamples = 3u;
    FWSpeedLoop.pPIpAWQ.f32PropGain = (tFrac32)FRAC32(0.1234);
    FWSpeedLoop.pPIpAWQ.f32IntegGain = (tFrac32)FRAC32(0.1675);
    FWSpeedLoop.pPIpAWQ.s16PropGainShift = 1;
    FWSpeedLoop.pPIpAWQ.s16IntegGainShift = 1;
    FWSpeedLoop.pPIpAWQ.f32LowerLimit = (tFrac32)-2006823469L;
    FWSpeedLoop.pPIpAWQ.f32UpperLimit = (tFrac32)2101527497L;
    FWSpeedLoop.pPIpAWFW.f32PropGain = (tFrac32)FRAC32(0.2348);
    FWSpeedLoop.pPIpAWFW.f32IntegGain = (tFrac32)FRAC32(0.3457);
    FWSpeedLoop.pPIpAWFW.s16PropGainShift = 1;
    FWSpeedLoop.pPIpAWFW.s16IntegGainShift = 1;
    FWSpeedLoop.pPIpAWFW.f32LowerLimit = (tFrac32)-1073741824L;
    FWSpeedLoop.pPIpAWFW.f32UpperLimit = (tFrac32)0;
    FWSpeedLoop.pRamp.f32RampUp = (tFrac32)FRAC32(0.4768);
    FWSpeedLoop.pRamp.f32RampDown = (tFrac32)FRAC32(0.3754);
    FWSpeedLoop.pIQFbck = &f32IDQFbck.f32Arg2;
    FWSpeedLoop.pUQReq = &f32UDQReq.f32Arg2;
    FWSpeedLoop.pUQLim = &CurrentLoop.pPIrAWQ.f32UpperLimit;

    // Clear AMCLIB_FWSpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit_F32(&FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit(&FWSpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoopInit(&FWSpeedLoop);

    // Initialize the AMCLIB_FWSpeedLoop state variables to predefined values
    // Warning: Parameters in FWSpeedLoop must be already initialized.
    f32FilterMAWOut = (tFrac32)123L;
    f32FilterMAFWOut = (tFrac32)123L;
    f32ControllerPIpAWQOut = (tFrac32)123L;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    f32ControllerPIpAWFWOut = (tFrac32)123L;
    f32RampOut = (tFrac32)123L;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState_F32(f32FilterMAWOut, f32FilterMAFWOut,
      f32ControllerPIpAWQOut, f32ControllerPIpAWFWOut,
      f32RampOut, &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState(f32FilterMAWOut, f32FilterMAFWOut,
      f32ControllerPIpAWQOut, f32ControllerPIpAWFWOut,
      f32RampOut, &FWSpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoopSetState(f32FilterMAWOut, f32FilterMAFWOut,
      f32ControllerPIpAWQOut, f32ControllerPIpAWFWOut,
      f32RampOut, &FWSpeedLoop);

    f32VelocityReq = (tFrac32)100L;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoop_F32(f32VelocityReq, f32VelocityFbck, &IDQReq,
      &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoop(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &FWSpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoop(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &FWSpeedLoop);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of f32VelocityFbck, f32IDQFbck, f32UDQReq
    // (...)
}
```

2.9.6.4 Function AMCLIB_FWSpeedLoopInit_F16

2.9.6.4.1 Declaration

```
void AMCLIB_FWSpeedLoopInit_F16(AMCLIB_FW_SPEED_LOOP_T_F16 *const
pCtrl);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.9.6.4.2 Arguments

Table 66. AMCLIB_FWSpeedLoopInit_F16 arguments

Type	Name	Direction	Description
AMCLIB_FW_SPEED_LOOP_T_F16 *const	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FWSpeedLoop state.

Note: *If pCtrl points to a structure of type AMCLIB_FW_SPEED_LOOP_DEBUG_T_F16, it must be recasted to AMCLIB_FW_SPEED_LOOP_T_F16 *.*

2.9.6.4.3 Code Example

```
#include "amclib.h"

AMCLIB_FW_SPEED_LOOP_T_F16 FWSpeedLoop;
SWLIBS_2Syst_F16 IDQReq;       // required dq currents
SWLIBS_2Syst_F16 f16IDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_F16 f16UDQReq;    // required dq voltages
tFrac16 f16VelocityReq;        // required velocity
tFrac16 f16VelocityFbck;       // actual velocity
AMCLIB_CURRENT_LOOP_T_F16 CurrentLoop;

void main (void)
{
    tFrac16 f16FilterMAWOut;
    tFrac16 f16FilterMAFWOut;
    tFrac16 f16ControllerPIpAWQOut;
    tFrac16 f16ControllerPIpAWFWOut;
    tFrac32 f32RampOut;

    // Initialize the parameters and pointers in FWSpeedLoop
    FWSpeedLoop.pFilterW.u16NSamples = 3u;
    FWSpeedLoop.pFilterFW.u16NSamples = 3u;
    FWSpeedLoop.pPIpAWQ.f16PropGain = (tFrac16)FRAC16(0.1234);
    FWSpeedLoop.pPIpAWQ.f16IntegGain = (tFrac16)FRAC16(0.1675);
    FWSpeedLoop.pPIpAWQ.s16PropGainShift = 1;
    FWSpeedLoop.pPIpAWQ.s16IntegGainShift = 1;
    FWSpeedLoop.pPIpAWQ.f16LowerLimit = (tFrac16)-30621;
    FWSpeedLoop.pPIpAWQ.f16UpperLimit = (tFrac16)32066;
    FWSpeedLoop.pPIpAWFW.f16PropGain = (tFrac16)FRAC16(0.2348);
    FWSpeedLoop.pPIpAWFW.f16IntegGain = (tFrac16)FRAC16(0.3457);
    FWSpeedLoop.pPIpAWFW.s16PropGainShift = 1;
    FWSpeedLoop.pPIpAWFW.s16IntegGainShift = 1;
    FWSpeedLoop.pPIpAWFW.f16LowerLimit = (tFrac16)-16384;
    FWSpeedLoop.pPIpAWFW.f16UpperLimit = (tFrac16)0;
    FWSpeedLoop.pRamp.f32RampUp = (tFrac32)FRAC32(0.4768);
    FWSpeedLoop.pRamp.f32RampDown = (tFrac32)FRAC32(0.3754);
    FWSpeedLoop.pIQFbck = &f16IDQFbck.f16Arg2;
    FWSpeedLoop.pUQReq = &f16UDQReq.f16Arg2;
    FWSpeedLoop.pUQLim = &CurrentLoop.pPIrAWQ.f16UpperLimit;

    // Clear AMCLIB_FWSpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit_F16(&FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit(&FWSpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoopInit(&FWSpeedLoop);

    // Initialize the AMCLIB_FWSpeedLoop state variables to predefined values
    // Warning: Parameters in FWSpeedLoop must be already initialized.
    f16FilterMAWOut = (tFrac16)123;
    f16FilterMAFWOut = (tFrac16)123;
    f16ControllerPIpAWQOut = (tFrac16)123;
    f16ControllerPIpAWFWOut = (tFrac16)123;
    f32RampOut = (tFrac32)123;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState_F16(f16FilterMAWOut, f16FilterMAFWOut,
      f16ControllerPIpAWQOut, f16ControllerPIpAWFWOut,
      f32RampOut, &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState(f16FilterMAWOut, f16FilterMAFWOut,
      f16ControllerPIpAWQOut, f16ControllerPIpAWFWOut,
      f32RampOut, &FWSpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoopSetState(f16FilterMAWOut, f16FilterMAFWOut,
      f16ControllerPIpAWQOut, f16ControllerPIpAWFWOut,
      f32RampOut, &FWSpeedLoop);

    f16VelocityReq = (tFrac16)100;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoop_F16(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoop(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &FWSpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoop(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &FWSpeedLoop);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
{
    // Calculate new values of f16VelocityFbck, f16IDQFbck, f16UDQReq
    // (...)
}
```

2.9.6.5 Function AMCLIB_FWSpeedLoopInit_FLT

2.9.6.5.1 Declaration

```
void AMCLIB_FWSpeedLoopInit_FLT(AMCLIB_FW_SPEED_LOOP_T_FLT *const
pCtrl);
```

2.9.6.5.2 Arguments

Table 67. AMCLIB_FWSpeedLoopInit_FLT arguments

Type	Name	Direction	Description
AMCLIB_FW_SPEED_LOOP_T_FLT *const	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FWSpeedLoop state.

Note: *If pCtrl points to a structure of type AMCLIB_FW_SPEED_LOOP_DEBUG_T_FLT, it must be recasted to AMCLIB_FW_SPEED_LOOP_T_FLT *.*

The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.

2.9.6.5.3 Code Example

```
#include "amclib.h"

AMCLIB_FW_SPEED_LOOP_T_FLT FWSpeedLoop;
SWLIBS_2Syst_FLT IDQReq;       // required dq currents
SWLIBS_2Syst_FLT fltIDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_FLT fltUDQReq;    // required dq voltages
tFloat fltVelocityReq;         // required velocity
tFloat fltVelocityFbck;        // actual velocity
AMCLIB_CURRENT_LOOP_T_FLT CurrentLoop;

void main (void)
{
    tFloat fltFilterMAWOut;
    tFloat fltFilterMAFWOut;
    tFloat fltControllerPIpAWQOut;
    tFloat fltControllerPIpAWFWOut;
    tFloat fltRampOut;

    // Initialize the parameters and pointers in FWSpeedLoop
    FWSpeedLoop.pFilterW.fltLambda = (tFloat)0.5;
    FWSpeedLoop.pFilterFW.fltLambda = (tFloat)0.6;
    FWSpeedLoop.pPIpAWQ.fltPropGain = (tFloat)0.1234;
    FWSpeedLoop.pPIpAWQ.fltIntegGain = (tFloat)0.1675;
    FWSpeedLoop.pPIpAWQ.fltLowerLimit = (tFloat)-0.934499979019165;
    FWSpeedLoop.pPIpAWQ.fltUpperLimit = (tFloat)0.978600025177002;
    FWSpeedLoop.pPIpAWFW.fltPropGain = (tFloat)0.2348;
    FWSpeedLoop.pPIpAWFW.fltIntegGain = (tFloat)0.3457;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    FWSpeedLoop.pPIpAWFW.fltLowerLimit = (tFloat)-1.570796326794897;
    FWSpeedLoop.pPIpAWFW.fltUpperLimit = (tFloat)0.0;
    FWSpeedLoop.pRamp.fltRampUp = (tFloat)0.4768;
    FWSpeedLoop.pRamp.fltRampDown = (tFloat)0.3754;
    FWSpeedLoop.pIQFbck = &fltIDQFbck.fltArg2;
    FWSpeedLoop.pUQReq = &fltUDQReq.fltArg2;
    FWSpeedLoop.pUQLim = &CurrentLoop.pPIrAWQ.fltUpperLimit;
    FWSpeedLoop.fltUmaxDivImax = (tFloat)1.0;

    // Clear AMCLIB_FWSpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit_FLT(&FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit(&FWSpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWSpeedLoopInit(&FWSpeedLoop);

    // Initialize the AMCLIB_FWSpeedLoop state variables to predefined values
    // Warning: Parameters in FWSpeedLoop must be already initialized.
    fltFilterMAWOut = 123.0F;
    fltFilterMAFWOut = 123.0F;
    fltControllerPIpAWQOut = 123.0F;
    fltControllerPIpAWFWOut = 123.0F;
    fltRampOut = 123.0F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState_FLT(fltFilterMAWOut, fltFilterMAFWOut,
      fltControllerPIpAWQOut, fltControllerPIpAWFWOut,
      fltRampOut, &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState(fltFilterMAWOut, fltFilterMAFWOut,
      fltControllerPIpAWQOut, fltControllerPIpAWFWOut,
      fltRampOut, &FWSpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWSpeedLoopSetState(fltFilterMAWOut, fltFilterMAFWOut,
      fltControllerPIpAWQOut, fltControllerPIpAWFWOut,
      fltRampOut, &FWSpeedLoop);

    fltVelocityReq = 100.0F;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    AMCLIB_FWSpeedLoop_FLT(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoop(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &FWSpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWSpeedLoop(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &FWSpeedLoop);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of fltVelocityFbck, fltIDQFbck, fltUDQReq
    // (...)
}
```

2.9.7 Function AMCLIB_FWSpeedLoopSetState

2.9.7.1 Description

This function initializes the AMCLIB_FWSpeedLoop state variables to achieve the required output values.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.9.7.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.9.7.3 Function AMCLIB_FWSpeedLoopSetState_F32

2.9.7.3.1 Declaration

```
void AMCLIB_FWSpeedLoopSetState_F32(tFrac32 f32FilterMAWOut,
tFrac32 f32FilterMAFWOut, tFrac32 f32ControllerPIpAWQOut,
tFrac32 f32ControllerPIpAWFWOut, tFrac32 f32RampOut,
AMCLIB_FW_SPEED_LOOP_T_F32 *pCtrl);
```

2.9.7.3.2 Arguments

Table 68. AMCLIB_FWSpeedLoopSetState_F32 arguments

Type	Name	Direction	Description
tFrac32	f32FilterMAWOut	**input**	Required output of the speed FilterMA.
tFrac32	f32FilterMAFWOut	**input**	Required output of the field-weakening FilterMA.
tFrac32	f32ControllerPIpAWQOut	**input**	Required output of the speed ControllerPIpAW.
tFrac32	f32ControllerPIpAWFWOut	**input**	Required output of the field-weakening ControllerPIpAW.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 68. AMCLIB_FWSpeedLoopSetState_F32 arguments...*continued*

Type	Name	Direction	Description
tFrac32	f32RampOut	**input**	Required output of the speed ramp.
AMCLIB_FW_SPEED_LOOP_T_F32 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FWSpeedLoop state.

CAUTION: *Set the parameters in the structure pointed to by pCtrl before calling this function.*

Note: *If pCtrl points to a structure of type AMCLIB_FW_SPEED_LOOP_DEBUG_T_F32, it must be recasted to AMCLIB_FW_SPEED_LOOP_T_F32 *.*

2.9.7.3.3 Code Example

```
#include "amclib.h"

AMCLIB_FW_SPEED_LOOP_T_F32 FWSpeedLoop;
SWLIBS_2Syst_F32 IDQReq;       // required dq currents
SWLIBS_2Syst_F32 f32IDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_F32 f32UDQReq;    // required dq voltages
tFrac32 f32VelocityReq;        // required velocity
tFrac32 f32VelocityFbck;       // actual velocity
AMCLIB_CURRENT_LOOP_T_F32 CurrentLoop;

void main (void)
{
    tFrac32 f32FilterMAWOut;
    tFrac32 f32FilterMAFWOut;
    tFrac32 f32ControllerPIpAWQOut;
    tFrac32 f32ControllerPIpAWFWOut;
    tFrac32 f32RampOut;

    // Initialize the parameters and pointers in FWSpeedLoop
    FWSpeedLoop.pFilterW.u16NSamples = 3u;
    FWSpeedLoop.pFilterFW.u16NSamples = 3u;
    FWSpeedLoop.pPIpAWQ.f32PropGain = (tFrac32)FRAC32(0.1234);
    FWSpeedLoop.pPIpAWQ.f32IntegGain = (tFrac32)FRAC32(0.1675);
    FWSpeedLoop.pPIpAWQ.s16PropGainShift = 1;
    FWSpeedLoop.pPIpAWQ.s16IntegGainShift = 1;
    FWSpeedLoop.pPIpAWQ.f32LowerLimit = (tFrac32)-2006823469L;
    FWSpeedLoop.pPIpAWQ.f32UpperLimit = (tFrac32)2101527497L;
    FWSpeedLoop.pPIpAWFW.f32PropGain = (tFrac32)FRAC32(0.2348);
    FWSpeedLoop.pPIpAWFW.f32IntegGain = (tFrac32)FRAC32(0.3457);
    FWSpeedLoop.pPIpAWFW.s16PropGainShift = 1;
    FWSpeedLoop.pPIpAWFW.s16IntegGainShift = 1;
    FWSpeedLoop.pPIpAWFW.f32LowerLimit = (tFrac32)-1073741824L;
    FWSpeedLoop.pPIpAWFW.f32UpperLimit = (tFrac32)0;
    FWSpeedLoop.pRamp.f32RampUp = (tFrac32)FRAC32(0.4768);
    FWSpeedLoop.pRamp.f32RampDown = (tFrac32)FRAC32(0.3754);
    FWSpeedLoop.pIQFbck = &f32IDQFbck.f32Arg2;
    FWSpeedLoop.pUQReq = &f32UDQReq.f32Arg2;
    FWSpeedLoop.pUQLim = &CurrentLoop.pPIrAWQ.f32UpperLimit;

    // Clear AMCLIB_FWSpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit_F32(&FWSpeedLoop);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit(&FWSpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoopInit(&FWSpeedLoop);

    // Initialize the AMCLIB_FWSpeedLoop state variables to predefined values
    // Warning: Parameters in FWSpeedLoop must be already initialized.
    f32FilterMAWOut = (tFrac32)123L;
    f32FilterMAFWOut = (tFrac32)123L;
    f32ControllerPIpAWQOut = (tFrac32)123L;
    f32ControllerPIpAWFWOut = (tFrac32)123L;
    f32RampOut = (tFrac32)123L;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState_F32(f32FilterMAWOut, f32FilterMAFWOut,
      f32ControllerPIpAWQOut, f32ControllerPIpAWFWOut,
      f32RampOut, &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState(f32FilterMAWOut, f32FilterMAFWOut,
      f32ControllerPIpAWQOut, f32ControllerPIpAWFWOut,
      f32RampOut, &FWSpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoopSetState(f32FilterMAWOut, f32FilterMAFWOut,
      f32ControllerPIpAWQOut, f32ControllerPIpAWFWOut,
      f32RampOut, &FWSpeedLoop);

    f32VelocityReq = (tFrac32)100L;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoop_F32(f32VelocityReq, f32VelocityFbck, &IDQReq,
      &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoop(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &FWSpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoop(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &FWSpeedLoop);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of f32VelocityFbck, f32IDQFbck, f32UDQReq
    // (...)
}
```

2.9.7.4 Function AMCLIB_FWSpeedLoopSetState_F16

2.9.7.4.1 Declaration

```
void AMCLIB_FWSpeedLoopSetState_F16(tFrac16 f16FilterMAWOut,
tFrac16 f16FilterMAFWOut, tFrac16 f16ControllerPIpAWQOut,
tFrac16 f16ControllerPIpAWFWOut, tFrac32 f32RampOut,
AMCLIB_FW_SPEED_LOOP_T_F16 *pCtrl);
```

2.9.7.4.2 Arguments

Table 69. AMCLIB_FWSpeedLoopSetState_F16 arguments

Type	Name	Direction	Description
tFrac16	f16FilterMAWOut	**input**	Required output of the speed FilterMA.
tFrac16	f16FilterMAFWOut	**input**	Required output of the field-weakening FilterMA.
tFrac16	f16ControllerPIpAWQOut	**input**	Required output of the speed ControllerPIpAW.
tFrac16	f16ControllerPIpAWFWOut	**input**	Required output of the field-weakening ControllerPIpAW.
tFrac32	f32RampOut	**input**	Required output of the speed ramp.
AMCLIB_FW_SPEED_LOOP_T_F16 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FWSpeedLoop state.

CAUTION: *Set the parameters in the structure pointed to by pCtrl before calling this function.*

Note: *If pCtrl points to a structure of type* AMCLIB_FW_SPEED_LOOP_DEBUG_T_F16*, it must be recasted to* AMCLIB_FW_SPEED_LOOP_T_F16 *.*

2.9.7.4.3 Code Example

```
#include "amclib.h"

AMCLIB_FW_SPEED_LOOP_T_F16 FWSpeedLoop;
SWLIBS_2Syst_F16 IDQReq;       // required dq currents
SWLIBS_2Syst_F16 f16IDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_F16 f16UDQReq;    // required dq voltages
tFrac16 f16VelocityReq;        // required velocity
tFrac16 f16VelocityFbck;       // actual velocity
AMCLIB_CURRENT_LOOP_T_F16 CurrentLoop;

void main (void)
{
    tFrac16 f16FilterMAWOut;
    tFrac16 f16FilterMAFWOut;
    tFrac16 f16ControllerPIpAWQOut;
    tFrac16 f16ControllerPIpAWFWOut;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    tFrac32 f32RampOut;

    // Initialize the parameters and pointers in FWSpeedLoop
    FWSpeedLoop.pFilterW.u16NSamples = 3u;
    FWSpeedLoop.pFilterFW.u16NSamples = 3u;
    FWSpeedLoop.pPIpAWQ.f16PropGain = (tFrac16)FRAC16(0.1234);
    FWSpeedLoop.pPIpAWQ.f16IntegGain = (tFrac16)FRAC16(0.1675);
    FWSpeedLoop.pPIpAWQ.s16PropGainShift = 1;
    FWSpeedLoop.pPIpAWQ.s16IntegGainShift = 1;
    FWSpeedLoop.pPIpAWQ.f16LowerLimit = (tFrac16)-30621;
    FWSpeedLoop.pPIpAWQ.f16UpperLimit = (tFrac16)32066;
    FWSpeedLoop.pPIpAWFW.f16PropGain = (tFrac16)FRAC16(0.2348);
    FWSpeedLoop.pPIpAWFW.f16IntegGain = (tFrac16)FRAC16(0.3457);
    FWSpeedLoop.pPIpAWFW.s16PropGainShift = 1;
    FWSpeedLoop.pPIpAWFW.s16IntegGainShift = 1;
    FWSpeedLoop.pPIpAWFW.f16LowerLimit = (tFrac16)-16384;
    FWSpeedLoop.pPIpAWFW.f16UpperLimit = (tFrac16)0;
    FWSpeedLoop.pRamp.f32RampUp = (tFrac32)FRAC32(0.4768);
    FWSpeedLoop.pRamp.f32RampDown = (tFrac32)FRAC32(0.3754);
    FWSpeedLoop.pIQFbck = &f16IDQFbck.f16Arg2;
    FWSpeedLoop.pUQReq = &f16UDQReq.f16Arg2;
    FWSpeedLoop.pUQLim = &CurrentLoop.pPIrAWQ.f16UpperLimit;

    // Clear AMCLIB_FWSpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit_F16(&FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit(&FWSpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoopInit(&FWSpeedLoop);

    // Initialize the AMCLIB_FWSpeedLoop state variables to predefined values
    // Warning: Parameters in FWSpeedLoop must be already initialized.
    f16FilterMAWOut = (tFrac16)123;
    f16FilterMAFWOut = (tFrac16)123;
    f16ControllerPIpAWQOut = (tFrac16)123;
    f16ControllerPIpAWFWOut = (tFrac16)123;
    f32RampOut = (tFrac32)123;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState_F16(f16FilterMAWOut, f16FilterMAFWOut,
      f16ControllerPIpAWQOut, f16ControllerPIpAWFWOut,
      f32RampOut, &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState(f16FilterMAWOut, f16FilterMAFWOut,
      f16ControllerPIpAWQOut, f16ControllerPIpAWFWOut,
      f32RampOut, &FWSpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoopSetState(f16FilterMAWOut, f16FilterMAFWOut,
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
      f16ControllerPIpAWQOut, f16ControllerPIpAWFWOut,
      f32RampOut, &FWSpeedLoop);

    f16VelocityReq = (tFrac16)100;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoop_F16(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoop(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &FWSpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoop(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &FWSpeedLoop);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of f16VelocityFbck, f16IDQFbck, f16UDQReq
    // (...)
}
```

2.9.7.5 Function AMCLIB_FWSpeedLoopSetState_FLT

2.9.7.5.1 Declaration

```
void AMCLIB_FWSpeedLoopSetState_FLT(tFloat fltFilterMAWOut,
tFloat fltFilterMAFWOut, tFloat fltControllerPIpAWQOut,
tFloat fltControllerPIpAWFWOut, tFloat fltRampOut,
AMCLIB_FW_SPEED_LOOP_T_FLT *pCtrl);
```

2.9.7.5.2 Arguments

Table 70. AMCLIB_FWSpeedLoopSetState_FLT arguments

Type	Name	Direction	Description
tFloat	fltFilterMAWOut	**input**	Required output of the speed FilterMA.
tFloat	fltFilterMAFWOut	**input**	Required output of the field-weakening FilterMA.
tFloat	fltControllerPIpAWQOut	**input**	Required output of the speed ControllerPIpAW.
tFloat	fltControllerPIpAWFWOut	**input**	Required output of the field-weakening ControllerPIpAW.
tFloat	fltRampOut	**input**	Required output of the speed ramp.
AMCLIB_FW_SPEED_LOOP_T_FLT *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FWSpeedLoop state.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

CAUTION: *Set the parameters in the structure pointed to by pCtrl before calling this function.*

Note: *If pCtrl points to a structure of type* AMCLIB_FW_SPEED_LOOP_DEBUG_T_FLT*, it must be recasted to* AMCLIB_FW_SPEED_LOOP_T_FLT *.*

The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.

2.9.7.5.3 Code Example

```
#include "amclib.h"

AMCLIB_FW_SPEED_LOOP_T_FLT FWSpeedLoop;
SWLIBS_2Syst_FLT IDQReq;       // required dq currents
SWLIBS_2Syst_FLT fltIDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_FLT fltUDQReq;    // required dq voltages
tFloat fltVelocityReq;         // required velocity
tFloat fltVelocityFbck;        // actual velocity
AMCLIB_CURRENT_LOOP_T_FLT CurrentLoop;

void main (void)
{
    tFloat fltFilterMAWOut;
    tFloat fltFilterMAFWOut;
    tFloat fltControllerPIpAWQOut;
    tFloat fltControllerPIpAWFWOut;
    tFloat fltRampOut;

    // Initialize the parameters and pointers in FWSpeedLoop
    FWSpeedLoop.pFilterW.fltLambda = (tFloat)0.5;
    FWSpeedLoop.pFilterFW.fltLambda = (tFloat)0.6;
    FWSpeedLoop.pPIpAWQ.fltPropGain = (tFloat)0.1234;
    FWSpeedLoop.pPIpAWQ.fltIntegGain = (tFloat)0.1675;
    FWSpeedLoop.pPIpAWQ.fltLowerLimit = (tFloat)-0.934499979019165;
    FWSpeedLoop.pPIpAWQ.fltUpperLimit = (tFloat)0.978600025177002;
    FWSpeedLoop.pPIpAWFW.fltPropGain = (tFloat)0.2348;
    FWSpeedLoop.pPIpAWFW.fltIntegGain = (tFloat)0.3457;
    FWSpeedLoop.pPIpAWFW.fltLowerLimit = (tFloat)-1.570796326794897;
    FWSpeedLoop.pPIpAWFW.fltUpperLimit = (tFloat)0.0;
    FWSpeedLoop.pRamp.fltRampUp = (tFloat)0.4768;
    FWSpeedLoop.pRamp.fltRampDown = (tFloat)0.3754;
    FWSpeedLoop.pIQFbck = &fltIDQFbck.fltArg2;
    FWSpeedLoop.pUQReq = &fltUDQReq.fltArg2;
    FWSpeedLoop.pUQLim = &CurrentLoop.pPIrAWQ.fltUpperLimit;
    FWSpeedLoop.fltUmaxDivImax = (tFloat)1.0;

    // Clear AMCLIB_FWSpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit_FLT(&FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit(&FWSpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWSpeedLoopInit(&FWSpeedLoop);

    // Initialize the AMCLIB_FWSpeedLoop state variables to predefined values
    // Warning: Parameters in FWSpeedLoop must be already initialized.
    fltFilterMAWOut = 123.0F;
    fltFilterMAFWOut = 123.0F;
    fltControllerPIpAWQOut = 123.0F;
    fltControllerPIpAWFWOut = 123.0F;
    fltRampOut = 123.0F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState_FLT(fltFilterMAWOut, fltFilterMAFWOut,
      fltControllerPIpAWQOut, fltControllerPIpAWFWOut,
      fltRampOut, &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState(fltFilterMAWOut, fltFilterMAFWOut,
      fltControllerPIpAWQOut, fltControllerPIpAWFWOut,
      fltRampOut, &FWSpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWSpeedLoopSetState(fltFilterMAWOut, fltFilterMAFWOut,
      fltControllerPIpAWQOut, fltControllerPIpAWFWOut,
      fltRampOut, &FWSpeedLoop);

    fltVelocityReq = 100.0F;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoop_FLT(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoop(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &FWSpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWSpeedLoop(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &FWSpeedLoop);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Calculate new values of fltVelocityFbck, fltIDQFbck, fltUDQReq
    // (...)
}
```

2.9.8 Function AMCLIB_FWSpeedLoopDebug

This function adjusts the torque of the motor to achieve the required speed. The function employs the PMSM Field Weakening technique to extend the available speed range. Debugging information is provided.

2.9.8.1 Description

This library function implements a portion of Field Oriented Control (FOC) algorithm. FOC (also called vector control) is a widely used control strategy for Permanent Magnet Synchronous Motors (PMSM). FOC is characterized by smooth rotation over the entire speed range of the motor, full torque control at zero speed, and fast acceleration/deceleration. FOC consists of a hierarchical cascade of inner current loop and outer speed loop. PI controllers within these closed loops maintain the required speed and torque based on feedback measurements.

AMCLIB_FWSpeedLoopDebug implements the speed PI controller and the field weakening controller in the outer control loop highlighted in Figure 66 . AMCLIB_FWSpeedLoopDebug combines the functionalities of AMCLIB_FWDebug and AMCLIB_SpeedLoopDebug in a more integrated form to simplify the application code and improve execution speed. AMCLIB_FWSpeedLoopDebug provides the same functionality as AMCLIB_FWSpeedLoop. Additionally, this function allows debugging of all internal variables. The debugging outputs are provided in the structure pointed to by `pCtrl`. Replace AMCLIB_FWSpeedLoopDebug by AMCLIB_FWSpeedLoop once the debugging is finished.

Figure 66. Principal Field Oriented Control scheme with AMCLIB_FWSpeedLoop

Field weakening technique employed in AMCLIB_FWSpeedLoopDebug extends the speed range of PMSM beyond the motor base speed by reducing the linkage magnetic flux through negative current id_req. See AMCLIB_FW for more details.

2.9.8.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.9.8.3 Function AMCLIB_FWSpeedLoopDebug_F32

2.9.8.3.1 Declaration

```
void AMCLIB_FWSpeedLoopDebug_F32(tFrac32 f32VelocityReq,
tFrac32 f32VelocityFbck, SWLIBS_2Syst_F32 *const pIDQReq,
AMCLIB_FW_SPEED_LOOP_DEBUG_T_F32 *pCtrl);
```

2.9.8.3.2 Arguments

Table 71. AMCLIB_FWSpeedLoopDebug_F32 arguments

Type	Name	Direction	Description
tFrac32	f32VelocityReq	**input**	Required electrical angular velocity (setpoint).
tFrac32	f32VelocityFbck	**input**	Actual electrical angular velocity from the feedback.
SWLIBS_2Syst_F32 *const	pIDQReq	**input, output**	Pointer to the structure with the required stator currents in the two-phase rotational orthogonal system (d-q).
AMCLIB_FW_SPEED_LOOP_DEBUG_T_F32 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FWSpeedLoop state and debugging information.

2.9.8.3.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_FWSpeedLoopDebug_F32.

Figure 67. Functions and data structures in AMCLIB_FWSpeedLoopDebug_F32

Refer to the description of AMCLIB_FWSpeedLoop_F32 function on how to set up the controller parameters.

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

2.9.8.3.4 Code Example

```
#include "amclib.h"

AMCLIB_FW_SPEED_LOOP_DEBUG_T_F32 FWSpeedLoop;
SWLIBS_2Syst_F32 IDQReq;        // required dq currents
SWLIBS_2Syst_F32 f32IDQFbck;    // calculated dq currents from the feedback
SWLIBS_2Syst_F32 f32UDQReq;     // required dq voltages
tFrac32 f32VelocityReq;         // required velocity
tFrac32 f32VelocityFbck;        // actual velocity
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
AMCLIB_CURRENT_LOOP_T_F32 CurrentLoop;

void main (void)
{
    tFrac32 f32FilterMAWOut;
    tFrac32 f32FilterMAFWOut;
    tFrac32 f32ControllerPIpAWQOut;
    tFrac32 f32ControllerPIpAWFWOut;
    tFrac32 f32RampOut;

    // Initialize the parameters and pointers in FWSpeedLoop
    FWSpeedLoop.pFilterW.u16NSamples = 3u;
    FWSpeedLoop.pFilterFW.u16NSamples = 3u;
    FWSpeedLoop.pPIpAWQ.f32PropGain = (tFrac32)FRAC32(0.1234);
    FWSpeedLoop.pPIpAWQ.f32IntegGain = (tFrac32)FRAC32(0.1675);
    FWSpeedLoop.pPIpAWQ.s16PropGainShift = 1;
    FWSpeedLoop.pPIpAWQ.s16IntegGainShift = 1;
    FWSpeedLoop.pPIpAWQ.f32LowerLimit = (tFrac32)-2006823469L;
    FWSpeedLoop.pPIpAWQ.f32UpperLimit = (tFrac32)2101527497L;
    FWSpeedLoop.pPIpAWFW.f32PropGain = (tFrac32)FRAC32(0.2348);
    FWSpeedLoop.pPIpAWFW.f32IntegGain = (tFrac32)FRAC32(0.3457);
    FWSpeedLoop.pPIpAWFW.s16PropGainShift = 1;
    FWSpeedLoop.pPIpAWFW.s16IntegGainShift = 1;
    FWSpeedLoop.pPIpAWFW.f32LowerLimit = (tFrac32)-1073741824L;
    FWSpeedLoop.pPIpAWFW.f32UpperLimit = (tFrac32)0;
    FWSpeedLoop.pRamp.f32RampUp = (tFrac32)FRAC32(0.4768);
    FWSpeedLoop.pRamp.f32RampDown = (tFrac32)FRAC32(0.3754);
    FWSpeedLoop.pIQFbck = &f32IDQFbck.f32Arg2;
    FWSpeedLoop.pUQReq = &f32UDQReq.f32Arg2;
    FWSpeedLoop.pUQLim = &CurrentLoop.pPIrAWQ.f32UpperLimit;

    // Clear AMCLIB_FWSpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit_F32((AMCLIB_FW_SPEED_LOOP_T_F32 *)&FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit((AMCLIB_FW_SPEED_LOOP_T_F32 *)&FWSpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoopInit((AMCLIB_FW_SPEED_LOOP_T_F32 *)&FWSpeedLoop);

    // Initialize the AMCLIB_FWSpeedLoop state variables to predefined values
    // Warning: Parameters in FWSpeedLoop must be already initialized.
    f32FilterMAWOut = (tFrac32)123L;
    f32FilterMAFWOut = (tFrac32)123L;
    f32ControllerPIpAWQOut = (tFrac32)123L;
    f32ControllerPIpAWFWOut = (tFrac32)123L;
    f32RampOut = (tFrac32)123L;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState_F32(f32FilterMAWOut, f32FilterMAFWOut,
      f32ControllerPIpAWQOut, f32ControllerPIpAWFWOut,
      f32RampOut, (AMCLIB_FW_SPEED_LOOP_T_F32 *)&FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    AMCLIB_FWSpeedLoopSetState(f32FilterMAWOut, f32FilterMAFWOut,
      f32ControllerPIpAWQOut, f32ControllerPIpAWFWOut,
      f32RampOut, (AMCLIB_FW_SPEED_LOOP_T_F32 *)&FWSpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoopSetState(f32FilterMAWOut, f32FilterMAFWOut,
      f32ControllerPIpAWQOut, f32ControllerPIpAWFWOut,
      f32RampOut, (AMCLIB_FW_SPEED_LOOP_T_F32 *)&FWSpeedLoop);

    f32VelocityReq = (tFrac32)100L;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopDebug_F32(f32VelocityReq, f32VelocityFbck, &IDQReq,
      &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopDebug(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &FWSpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoopDebug(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &FWSpeedLoop);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of f32VelocityFbck, f32IDQFbck, f32UDQReq
    // (...)
}
```

2.9.8.4 Function AMCLIB_FWSpeedLoopDebug_F16

2.9.8.4.1 Declaration

```
void AMCLIB_FWSpeedLoopDebug_F16(tFrac16 f16VelocityReq,
tFrac16 f16VelocityFbck, SWLIBS_2Syst_F16 *const pIDQReq,
AMCLIB_FW_SPEED_LOOP_DEBUG_T_F16 *pCtrl);
```

2.9.8.4.2 Arguments

Table 72. AMCLIB_FWSpeedLoopDebug_F16 arguments

Type	Name	Direction	Description
tFrac16	f16VelocityReq	**input**	Required electrical angular velocity (setpoint).
tFrac16	f16VelocityFbck	**input**	Actual electrical angular velocity from the feedback.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 72. AMCLIB_FWSpeedLoopDebug_F16 arguments...*continued*

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *const	pIDQReq	**input, output**	Pointer to the structure with the required stator currents in the two-phase rotational orthogonal system (d-q).
AMCLIB_FW_SPEED_LOOP_DEBUG_T_F16 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FWSpeedLoop state and debugging information.

2.9.8.4.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_FWSpeedLoopDebug_F16.

Figure 68. Functions and data structures in AMCLIB_FWSpeedLoopDebug_F16

Refer to the description of AMCLIB_FWSpeedLoop_F16 function on how to set up the controller parameters.

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

2.9.8.4.4 Code Example

```
#include "amclib.h"

AMCLIB_FW_SPEED_LOOP_DEBUG_T_F16 FWSpeedLoop;
SWLIBS_2Syst_F16 IDQReq;       // required dq currents
SWLIBS_2Syst_F16 f16IDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_F16 f16UDQReq;    // required dq voltages
tFrac16 f16VelocityReq;        // required velocity
tFrac16 f16VelocityFbck;       // actual velocity
AMCLIB_CURRENT_LOOP_T_F16 CurrentLoop;

void main (void)
{
    tFrac16 f16FilterMAWOut;
    tFrac16 f16FilterMAFWOut;
    tFrac16 f16ControllerPIpAWQOut;
    tFrac16 f16ControllerPIpAWFWOut;
    tFrac32 f32RampOut;

    // Initialize the parameters and pointers in FWSpeedLoop
    FWSpeedLoop.pFilterW.u16NSamples = 3u;
    FWSpeedLoop.pFilterFW.u16NSamples = 3u;
    FWSpeedLoop.pPIpAWQ.f16PropGain = (tFrac16)FRAC16(0.1234);
    FWSpeedLoop.pPIpAWQ.f16IntegGain = (tFrac16)FRAC16(0.1675);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    FWSpeedLoop.pPIpAWQ.s16PropGainShift = 1;
    FWSpeedLoop.pPIpAWQ.s16IntegGainShift = 1;
    FWSpeedLoop.pPIpAWQ.f16LowerLimit = (tFrac16)-30621;
    FWSpeedLoop.pPIpAWQ.f16UpperLimit = (tFrac16)32066;
    FWSpeedLoop.pPIpAWFW.f16PropGain = (tFrac16)FRAC16(0.2348);
    FWSpeedLoop.pPIpAWFW.f16IntegGain = (tFrac16)FRAC16(0.3457);
    FWSpeedLoop.pPIpAWFW.s16PropGainShift = 1;
    FWSpeedLoop.pPIpAWFW.s16IntegGainShift = 1;
    FWSpeedLoop.pPIpAWFW.f16LowerLimit = (tFrac16)-16384;
    FWSpeedLoop.pPIpAWFW.f16UpperLimit = (tFrac16)0;
    FWSpeedLoop.pRamp.f32RampUp = (tFrac32)FRAC32(0.4768);
    FWSpeedLoop.pRamp.f32RampDown = (tFrac32)FRAC32(0.3754);
    FWSpeedLoop.pIQFbck = &f16IDQFbck.f16Arg2;
    FWSpeedLoop.pUQReq = &f16UDQReq.f16Arg2;
    FWSpeedLoop.pUQLim = &CurrentLoop.pPIrAWQ.f16UpperLimit;

    // Clear AMCLIB_FWSpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit_F16((AMCLIB_FW_SPEED_LOOP_T_F16 *)&FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit((AMCLIB_FW_SPEED_LOOP_T_F16 *)&FWSpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoopInit((AMCLIB_FW_SPEED_LOOP_T_F16 *)&FWSpeedLoop);

    // Initialize the AMCLIB_FWSpeedLoop state variables to predefined values
    // Warning: Parameters in FWSpeedLoop must be already initialized.
    f16FilterMAWOut = (tFrac16)123;
    f16FilterMAFWOut = (tFrac16)123;
    f16ControllerPIpAWQOut = (tFrac16)123;
    f16ControllerPIpAWFWOut = (tFrac16)123;
    f32RampOut = (tFrac32)123;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState_F16(f16FilterMAWOut, f16FilterMAFWOut,
      f16ControllerPIpAWQOut, f16ControllerPIpAWFWOut,
      f32RampOut, (AMCLIB_FW_SPEED_LOOP_T_F16 *)&FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState(f16FilterMAWOut, f16FilterMAFWOut,
      f16ControllerPIpAWQOut, f16ControllerPIpAWFWOut,
      f32RampOut, (AMCLIB_FW_SPEED_LOOP_T_F16 *)&FWSpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoopSetState(f16FilterMAWOut, f16FilterMAFWOut,
      f16ControllerPIpAWQOut, f16ControllerPIpAWFWOut,
      f32RampOut, (AMCLIB_FW_SPEED_LOOP_T_F16 *)&FWSpeedLoop);

    f16VelocityReq = (tFrac16)100;
    while(1);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopDebug_F16(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopDebug(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &FWSpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_FWSpeedLoopDebug(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &FWSpeedLoop);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of f16VelocityFbck, f16IDQFbck, f16UDQReq
    // (...)
}
```

2.9.8.5 Function AMCLIB_FWSpeedLoopDebug_FLT

2.9.8.5.1 Declaration

```
void AMCLIB_FWSpeedLoopDebug_FLT(tFloat fltVelocityReq,
tFloat fltVelocityFbck, SWLIBS_2Syst_FLT *const pIDQReq,
AMCLIB_FW_SPEED_LOOP_DEBUG_T_FLT *pCtrl);
```

2.9.8.5.2 Arguments

Table 73. AMCLIB_FWSpeedLoopDebug_FLT arguments

Type	Name	Direction	Description
tFloat	fltVelocityReq	**input**	Required electrical angular velocity (setpoint).
tFloat	fltVelocityFbck	**input**	Actual electrical angular velocity from the feedback.
SWLIBS_2Syst_FLT *const	pIDQReq	**input, output**	Pointer to the structure with the required stator currents in the two-phase rotational orthogonal system (d-q).
AMCLIB_FW_SPEED_LOOP_DEBUG_T_FLT *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_FWSpeedLoop state and debugging information.

2.9.8.5.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_FWSpeedLoopDebug_FLT.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 69. Functions and data structures in AMCLIB_FWSpeedLoopDebug_FLT

Refer to the description of AMCLIB_FWSpeedLoop_FLT function on how to set up the controller parameters.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.9.8.5.4 Code Example

```
#include "amclib.h"

AMCLIB_FW_SPEED_LOOP_DEBUG_T_FLT FWSpeedLoop;
SWLIBS_2Syst_FLT IDQReq;       // required dq currents
SWLIBS_2Syst_FLT fltIDQFbck;   // calculated dq currents from the feedback
SWLIBS_2Syst_FLT fltUDQReq;    // required dq voltages
tFloat fltVelocityReq;         // required velocity
tFloat fltVelocityFbck;        // actual velocity
AMCLIB_CURRENT_LOOP_T_FLT CurrentLoop;

void main (void)
{
    tFloat fltFilterMAWOut;
    tFloat fltFilterMAFWOut;
    tFloat fltControllerPIpAWQOut;
    tFloat fltControllerPIpAWFWOut;
    tFloat fltRampOut;

    // Initialize the parameters and pointers in FWSpeedLoop
    FWSpeedLoop.pFilterW.fltLambda = (tFloat)0.5;
    FWSpeedLoop.pFilterFW.fltLambda = (tFloat)0.6;
    FWSpeedLoop.pPIpAWQ.fltPropGain = (tFloat)0.1234;
    FWSpeedLoop.pPIpAWQ.fltIntegGain = (tFloat)0.1675;
    FWSpeedLoop.pPIpAWQ.fltLowerLimit = (tFloat)-0.934499979019165;
    FWSpeedLoop.pPIpAWQ.fltUpperLimit = (tFloat)0.978600025177002;
    FWSpeedLoop.pPIpAWFW.fltPropGain = (tFloat)0.2348;
    FWSpeedLoop.pPIpAWFW.fltIntegGain = (tFloat)0.3457;
    FWSpeedLoop.pPIpAWFW.fltLowerLimit = (tFloat)-1.570796326794897;
    FWSpeedLoop.pPIpAWFW.fltUpperLimit = (tFloat)0.0;
    FWSpeedLoop.pRamp.fltRampUp = (tFloat)0.4768;
    FWSpeedLoop.pRamp.fltRampDown = (tFloat)0.3754;
    FWSpeedLoop.pIQFbck = &fltIDQFbck.fltArg2;
    FWSpeedLoop.pUQReq = &fltUDQReq.fltArg2;
    FWSpeedLoop.pUQLim = &CurrentLoop.pPIrAWQ.fltUpperLimit;
    FWSpeedLoop.fltUmaxDivImax = (tFloat)1.0;

    // Clear AMCLIB_FWSpeedLoop state variables
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit_FLT((AMCLIB_FW_SPEED_LOOP_T_FLT *)&FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopInit((AMCLIB_FW_SPEED_LOOP_T_FLT *)&FWSpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWSpeedLoopInit((AMCLIB_FW_SPEED_LOOP_T_FLT *)&FWSpeedLoop);

    // Initialize the AMCLIB_FWSpeedLoop state variables to predefined values
    // Warning: Parameters in FWSpeedLoop must be already initialized.
    fltFilterMAWOut = 123.0F;
    fltFilterMAFWOut = 123.0F;
    fltControllerPIpAWQOut = 123.0F;
    fltControllerPIpAWFWOut = 123.0F;
    fltRampOut = 123.0F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState_FLT(fltFilterMAWOut, fltFilterMAFWOut,
      fltControllerPIpAWQOut, fltControllerPIpAWFWOut,
      fltRampOut, (AMCLIB_FW_SPEED_LOOP_T_FLT *)&FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopSetState(fltFilterMAWOut, fltFilterMAFWOut,
      fltControllerPIpAWQOut, fltControllerPIpAWFWOut,
      fltRampOut, (AMCLIB_FW_SPEED_LOOP_T_FLT *)&FWSpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWSpeedLoopSetState(fltFilterMAWOut, fltFilterMAFWOut,
      fltControllerPIpAWQOut, fltControllerPIpAWFWOut,
      fltRampOut, (AMCLIB_FW_SPEED_LOOP_T_FLT *)&FWSpeedLoop);

    fltVelocityReq = 100.0F;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopDebug_FLT(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &FWSpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_FWSpeedLoopDebug(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &FWSpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_FWSpeedLoopDebug(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &FWSpeedLoop);
}

// Periodical function or interrupt - current control loop
void FastLoop(void)
{
    // Calculate new values of fltVelocityFbck, fltIDQFbck, fltUDQReq
    // (...)
}
```

2.10 Function AMCLIB_SpeedLoop

This function implements the PI controller in the speed FOC outer control loop.

2.10.1 Description

This library function implements the speed control loop which is the outer loop in the cascade control structure of the speed FOC. FOC (also called vector control) is a widely used control strategy for Permanent Magnet Synchronous Motors (PMSM). FOC is characterized by smooth rotation over the entire speed range of the motor, full torque control at zero speed, and fast acceleration/deceleration. A PI controller in speed closed-loop system maintains the required speed by setting the torque producing current in the q-axis.

AMCLIB_SpeedLoop implements the speed controller in the outer control loop highlighted in Figure 70 . Use AMCLIB_FWSpeedLoop instead to take advantage of the field weakening technique when the application requires motor speeds beyond the nominal range. AMCLIB_SpeedLoop does not allow debugging of all internal variables. Use AMCLIB_SpeedLoopDebug for debugging purposes and replace it with AMCLIB_SpeedLoop once the debugging is finished.

Figure 70. Principal Field Oriented Control scheme with AMCLIB_SpeedLoop

Before using the FOC with a particular motor, the user needs to provide a set of coefficients through the `pCtrl` input pointer. The controller coefficient values can be calculated from motor parameters.

Refer to the following resources to find out how the NXP motor control tuning and debugging tools for NXP microcontrollers can help you deploy the FOC in your application:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

- AN4642 - Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM
- FREEMASTER - FreeMASTER Run-Time Debugging Tool

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.10.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.10.3 Function AMCLIB_SpeedLoop_F32

2.10.3.1 Declaration

```
void AMCLIB_SpeedLoop_F32(tFrac32 f32VelocityReq,
tFrac32 f32VelocityFbck, SWLIBS_2Syst_F32 *const pIDQReq,
AMCLIB_SPEED_LOOP_T_F32 *pCtrl);
```

2.10.3.2 Arguments

Table 74. AMCLIB_SpeedLoop_F32 arguments

Type	Name	Direction	Description
tFrac32	f32VelocityReq	**input**	Required electrical angular velocity (setpoint).
tFrac32	f32VelocityFbck	**input**	Actual electrical angular velocity from the feedback.
SWLIBS_2Syst_F32 *const	pIDQReq	**output**	Pointer to the structure with the required stator currents in the two-phase rotational orthogonal system (d-q). Only the q-axis component is written by the function.
AMCLIB_SPEED_LOOP_T_F32 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_SpeedLoop state.

2.10.3.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_SpeedLoop_F32.

Figure 71. Functions and data structures in AMCLIB_SpeedLoop_F32

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. All inputs and outputs of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

Table 75. Scaling constants

Scaling constant	Symbol	Calculation
Maximum phase current [A]	I_{MAX}	Maximum current depends on power stage capabilities.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 75. Scaling constants...*continued*

Maximum speed [rad/s]	Ω_{MAX}	Maximum application required speed, at least the motor electrical rated speed.

The proportional gain K_P and integral gain K_I of the speed controller (GFLIB_ControllerPIpAW_F32) can be calculated as follows:

$$K_P = 2 \cdot \xi \cdot \omega_0 \cdot \frac{J}{K_T}$$

$$K_I = \omega_0^2 \cdot \frac{J}{K_T}$$

Equation AMCLIB_SpeedLoop_F32_Eq1

where ξ is the speed loop attenuation, ω_0 is the speed loop natural frequency [rad/s], J is the moment of inertia [kg·m^2], K_T is the motor torque constant [N·m·A^{-1}].

The continuous-time controller parameters K_P and K_I are then discretized using the bilinear transform. The resulting difference equation coefficients must be scaled so that the input and output values of the controller fit in the fractional range.

$$K_{P_SC} = K_P \cdot \frac{\Omega_{MAX}}{I_{MAX}}$$

$$K_{I_SC} = \frac{K_I T_S}{2} \cdot \frac{\Omega_{MAX}}{I_{MAX}}$$

Equation AMCLIB_SpeedLoop_F32_Eq2

T_S is the sampling period [s]. In case the coefficients K_{P_SC} and K_{I_SC} exceed the fractional range [-1, 1) or they are extremely small (which could lead to loss of precision), their value must be bit-shifted by *NShiftP* and *NShiftI*, respectively. The shift amount can be calculated as

$$NShiftP = \lceil \log_2(|K_{P_SC}|) \rceil$$

$$NShiftI = \lceil \log_2(|K_{I_SC}|) \rceil$$

Equation AMCLIB_SpeedLoop_F32_Eq3

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 31st power of two (corresponds to the Q1.31 encoding):

$$pPIpAWQ.f32PropGain = \left(K_{P_SC} \cdot 2^{-NShiftP}\right) \cdot 2^{31}$$

$$pPIpAWQ.f32IntegGain = \left(K_{I_SC} \cdot 2^{-NShiftI}\right) \cdot 2^{31}$$

$$pPIpAWQ.s16PropGainShift = NShiftP$$

$$pPIpAWQ.s16IntegGainShift = NShiftI$$

$$pPIpAWQ.f32UpperLimit = 2^{31} - 1$$

$$pPIpAWQ.f32LowerLimit = -2^{31}$$

Equation AMCLIB_SpeedLoop_F32_Eq4

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The smoothing factor of the GDFLIB_FilterMA_F32 and the slope of the GFLIB_Ramp_F32 on the input of the speed controller should reflect the achievable dynamics of the drive. The AN4642 (Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM) can help with tuning of these parameters.

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

2.10.3.4 Code Example

```
#include "amclib.h"

AMCLIB_SPEED_LOOP_T_F32 SpeedLoop;
SWLIBS_2Syst_F32 IDQReq;       // required dq currents
tFrac32 f32VelocityReq;        // required velocity
tFrac32 f32VelocityFbck;       // actual velocity

void main (void)
{
    tFrac32 f32FilterMAWOut;
    tFrac32 f32ControllerPIpAWQOut;
    tFrac32 f32RampOut;

    // Initialize the parameters in SpeedLoop
    SpeedLoop.pFilterW.u16NSamples       = 3u;
    SpeedLoop.pPIpAWQ.f32PropGain        = (tFrac32)FRAC32(0.1234);
    SpeedLoop.pPIpAWQ.f32IntegGain       = (tFrac32)FRAC32(0.1675);
    SpeedLoop.pPIpAWQ.s16PropGainShift   = 1;
    SpeedLoop.pPIpAWQ.s16IntegGainShift  = 1;
    SpeedLoop.pPIpAWQ.f32LowerLimit      = (tFrac32)-2006823469L;
    SpeedLoop.pPIpAWQ.f32UpperLimit      = (tFrac32)2101527497L;
    SpeedLoop.pRamp.f32RampUp            = (tFrac32)FRAC32(0.4768);
    SpeedLoop.pRamp.f32RampDown          = (tFrac32)FRAC32(0.3754);

    // Clear AMCLIB_SpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit_F32(&SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit(&SpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoopInit(&SpeedLoop);

    // Initialize the AMCLIB_SpeedLoop state variables to predefined values
    // Warning: Parameters in SpeedLoop must be already initialized.
    f32FilterMAWOut = (tFrac32)123L;
    f32ControllerPIpAWQOut = (tFrac32)123L;
    f32RampOut = (tFrac32)123L;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState_F32(f32FilterMAWOut, f32ControllerPIpAWQOut,
      f32RampOut, &SpeedLoop);

    // Alternative 2: API call with implementation parameter
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState(f32FilterMAWOut, f32ControllerPIpAWQOut,
      f32RampOut, &SpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoopSetState(f32FilterMAWOut, f32ControllerPIpAWQOut,
      f32RampOut, &SpeedLoop);

    f32VelocityReq = (tFrac32)100L;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoop_F32(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoop(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &SpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoop(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &SpeedLoop);
}
```

2.10.4 Function AMCLIB_SpeedLoop_F16

2.10.4.1 Declaration

```
void AMCLIB_SpeedLoop_F16(tFrac16 f16VelocityReq,
tFrac16 f16VelocityFbck, SWLIBS_2Syst_F16 *const pIDQReq,
AMCLIB_SPEED_LOOP_T_F16 *pCtrl);
```

2.10.4.2 Arguments

Table 76. AMCLIB_SpeedLoop_F16 arguments

Type	Name	Direction	Description
tFrac16	f16VelocityReq	**input**	Required electrical angular velocity (setpoint).
tFrac16	f16VelocityFbck	**input**	Actual electrical angular velocity from the feedback.
SWLIBS_2Syst_F16 *const	pIDQReq	**output**	Pointer to the structure with the required stator currents in the two-phase rotational orthogonal system (d-q). Only the q-axis component is written by the function.
AMCLIB_SPEED_LOOP_T_F16 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_SpeedLoop state.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.10.4.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_SpeedLoop_F16.

Figure 72. Functions and data structures in AMCLIB_SpeedLoop_F16

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. All inputs and outputs of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

Table 77. Scaling constants

Scaling constant	Symbol	Calculation
Maximum phase current [A]	I_{MAX}	Maximum current depends on power stage capabilities.
Maximum speed [rad/s]	Ω_{MAX}	Maximum application required speed, at least the motor electrical rated speed.

The proportional gain K_P and integral gain K_I of the speed controller (GFLIB_ControllerPIpAW_F16) can be calculated as follows:

$$K_P = 2 \cdot \xi \cdot \omega_0 \cdot \frac{J}{K_T}$$

$$K_I = \omega_0^2 \cdot \frac{J}{K_T}$$

Equation AMCLIB_SpeedLoop_F16_Eq1

where ξ is the speed loop attenuation, ω_0 is the speed loop natural frequency [rad/s], J is the moment of inertia [kg·m^2], K_T is the motor torque constant [N·m·A^{-1}].

The continuous-time controller parameters K_P and K_I are then discretized using the bilinear transform. The resulting difference equation coefficients must be scaled so that the input and output values of the controller fit in the fractional range.

$$K_{P_SC} = K_P \cdot \frac{\Omega_{MAX}}{I_{MAX}}$$

$$K_{I_SC} = \frac{K_I T_S}{2} \cdot \frac{\Omega_{MAX}}{I_{MAX}}$$

Equation AMCLIB_SpeedLoop_F16_Eq2

T_S is the sampling period [s]. In case the coefficients K_{P_SC} and K_{I_SC} exceed the fractional range [-1, 1) or they are extremely small (which could lead to loss of precision), their value must be bit-shifted by *NShiftP* and *NShiftI*, respectively. The shift amount can be calculated as

$$NShiftP = \lceil \log_2(|K_{P_SC}|) \rceil$$

$$NShiftI = \lceil \log_2(|K_{I_SC}|) \rceil$$

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Equation AMCLIB_SpeedLoop_F16_Eq3

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 15th power of two (corresponds to the Q1.15 encoding):

$$pPIpAWQ.f16PropGain = \left(K_{P_SC} \cdot 2^{-NShiftP}\right) \cdot 2^{15}$$
$$pPIpAWQ.f16IntegGain = \left(K_{I_SC} \cdot 2^{-NShiftI}\right) \cdot 2^{15}$$
$$pPIpAWQ.s16PropGainShift = NShiftP$$
$$pPIpAWQ.s16IntegGainShift = NShiftI$$
$$pPIpAWQ.f16UpperLimit = 2^{15} - 1$$
$$pPIpAWQ.f16LowerLimit = -2^{15}$$

Equation AMCLIB_SpeedLoop_F16_Eq4

The smoothing factor of the GDFLIB_FilterMA_F16 and the slope of the GFLIB_Ramp_F16 on the input of the speed controller should reflect the achievable dynamics of the drive. The AN4642 (Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM) can help with tuning of these parameters.

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

2.10.4.4 Code Example

```
#include "amclib.h"

AMCLIB_SPEED_LOOP_T_F16 SpeedLoop;
SWLIBS_2Syst_F16 IDQReq;       // required dq currents
tFrac16 f16VelocityReq;        // required velocity
tFrac16 f16VelocityFbck;       // actual velocity

void main (void)
{
    tFrac16 f16FilterMAWOut;
    tFrac16 f16ControllerPIpAWQOut;
    tFrac32 f32RampOut;

    // Initialize the parameters in SpeedLoop
    SpeedLoop.pFilterW.u16NSamples      = 3u;
    SpeedLoop.pPIpAWQ.f16PropGain       = (tFrac16)FRAC16(0.1234);
    SpeedLoop.pPIpAWQ.f16IntegGain      = (tFrac16)FRAC16(0.1675);
    SpeedLoop.pPIpAWQ.s16PropGainShift  = 1;
    SpeedLoop.pPIpAWQ.s16IntegGainShift = 1;
    SpeedLoop.pPIpAWQ.f16LowerLimit     = (tFrac16)-30621;
    SpeedLoop.pPIpAWQ.f16UpperLimit     = (tFrac16)32066;
    SpeedLoop.pRamp.f32RampUp           = (tFrac32)FRAC32(0.4768);
    SpeedLoop.pRamp.f32RampDown         = (tFrac32)FRAC32(0.3754);

    // Clear AMCLIB_SpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit_F16(&SpeedLoop);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit(&SpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoopInit(&SpeedLoop);

    // Initialize the AMCLIB_SpeedLoop state variables to predefined values
    // Warning: Parameters in SpeedLoop must be already initialized.
    f16FilterMAWOut = (tFrac16)123L;
    f16ControllerPIpAWQOut = (tFrac16)123L;
    f32RampOut = (tFrac32)123L;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState_F16(f16FilterMAWOut, f16ControllerPIpAWQOut,
      f32RampOut, &SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState(f16FilterMAWOut, f16ControllerPIpAWQOut,
      f32RampOut, &SpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoopSetState(f16FilterMAWOut, f16ControllerPIpAWQOut,
      f32RampOut, &SpeedLoop);

    f16VelocityReq = (tFrac16)100;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoop_F16(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoop(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &SpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoop(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &SpeedLoop);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.10.5 Function AMCLIB_SpeedLoop_FLT

2.10.5.1 Declaration

```
void AMCLIB_SpeedLoop_FLT(tFloat fltVelocityReq, tFloat
fltVelocityFbck, SWLIBS_2Syst_FLT *const pIDQReq,
AMCLIB_SPEED_LOOP_T_FLT *pCtrl);
```

2.10.5.2 Arguments

Table 78. AMCLIB_SpeedLoop_FLT arguments

Type	Name	Direction	Description
tFloat	fltVelocityReq	**input**	Required electrical angular velocity (setpoint).
tFloat	fltVelocityFbck	**input**	Actual electrical angular velocity from the feedback.
SWLIBS_2Syst_FLT *const	pIDQReq	**output**	Pointer to the structure with the required stator currents in the two-phase rotational orthogonal system (d-q). Only the q-axis component is written by the function.
AMCLIB_SPEED_LOOP_T_FLT *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_SpeedLoop state.

2.10.5.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_SpeedLoop_FLT.

Figure 73. Functions and data structures in AMCLIB_SpeedLoop_FLT

Parameters of the speed PI controller (GFLIB_ControllerPIpAW_FLT) can be calculated using the following equations:

$$pPIpAWQ.fltPropGain = 2 \cdot \xi \cdot \omega_0 \cdot \frac{J}{K_T}$$

$$pPIpAWQ.fltIntegGain = \omega_0^2 \cdot \frac{J}{K_T} \cdot \frac{T_S}{2}$$

$$pPIpAWQ.fltUpperLimit = I_{MAX}$$

$$pPIpAWQ.fltLowerLimit = -I_{MAX}$$

Equation AMCLIB_SpeedLoop_FLT_Eq1

where ξ is the speed loop attenuation, ω_0 is the speed loop natural frequency [rad/s], J is the moment of inertia [kg·m^2], K_T is the motor torque constant [N·m·A^{-1}], T_S is the sampling period [s], and I_{MAX} is the maximum phase current [A].

The smoothing factor of the GDFLIB_FilterMA_FLT and the slope of the GFLIB_Ramp_FLT on the input of the speed controller should reflect the achievable dynamics of the drive. The AN4642 (Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM) can help with tuning of these parameters.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.10.5.4 Code Example

```
#include "amclib.h"

AMCLIB_SPEED_LOOP_T_FLT SpeedLoop;
SWLIBS_2Syst_FLT IDQReq;       // required dq currents
tFloat fltVelocityReq;         // required velocity
tFloat fltVelocityFbck;        // actual velocity

void main (void)
{
    tFloat fltFilterMAWOut;
    tFloat fltControllerPIpAWQOut;
    tFloat fltRampOut;

    // Initialize the parameters in SpeedLoop
    SpeedLoop.pFilterW.fltLambda      = (tFloat)0.5;
    SpeedLoop.pPIpAWQ.fltPropGain     = (tFloat)0.1234;
    SpeedLoop.pPIpAWQ.fltIntegGain    = (tFloat)0.1675;
    SpeedLoop.pPIpAWQ.fltLowerLimit   = (tFloat)-0.934499979019165;
    SpeedLoop.pPIpAWQ.fltUpperLimit   = (tFloat)0.978600025177002;
    SpeedLoop.pRamp.fltRampUp         = (tFloat)0.4768;
    SpeedLoop.pRamp.fltRampDown       = (tFloat)0.3754;

    // Clear AMCLIB_SpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit_FLT(&SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit(&SpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_SpeedLoopInit(&SpeedLoop);

    // Initialize the AMCLIB_SpeedLoop state variables to predefined values
    // Warning: Parameters in SpeedLoop must be already initialized.
    fltFilterMAWOut = 123.0F;
    fltControllerPIpAWQOut = 123.0F;
    fltRampOut = 123.0F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState_FLT(fltFilterMAWOut, fltControllerPIpAWQOut,
      fltRampOut, &SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState(fltFilterMAWOut, fltControllerPIpAWQOut,
      fltRampOut, &SpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_SpeedLoopSetState(fltFilterMAWOut, fltControllerPIpAWQOut,
      fltRampOut, &SpeedLoop);

    fltVelocityReq = 100.0F;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoop_FLT(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoop(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &SpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_SpeedLoop(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &SpeedLoop);
}
```

2.10.6 Function AMCLIB_SpeedLoopInit

2.10.6.1 Description

This function clears the AMCLIB_SpeedLoop state variables.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.10.6.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.10.6.3 Function AMCLIB_SpeedLoopInit_F32

2.10.6.3.1 Declaration

```
void AMCLIB_SpeedLoopInit_F32(AMCLIB_SPEED_LOOP_T_F32 *const
pCtrl);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.10.6.3.2 Arguments

Table 79. AMCLIB_SpeedLoopInit_F32 arguments

Type	Name	Direction	Description
AMCLIB_SPEED_LOOP_T_F32 *const	pCtrl	**input, output**	Pointer to the structure with AMCLIB_SpeedLoop state.

Note: *If pCtrl points to a structure of type AMCLIB_SPEED_LOOP_DEBUG_T_F32, it must be recasted to AMCLIB_SPEED_LOOP_T_F32 *.*

2.10.6.3.3 Code Example

```
#include "amclib.h"

AMCLIB_SPEED_LOOP_T_F32 SpeedLoop;
SWLIBS_2Syst_F32 IDQReq;       // required dq currents
tFrac32 f32VelocityReq;        // required velocity
tFrac32 f32VelocityFbck;       // actual velocity

void main (void)
{
    tFrac32 f32FilterMAWOut;
    tFrac32 f32ControllerPIpAWQOut;
    tFrac32 f32RampOut;

    // Initialize the parameters in SpeedLoop
    SpeedLoop.pFilterW.u16NSamples      = 3u;
    SpeedLoop.pPIpAWQ.f32PropGain       = (tFrac32)FRAC32(0.1234);
    SpeedLoop.pPIpAWQ.f32IntegGain      = (tFrac32)FRAC32(0.1675);
    SpeedLoop.pPIpAWQ.s16PropGainShift  = 1;
    SpeedLoop.pPIpAWQ.s16IntegGainShift = 1;
    SpeedLoop.pPIpAWQ.f32LowerLimit     = (tFrac32)-2006823469L;
    SpeedLoop.pPIpAWQ.f32UpperLimit     = (tFrac32)2101527497L;
    SpeedLoop.pRamp.f32RampUp           = (tFrac32)FRAC32(0.4768);
    SpeedLoop.pRamp.f32RampDown         = (tFrac32)FRAC32(0.3754);

    // Clear AMCLIB_SpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit_F32(&SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit(&SpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoopInit(&SpeedLoop);

    // Initialize the AMCLIB_SpeedLoop state variables to predefined values
    // Warning: Parameters in SpeedLoop must be already initialized.
    f32FilterMAWOut = (tFrac32)123L;
    f32ControllerPIpAWQOut = (tFrac32)123L;
    f32RampOut = (tFrac32)123L;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState_F32(f32FilterMAWOut, f32ControllerPIpAWQOut,
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
      f32RampOut, &SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState(f32FilterMAWOut, f32ControllerPIpAWQOut,
      f32RampOut, &SpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoopSetState(f32FilterMAWOut, f32ControllerPIpAWQOut,
      f32RampOut, &SpeedLoop);

    f32VelocityReq = (tFrac32)100L;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoop_F32(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoop(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &SpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoop(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &SpeedLoop);
}
```

2.10.6.4 Function AMCLIB_SpeedLoopInit_F16

2.10.6.4.1 Declaration

```
void AMCLIB_SpeedLoopInit_F16(AMCLIB_SPEED_LOOP_T_F16 *const
pCtrl);
```

2.10.6.4.2 Arguments

Table 80. AMCLIB_SpeedLoopInit_F16 arguments

Type	Name	Direction	Description
AMCLIB_SPEED_LOOP_T_F16 *const	pCtrl	**input, output**	Pointer to the structure with AMCLIB_SpeedLoop state.

Note: *If pCtrl points to a structure of type AMCLIB_SPEED_LOOP_DEBUG_T_F16, it must be recasted to AMCLIB_SPEED_LOOP_T_F16 *.*

2.10.6.4.3 Code Example

```
#include "amclib.h"
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
AMCLIB_SPEED_LOOP_T_F16 SpeedLoop;
SWLIBS_2Syst_F16 IDQReq;       // required dq currents
tFrac16 f16VelocityReq;        // required velocity
tFrac16 f16VelocityFbck;       // actual velocity

void main (void)
{
    tFrac16 f16FilterMAWOut;
    tFrac16 f16ControllerPIpAWQOut;
    tFrac32 f32RampOut;

    // Initialize the parameters in SpeedLoop
    SpeedLoop.pFilterW.u16NSamples      = 3u;
    SpeedLoop.pPIpAWQ.f16PropGain       = (tFrac16)FRAC16(0.1234);
    SpeedLoop.pPIpAWQ.f16IntegGain      = (tFrac16)FRAC16(0.1675);
    SpeedLoop.pPIpAWQ.s16PropGainShift  = 1;
    SpeedLoop.pPIpAWQ.s16IntegGainShift = 1;
    SpeedLoop.pPIpAWQ.f16LowerLimit     = (tFrac16)-30621;
    SpeedLoop.pPIpAWQ.f16UpperLimit     = (tFrac16)32066;
    SpeedLoop.pRamp.f32RampUp           = (tFrac32)FRAC32(0.4768);
    SpeedLoop.pRamp.f32RampDown         = (tFrac32)FRAC32(0.3754);

    // Clear AMCLIB_SpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit_F16(&SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit(&SpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoopInit(&SpeedLoop);

    // Initialize the AMCLIB_SpeedLoop state variables to predefined values
    // Warning: Parameters in SpeedLoop must be already initialized.
    f16FilterMAWOut = (tFrac16)123L;
    f16ControllerPIpAWQOut = (tFrac16)123L;
    f32RampOut = (tFrac32)123L;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState_F16(f16FilterMAWOut, f16ControllerPIpAWQOut,
      f32RampOut, &SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState(f16FilterMAWOut, f16ControllerPIpAWQOut,
      f32RampOut, &SpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoopSetState(f16FilterMAWOut, f16ControllerPIpAWQOut,
      f32RampOut, &SpeedLoop);

    f16VelocityReq = (tFrac16)100;
    while(1);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoop_F16(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoop(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &SpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoop(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &SpeedLoop);
}
```

2.10.6.5 Function AMCLIB_SpeedLoopInit_FLT

2.10.6.5.1 Declaration

```
void AMCLIB_SpeedLoopInit_FLT(AMCLIB_SPEED_LOOP_T_FLT *const
pCtrl);
```

2.10.6.5.2 Arguments

Table 81. AMCLIB_SpeedLoopInit_FLT arguments

Type	Name	Direction	Description
AMCLIB_SPEED_LOOP_T_FLT *const	pCtrl	**input, output**	Pointer to the structure with AMCLIB_SpeedLoop state.

Note: *If pCtrl points to a structure of type AMCLIB_SPEED_LOOP_DEBUG_T_FLT, it must be recasted to AMCLIB_SPEED_LOOP_T_FLT *.*

The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.

2.10.6.5.3 Code Example

```
#include "amclib.h"

AMCLIB_SPEED_LOOP_T_FLT SpeedLoop;
SWLIBS_2Syst_FLT IDQReq;       // required dq currents
tFloat fltVelocityReq;         // required velocity
tFloat fltVelocityFbck;        // actual velocity

void main (void)
{
    tFloat fltFilterMAWOut;
    tFloat fltControllerPIpAWQOut;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    tFloat fltRampOut;

    // Initialize the parameters in SpeedLoop
    SpeedLoop.pFilterW.fltLambda      = (tFloat)0.5;
    SpeedLoop.pPIpAWQ.fltPropGain     = (tFloat)0.1234;
    SpeedLoop.pPIpAWQ.fltIntegGain    = (tFloat)0.1675;
    SpeedLoop.pPIpAWQ.fltLowerLimit   = (tFloat)-0.934499979019165;
    SpeedLoop.pPIpAWQ.fltUpperLimit   = (tFloat)0.978600025177002;
    SpeedLoop.pRamp.fltRampUp         = (tFloat)0.4768;
    SpeedLoop.pRamp.fltRampDown       = (tFloat)0.3754;

    // Clear AMCLIB_SpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit_FLT(&SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit(&SpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_SpeedLoopInit(&SpeedLoop);

    // Initialize the AMCLIB_SpeedLoop state variables to predefined values
    // Warning: Parameters in SpeedLoop must be already initialized.
    fltFilterMAWOut = 123.0F;
    fltControllerPIpAWQOut = 123.0F;
    fltRampOut = 123.0F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState_FLT(fltFilterMAWOut, fltControllerPIpAWQOut,
      fltRampOut, &SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState(fltFilterMAWOut, fltControllerPIpAWQOut,
      fltRampOut, &SpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_SpeedLoopSetState(fltFilterMAWOut, fltControllerPIpAWQOut,
      fltRampOut, &SpeedLoop);

    fltVelocityReq = 100.0F;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoop_FLT(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &SpeedLoop);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoop(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &SpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_SpeedLoop(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &SpeedLoop);
}
```

2.10.7 Function AMCLIB_SpeedLoopSetState

2.10.7.1 Description

This function initializes the AMCLIB_SpeedLoop state variables to achieve the required output values.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.10.7.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.10.7.3 Function AMCLIB_SpeedLoopSetState_F32

2.10.7.3.1 Declaration

```
void AMCLIB_SpeedLoopSetState_F32(tFrac32 f32FilterMAWOut,
tFrac32 f32ControllerPIpAWQOut, tFrac32 f32RampOut,
AMCLIB_SPEED_LOOP_T_F32 *pCtrl);
```

2.10.7.3.2 Arguments

Table 82. AMCLIB_SpeedLoopSetState_F32 arguments

Type	Name	Direction	Description
tFrac32	f32FilterMAWOut	**input**	Required output of the FilterMA.
tFrac32	f32ControllerPIpAWQOut	**input**	Required output of the ControllerPIpAW.
tFrac32	f32RampOut	**input**	Required output of the speed ramp.
AMCLIB_SPEED_LOOP_T_F32 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_SpeedLoop state.

CAUTION: *Set the parameters in the structure pointed to by pCtrl before calling this function.*

Note: *If pCtrl points to a structure of type AMCLIB_SPEED_LOOP_DEBUG_T_F32, it must be recasted to AMCLIB_SPEED_LOOP_T_F32 *.*

2.10.7.3.3 Code Example

```
#include "amclib.h"
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
AMCLIB_SPEED_LOOP_T_F32 SpeedLoop;
SWLIBS_2Syst_F32 IDQReq;       // required dq currents
tFrac32 f32VelocityReq;        // required velocity
tFrac32 f32VelocityFbck;       // actual velocity

void main (void)
{
    tFrac32 f32FilterMAWOut;
    tFrac32 f32ControllerPIpAWQOut;
    tFrac32 f32RampOut;

    // Initialize the parameters in SpeedLoop
    SpeedLoop.pFilterW.u16NSamples      = 3u;
    SpeedLoop.pPIpAWQ.f32PropGain       = (tFrac32)FRAC32(0.1234);
    SpeedLoop.pPIpAWQ.f32IntegGain      = (tFrac32)FRAC32(0.1675);
    SpeedLoop.pPIpAWQ.s16PropGainShift  = 1;
    SpeedLoop.pPIpAWQ.s16IntegGainShift = 1;
    SpeedLoop.pPIpAWQ.f32LowerLimit     = (tFrac32)-2006823469L;
    SpeedLoop.pPIpAWQ.f32UpperLimit     = (tFrac32)2101527497L;
    SpeedLoop.pRamp.f32RampUp           = (tFrac32)FRAC32(0.4768);
    SpeedLoop.pRamp.f32RampDown         = (tFrac32)FRAC32(0.3754);

    // Clear AMCLIB_SpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit_F32(&SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit(&SpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoopInit(&SpeedLoop);

    // Initialize the AMCLIB_SpeedLoop state variables to predefined values
    // Warning: Parameters in SpeedLoop must be already initialized.
    f32FilterMAWOut = (tFrac32)123L;
    f32ControllerPIpAWQOut = (tFrac32)123L;
    f32RampOut = (tFrac32)123L;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState_F32(f32FilterMAWOut, f32ControllerPIpAWQOut,
      f32RampOut, &SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState(f32FilterMAWOut, f32ControllerPIpAWQOut,
      f32RampOut, &SpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoopSetState(f32FilterMAWOut, f32ControllerPIpAWQOut,
      f32RampOut, &SpeedLoop);

    f32VelocityReq = (tFrac32)100L;
    while(1);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoop_F32(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoop(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &SpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoop(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &SpeedLoop);
}
```

2.10.7.4 Function AMCLIB_SpeedLoopSetState_F16

2.10.7.4.1 Declaration

```
void AMCLIB_SpeedLoopSetState_F16(tFrac16 f16FilterMAWOut,
tFrac16 f16ControllerPIpAWQOut, tFrac32 f32RampOut,
AMCLIB_SPEED_LOOP_T_F16 *pCtrl);
```

2.10.7.4.2 Arguments

Table 83. AMCLIB_SpeedLoopSetState_F16 arguments

Type	Name	Direction	Description
tFrac16	f16FilterMAWOut	**input**	Required output of the FilterMA.
tFrac16	f16ControllerPIpAWQOut	**input**	Required output of the ControllerPIpAW.
tFrac32	f32RampOut	**input**	Required output of the speed ramp.
AMCLIB_SPEED_LOOP_T_F16 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_SpeedLoop state.

CAUTION: *Set the parameters in the structure pointed to by pCtrl before calling this function.*

Note: *If pCtrl points to a structure of type* AMCLIB_SPEED_LOOP_DEBUG_T_F16*, it must be recasted to* AMCLIB_SPEED_LOOP_T_F16 *.*

2.10.7.4.3 Code Example

```
#include "amclib.h"

AMCLIB_SPEED_LOOP_T_F16 SpeedLoop;
SWLIBS_2Syst_F16 IDQReq;       // required dq currents
tFrac16 f16VelocityReq;        // required velocity
tFrac16 f16VelocityFbck;       // actual velocity

void main (void)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
{
    tFrac16 f16FilterMAWOut;
    tFrac16 f16ControllerPIpAWQOut;
    tFrac32 f32RampOut;

    // Initialize the parameters in SpeedLoop
    SpeedLoop.pFilterW.u16NSamples      = 3u;
    SpeedLoop.pPIpAWQ.f16PropGain       = (tFrac16)FRAC16(0.1234);
    SpeedLoop.pPIpAWQ.f16IntegGain      = (tFrac16)FRAC16(0.1675);
    SpeedLoop.pPIpAWQ.s16PropGainShift  = 1;
    SpeedLoop.pPIpAWQ.s16IntegGainShift = 1;
    SpeedLoop.pPIpAWQ.f16LowerLimit     = (tFrac16)-30621;
    SpeedLoop.pPIpAWQ.f16UpperLimit     = (tFrac16)32066;
    SpeedLoop.pRamp.f32RampUp           = (tFrac32)FRAC32(0.4768);
    SpeedLoop.pRamp.f32RampDown         = (tFrac32)FRAC32(0.3754);

    // Clear AMCLIB_SpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit_F16(&SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit(&SpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoopInit(&SpeedLoop);

    // Initialize the AMCLIB_SpeedLoop state variables to predefined values
    // Warning: Parameters in SpeedLoop must be already initialized.
    f16FilterMAWOut = (tFrac16)123L;
    f16ControllerPIpAWQOut = (tFrac16)123L;
    f32RampOut = (tFrac32)123L;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState_F16(f16FilterMAWOut, f16ControllerPIpAWQOut,
      f32RampOut, &SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState(f16FilterMAWOut, f16ControllerPIpAWQOut,
      f32RampOut, &SpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoopSetState(f16FilterMAWOut, f16ControllerPIpAWQOut,
      f32RampOut, &SpeedLoop);

    f16VelocityReq = (tFrac16)100;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    AMCLIB_SpeedLoop_F16(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoop(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &SpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoop(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &SpeedLoop);
}
```

2.10.7.5 Function AMCLIB_SpeedLoopSetState_FLT

2.10.7.5.1 Declaration

```
void AMCLIB_SpeedLoopSetState_FLT(tFloat fltFilterMAWOut,
tFloat fltControllerPIpAWQOut, tFloat fltRampOut,
AMCLIB_SPEED_LOOP_T_FLT *pCtrl);
```

2.10.7.5.2 Arguments

Table 84. AMCLIB_SpeedLoopSetState_FLT arguments

Type	Name	Direction	Description
tFloat	fltFilterMAWOut	**input**	Required output of the FilterMA.
tFloat	fltControllerPIpAWQOut	**input**	Required output of the ControllerPIpAW.
tFloat	fltRampOut	**input**	Required output of the speed ramp.
AMCLIB_SPEED_LOOP_T_FLT *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_SpeedLoop state.

CAUTION: *Set the parameters in the structure pointed to by pCtrl before calling this function.*

Note: *If pCtrl points to a structure of type AMCLIB_SPEED_LOOP_DEBUG_T_FLT, it must be recasted to AMCLIB_SPEED_LOOP_T_FLT *.*

The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.

2.10.7.5.3 Code Example

```
#include "amclib.h"

AMCLIB_SPEED_LOOP_T_FLT SpeedLoop;
SWLIBS_2Syst_FLT IDQReq;       // required dq currents
tFloat fltVelocityReq;         // required velocity
tFloat fltVelocityFbck;        // actual velocity

void main (void)
{
    tFloat fltFilterMAWOut;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    tFloat fltControllerPIpAWQOut;
    tFloat fltRampOut;

    // Initialize the parameters in SpeedLoop
    SpeedLoop.pFilterW.fltLambda      = (tFloat)0.5;
    SpeedLoop.pPIpAWQ.fltPropGain     = (tFloat)0.1234;
    SpeedLoop.pPIpAWQ.fltIntegGain    = (tFloat)0.1675;
    SpeedLoop.pPIpAWQ.fltLowerLimit   = (tFloat)-0.934499979019165;
    SpeedLoop.pPIpAWQ.fltUpperLimit   = (tFloat)0.978600025177002;
    SpeedLoop.pRamp.fltRampUp         = (tFloat)0.4768;
    SpeedLoop.pRamp.fltRampDown       = (tFloat)0.3754;

    // Clear AMCLIB_SpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit_FLT(&SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit(&SpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_SpeedLoopInit(&SpeedLoop);

    // Initialize the AMCLIB_SpeedLoop state variables to predefined values
    // Warning: Parameters in SpeedLoop must be already initialized.
    fltFilterMAWOut = 123.0F;
    fltControllerPIpAWQOut = 123.0F;
    fltRampOut = 123.0F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState_FLT(fltFilterMAWOut, fltControllerPIpAWQOut,
      fltRampOut, &SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState(fltFilterMAWOut, fltControllerPIpAWQOut,
      fltRampOut, &SpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_SpeedLoopSetState(fltFilterMAWOut, fltControllerPIpAWQOut,
      fltRampOut, &SpeedLoop);

    fltVelocityReq = 100.0F;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoop_FLT(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &SpeedLoop);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoop(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &SpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_SpeedLoop(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &SpeedLoop);
}
```

2.10.8 Function AMCLIB_SpeedLoopDebug

This function adjusts the torque of the motor to achieve the required speed. Debugging information is provided.

2.10.8.1 Description

This library function implements the speed control loop which is the outer loop in the cascade control structure of the speed FOC. FOC (also called vector control) is a widely used control strategy for Permanent Magnet Synchronous Motors (PMSM). FOC is characterized by smooth rotation over the entire speed range of the motor, full torque control at zero speed, and fast acceleration/deceleration. A PI controller in speed closed-loop system maintains the required speed by setting the torque producing current in the q-axis.

AMCLIB_SpeedLoopDebug implements the speed controller in the outer control loop highlighted in Figure 74 . AMCLIB_SpeedLoopDebug provides the same functionality as AMCLIB_SpeedLoop. Additionally, this function allows debugging of all internal variables. The debugging outputs are provided in the structure pointed to by `pCtrl`. Replace AMCLIB_SpeedLoopDebug by AMCLIB_SpeedLoop once the debugging is finished.

Figure 74. Principal Field Oriented Control scheme with AMCLIB_SpeedLoop

2.10.8.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.10.8.3 Function AMCLIB_SpeedLoopDebug_F32

2.10.8.3.1 Declaration

```
void AMCLIB_SpeedLoopDebug_F32(tFrac32 f32VelocityReq,
tFrac32 f32VelocityFbck, SWLIBS_2Syst_F32 *const pIDQReq,
AMCLIB_SPEED_LOOP_DEBUG_T_F32 *pCtrl);
```

2.10.8.3.2 Arguments

Table 85. AMCLIB_SpeedLoopDebug_F32 arguments

Type	Name	Direction	Description
tFrac32	f32VelocityReq	**input**	Required electrical angular velocity (setpoint).
tFrac32	f32VelocityFbck	**input**	Actual electrical angular velocity from the feedback.
SWLIBS_2Syst_F32 *const	pIDQReq	**output**	Pointer to the structure with the required stator currents in the two-phase rotational orthogonal system (d-q). Only the q-axis component is written by the function.
AMCLIB_SPEED_LOOP_DEBUG_T_F32 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_SpeedLoop state and debugging information.

2.10.8.3.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_SpeedLoopDebug_F32.

Figure 75. Functions and data structures in AMCLIB_SpeedLoopDebug_F32

Refer to the description of AMCLIB_SpeedLoop_F32 function on how to set up the controller parameters.

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

2.10.8.3.4 Code Example

```
#include "amclib.h"

AMCLIB_SPEED_LOOP_DEBUG_T_F32 SpeedLoop;
SWLIBS_2Syst_F32 IDQReq;        // required dq currents
tFrac32 f32VelocityReq;         // required velocity
tFrac32 f32VelocityFbck;        // actual velocity

void main (void)
{
    tFrac32 f32FilterMAWOut;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    tFrac32 f32ControllerPIpAWQOut;
    tFrac32 f32RampOut;

    // Initialize the parameters in SpeedLoop
    SpeedLoop.pFilterW.u16NSamples      = 3u;
    SpeedLoop.pPIpAWQ.f32PropGain       = (tFrac32)FRAC32(0.1234);
    SpeedLoop.pPIpAWQ.f32IntegGain      = (tFrac32)FRAC32(0.1675);
    SpeedLoop.pPIpAWQ.s16PropGainShift  = 1;
    SpeedLoop.pPIpAWQ.s16IntegGainShift = 1;
    SpeedLoop.pPIpAWQ.f32LowerLimit     = (tFrac32)-2006823469L;
    SpeedLoop.pPIpAWQ.f32UpperLimit     = (tFrac32)2101527497L;
    SpeedLoop.pRamp.f32RampUp           = (tFrac32)FRAC32(0.4768);
    SpeedLoop.pRamp.f32RampDown         = (tFrac32)FRAC32(0.3754);

    // Clear AMCLIB_SpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit_F32((AMCLIB_SPEED_LOOP_T_F32 *)&SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit((AMCLIB_SPEED_LOOP_T_F32 *)&SpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoopInit((AMCLIB_SPEED_LOOP_T_F32 *)&SpeedLoop);

    // Initialize the AMCLIB_SpeedLoop state variables to predefined values
    // Warning: Parameters in SpeedLoop must be already initialized.
    f32FilterMAWOut = (tFrac32)123L;
    f32ControllerPIpAWQOut = (tFrac32)123L;
    f32RampOut = (tFrac32)123L;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState_F32(f32FilterMAWOut, f32ControllerPIpAWQOut,
      f32RampOut, (AMCLIB_SPEED_LOOP_T_F32 *)&SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState(f32FilterMAWOut, f32ControllerPIpAWQOut,
      f32RampOut, (AMCLIB_SPEED_LOOP_T_F32 *)&SpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoopSetState(f32FilterMAWOut, f32ControllerPIpAWQOut,
      f32RampOut, (AMCLIB_SPEED_LOOP_T_F32 *)&SpeedLoop);

    f32VelocityReq = (tFrac32)100L;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopDebug_F32(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &SpeedLoop);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopDebug(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &SpeedLoop, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoopDebug(f32VelocityReq, f32VelocityFbck,
      &IDQReq, &SpeedLoop);
}
```

2.10.8.4 Function AMCLIB_SpeedLoopDebug_F16

2.10.8.4.1 Declaration

```
void AMCLIB_SpeedLoopDebug_F16(tFrac16 f16VelocityReq,
tFrac16 f16VelocityFbck, SWLIBS_2Syst_F16 *const pIDQReq,
AMCLIB_SPEED_LOOP_DEBUG_T_F16 *pCtrl);
```

2.10.8.4.2 Arguments

Table 86. AMCLIB_SpeedLoopDebug_F16 arguments

Type	Name	Direction	Description
tFrac16	f16VelocityReq	**input**	Required electrical angular velocity (setpoint).
tFrac16	f16VelocityFbck	**input**	Actual electrical angular velocity from the feedback.
SWLIBS_2Syst_F16 *const	pIDQReq	**output**	Pointer to the structure with the required stator currents in the two-phase rotational orthogonal system (d-q). Only the q-axis component is written by the function.
AMCLIB_SPEED_LOOP_DEBUG_T_F16 *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_SpeedLoop state and debugging information.

2.10.8.4.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_SpeedLoopDebug_F16.

Figure 76. Functions and data structures in AMCLIB_SpeedLoopDebug_F16

Refer to the description of AMCLIB_SpeedLoop_F16 function on how to set up the controller parameters.

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.10.8.4.4 Code Example

```
#include "amclib.h"

AMCLIB_SPEED_LOOP_DEBUG_T_F16 SpeedLoop;
SWLIBS_2Syst_F16 IDQReq;       // required dq currents
tFrac16 f16VelocityReq;        // required velocity
tFrac16 f16VelocityFbck;       // actual velocity

void main (void)
{
    tFrac16 f16FilterMAWOut;
    tFrac16 f16ControllerPIpAWQOut;
    tFrac32 f32RampOut;

    // Initialize the parameters in SpeedLoop
    SpeedLoop.pFilterW.u16NSamples      = 3u;
    SpeedLoop.pPIpAWQ.f16PropGain       = (tFrac16)FRAC16(0.1234);
    SpeedLoop.pPIpAWQ.f16IntegGain      = (tFrac16)FRAC16(0.1675);
    SpeedLoop.pPIpAWQ.s16PropGainShift  = 1;
    SpeedLoop.pPIpAWQ.s16IntegGainShift = 1;
    SpeedLoop.pPIpAWQ.f16LowerLimit     = (tFrac16)-30621;
    SpeedLoop.pPIpAWQ.f16UpperLimit     = (tFrac16)32066;
    SpeedLoop.pRamp.f32RampUp           = (tFrac32)FRAC32(0.4768);
    SpeedLoop.pRamp.f32RampDown         = (tFrac32)FRAC32(0.3754);

    // Clear AMCLIB_SpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit_F16((AMCLIB_SPEED_LOOP_T_F16 *)&SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit((AMCLIB_SPEED_LOOP_T_F16 *)&SpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoopInit((AMCLIB_SPEED_LOOP_T_F16 *)&SpeedLoop);

    // Initialize the AMCLIB_SpeedLoop state variables to predefined values
    // Warning: Parameters in SpeedLoop must be already initialized.
    f16FilterMAWOut = (tFrac16)123L;
    f16ControllerPIpAWQOut = (tFrac16)123L;
    f32RampOut = (tFrac32)123L;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState_F16(f16FilterMAWOut, f16ControllerPIpAWQOut,
      f32RampOut, (AMCLIB_SPEED_LOOP_T_F16 *)&SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState(f16FilterMAWOut, f16ControllerPIpAWQOut,
      f32RampOut, (AMCLIB_SPEED_LOOP_T_F16 *)&SpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoopSetState(f16FilterMAWOut, f16ControllerPIpAWQOut,
      f32RampOut, (AMCLIB_SPEED_LOOP_T_F16 *)&SpeedLoop);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    f16VelocityReq = (tFrac16)100;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopDebug_F16(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopDebug(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &SpeedLoop, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_SpeedLoopDebug(f16VelocityReq, f16VelocityFbck,
      &IDQReq, &SpeedLoop);
}
```

2.10.8.5 Function AMCLIB_SpeedLoopDebug_FLT

2.10.8.5.1 Declaration

```
void AMCLIB_SpeedLoopDebug_FLT(tFloat fltVelocityReq,
tFloat fltVelocityFbck, SWLIBS_2Syst_FLT *const pIDQReq,
AMCLIB_SPEED_LOOP_DEBUG_T_FLT *pCtrl);
```

2.10.8.5.2 Arguments

Table 87. AMCLIB_SpeedLoopDebug_FLT arguments

Type	Name	Direction	Description
tFloat	fltVelocityReq	**input**	Required electrical angular velocity (setpoint).
tFloat	fltVelocityFbck	**input**	Actual electrical angular velocity from the feedback.
SWLIBS_2Syst_FLT *const	pIDQReq	**output**	Pointer to the structure with the required stator currents in the two-phase rotational orthogonal system (d-q). Only the q-axis component is written by the function.
AMCLIB_SPEED_LOOP_DEBUG_T_FLT *	pCtrl	**input, output**	Pointer to the structure with AMCLIB_SpeedLoop state and debugging information.

2.10.8.5.3 Implementation details

The following block diagram shows the internal functions and data structures of AMCLIB_SpeedLoopDebug_FLT.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 77. Functions and data structures in AMCLIB_SpeedLoopDebug_FLT

Refer to the description of AMCLIB_SpeedLoop_FLT function on how to set up the controller parameters.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.10.8.5.4 Code Example

```
#include "amclib.h"

AMCLIB_SPEED_LOOP_DEBUG_T_FLT SpeedLoop;
SWLIBS_2Syst_FLT IDQReq;        // required dq currents
tFloat fltVelocityReq;          // required velocity
tFloat fltVelocityFbck;         // actual velocity

void main (void)
{
    tFloat fltFilterMAWOut;
    tFloat fltControllerPIpAWQOut;
    tFloat fltRampOut;

    // Initialize the parameters in SpeedLoop
    SpeedLoop.pFilterW.fltLambda       = (tFloat)0.5;
    SpeedLoop.pPIpAWQ.fltPropGain      = (tFloat)0.1234;
    SpeedLoop.pPIpAWQ.fltIntegGain     = (tFloat)0.1675;
    SpeedLoop.pPIpAWQ.fltLowerLimit    = (tFloat)-0.934499979019165;
    SpeedLoop.pPIpAWQ.fltUpperLimit    = (tFloat)0.978600025177002;
    SpeedLoop.pRamp.fltRampUp          = (tFloat)0.4768;
    SpeedLoop.pRamp.fltRampDown        = (tFloat)0.3754;

    // Clear AMCLIB_SpeedLoop state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit_FLT((AMCLIB_SPEED_LOOP_T_FLT *)&SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopInit((AMCLIB_SPEED_LOOP_T_FLT *)&SpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_SpeedLoopInit((AMCLIB_SPEED_LOOP_T_FLT *)&SpeedLoop);

    // Initialize the AMCLIB_SpeedLoop state variables to predefined values
    // Warning: Parameters in SpeedLoop must be already initialized.
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    fltFilterMAWOut = 123.0F;
    fltControllerPIpAWQOut = 123.0F;
    fltRampOut = 123.0F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState_FLT(fltFilterMAWOut, fltControllerPIpAWQOut,
      fltRampOut, (AMCLIB_SPEED_LOOP_T_FLT *)&SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopSetState(fltFilterMAWOut, fltControllerPIpAWQOut,
      fltRampOut, (AMCLIB_SPEED_LOOP_T_FLT *)&SpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_SpeedLoopSetState(fltFilterMAWOut, fltControllerPIpAWQOut,
      fltRampOut, (AMCLIB_SPEED_LOOP_T_FLT *)&SpeedLoop);

    fltVelocityReq = 100.0F;
    while(1);
}

// Periodical function or interrupt - speed control loop
void SlowLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopDebug_FLT(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &SpeedLoop);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_SpeedLoopDebug(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &SpeedLoop, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating point implementation is selected
    // as default.
    AMCLIB_SpeedLoopDebug(fltVelocityReq, fltVelocityFbck,
      &IDQReq, &SpeedLoop);
}
```

2.11 Function AMCLIB_TrackObsrv

This function implements the position tracking observer algorithm which calculates the estimated angular velocity and position of the rotor.

2.11.1 Description

This function takes the phase error Θ_{err} calculated by the the AMCLIB_BemfObsrvDQ and uses it to estimate the angular velocity and position of the rotor.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 78. Principal Field Oriented Control scheme with AMCLIB_TrackObsrv

The estimated rotor position is obtained by driving the phase error Θ_{err} of the γ-δ reference frame towards zero in a phase-locked loop (PLL). The PLL structure is depicted in the following diagram.

Figure 79. Position tracking PLL

The position tracking structure, described in Figure 79 , can be linearized around the operating point $\Theta_{estim} = \Theta_e$. The linear approximation of the position tracking observer with a PI controller is shown in the following picture:

Figure 80. Linear approximation of the position tracking observer with a PI controller

Linearized position tracking observer, depicted in Figure 80 has the following transfer function:

$$G(s) = \frac{\theta_{estim}(s)}{\theta_e(s)} = \frac{sK_p + K_i}{s^2 + sK_p + K_i}$$

Equation AMCLIB_TrackObsrv_Eq1

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Assuming that the $s*K_p$ term in the numerator as negligible, the controller gains K_p and K_i can be calculated by comparing the characteristic polynomial of the resulting transfer function to a standard second order system polynomial:

$$K_p = 4\pi\xi f_0$$
$$K_i = \left(2\pi f_0\right)^2$$

Equation AMCLIB_TrackObsrv_Eq2

where ξ is the required attenuation and f_0 [Hz] is the required bandwidth of the control loop. Since the position error signal is calculated by the state filter formed in the rotating reference frame, the dynamics of the position tracking loop includes only frequencies proportional to the rate of change of the estimated rotor flux displacement. As demonstrated in Figure 79 , the position tracking observer consists of a PI controller and an integrator. The output of the continuous-time PI controller is defined by the following equation:

$$\omega_{estim}(t) = \theta_{err}(t)\cdot K_p + K_i\int_0^{\infty}\theta_{err}(t)\,dt$$

Equation AMCLIB_TrackObsrv_Eq3

and the output of the continuous-time integrator is as follows:

$$\theta_{estim}(t) = \int_0^{\infty}\omega_{estim}(t)\,dt$$

Equation AMCLIB_TrackObsrv_Eq4

where K_p is the proportional gain and K_i is the integral gain. Using the Laplace transform, equations AMCLIB_TrackObsrv_Eq3 and AMCLIB_TrackObsrv_Eq4 can be transformed into

$$\omega_{estim}(s) = K_p\cdot\theta_{err}(s) + \frac{1}{s}\cdot K_i\cdot\theta_{err}(s)$$
$$\theta_{estim}(s) = \frac{1}{s}\cdot\omega_{estim}(s)$$

Equation AMCLIB_TrackObsrv_Eq5

In order to implement the PI controller and integrator in a discrete digital control system, both blocks need to be transformed into the discrete time domain. Using the trapezoid (bilinear) discretization method, the equations AMCLIB_TrackObsrv_Eq5 are transformed into the following difference equations:

$$\omega_{estim}(k) = \omega_{estim}(k-1) + \theta_{err}(k)\cdot\left(K_p + \frac{K_i T_s}{2}\right) + \theta_{err}(k-1)\cdot\left(-K_p + \frac{K_i T_s}{2}\right)$$
$$\theta_{estim}(k) = \theta_{estim}(k-1) + \omega_{estim}(k)\cdot\frac{T_s}{2} + \omega_{estim}(k-1)\cdot\frac{T_s}{2}$$

Equation AMCLIB_TrackObsrv_Eq6

where:

- T_s is the sampling period [s]
- $\Theta_{err}(k)$ is the input phase error in the current step [rad]
- $\Theta_{err}(k-1)$ is the input phase error in the previous calculation step [rad]
- $\omega_{estim}(k)$ is the estimated angular velocity in the current step [rad/s]

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

- ω_{estim}(k-1) is the estimated angular velocity in the previous calculation step [rad/s].

Using the substitution

$$CC1 = K_p + \frac{K_i T_s}{2}$$
$$CC2 = -K_p + \frac{K_i T_s}{2}$$
$$C1 = \frac{T_s}{2}$$

Equation AMCLIB_TrackObsrv_Eq7

the equation AMCLIB_TrackObsrv_Eq6 can be rewritten into:

$$\omega_{estim}(k) = \omega_{estim}(k-1) + \theta_{err}(k) \cdot CC_1 + \theta_{err}(k-1) \cdot CC_2$$
$$\theta_{estim}(k) = \theta_{estim}(k-1) + \omega_{estim}(k) \cdot C_1 + \omega_{estim}(k-1) \cdot C_1$$

Equation AMCLIB_TrackObsrv_Eq8

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.11.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.11.3 Function AMCLIB_TrackObsrv_F32

2.11.3.1 Declaration

```
void AMCLIB_TrackObsrv_F32(tFrac32 f32PhaseErr, tFrac32 *pPosEst,
tFrac32 *pVelocityEst, AMCLIB_TRACK_OBSRV_T_F32 *pCtrl);
```

2.11.3.2 Arguments

Table 88. AMCLIB_TrackObsrv_F32 arguments

Type	Name	Direction	Description
tFrac32	f32PhaseErr	**input**	Input signal representing phase error of system to be estimated.
tFrac32 *	pPosEst	**output**	Estimated output position.
tFrac32 *	pVelocityEst	**output**	Estimated output velocity.
AMCLIB_TRACK_OBSRV_T_F32 *	pCtrl	**input, output**	Pointer to a tracking observer structure AMCLIB_TRACK_OBSRV_T_F32, which contains algorithm coefficients.

2.11.3.3 Implementation details

Prior to calculating the tracking observer parameters, it is necessary to set the scaling constants. All inputs and outputs of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

Table 89. Scaling constants

Scaling constant	Symbol	Calculation

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 89. Scaling constants...*continued*

Maximum angle [radians]	Θ^{MAX}	Must be equal to π to allow continuous integration
Maximum speed [rad/s]	Ω^{MAX}	Maximum application required speed, at least the motor electrical rated speed.

The tracking observer contains a PI controller (GFLIB_ControllerPIrAW_F32) and an integrator (GFLIB_IntegratorTR_F32). The proportional gain K_P and the integral gain K_I of the controller can be calculated according to the AMCLIB_TrackObsrv_eq2.

The continuous-time controller parameters K_P and K_I and the integrator constant are then discretized using the bilinear transform. The resulting difference equation coefficients *CC1*, *CC2*, and *C1* (see AMCLIB_TrackObsrv_eq7) must be scaled so that the input and output values of the controller and integrator fit in the fractional range.

$$CC1_{SC} = CC1 \cdot \frac{\theta^{MAX}}{\omega^{MAX}}$$
$$CC2_{SC} = CC2 \cdot \frac{\theta^{MAX}}{\omega^{MAX}}$$
$$C1_{SC} = C1 \cdot \frac{\omega^{MAX}}{\theta^{MAX}}$$

Equation AMCLIB_TrackObsrv_F32_Eq1

In case the coefficients $CC1_{SC}$ or $CC2_{SC}$ exceed the fractional range, their value must be right-shifted by *NShift*. Similarly, if $C1_{SC}$ exceeds the fractional range, its value must be right-shifted by *NIntegSh*. The binary shift amount corresponds to

$$NShift = \max\left(0, \left\lceil \log_2\left(\max\left(\left|CC1_{SC}\right|, \left|CC2_{SC}\right|\right)\right)\right\rceil\right)$$
$$NIntegSh = \max\left(0, \left\lceil \log_2\left(\left|C1_{SC}\right|\right)\right\rceil\right)$$

Equation AMCLIB_TrackObsrv_F32_Eq2

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 31st power of two (corresponds to the Q1.31 encoding):

$$pParamPI.f32CC1sc = \left(CC1_{SC} \cdot 2^{-NShift}\right) \cdot 2^{31}$$
$$pParamPI.f32CC2sc = \left(CC2_{SC} \cdot 2^{-NShift}\right) \cdot 2^{31}$$
$$pParamPI.u16NShift = NShift$$
$$pParamInteg.f32C1 = \left(C1_{SC} \cdot 2^{-NIntegSh}\right) \cdot 2^{31}$$
$$pParamInteg.u16NShift = NIntegSh$$

Equation AMCLIB_TrackObsrv_F32_Eq3

The output estimated angular velocity is limited according to the PI controller limits *pParamPI.f32UpperLimit* and *pParamPI.f32LowerLimit*.

The output estimated position Θ_{estim} wraps around to allow continuous integration of the rotor position. Make sure the Θ^{MAX} scaling constant is equal to π so that the fractional range corresponds to one complete rotor revolution.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The accuracy of the results is guaranteed for outputs *f32PosEstim* and *f32VelocityEstim* only if (pCtrl.pParamPI.u16NShift + pCtrlpParamInteg.u16NShift) < 15.

2.11.3.4 Code example

```
#define Wmax (2618.0F)
#define pi (3.1415927F)
#define Ts (1e-4F)
#define f0 (15.0F)
#define xi (0.707F)
#define w0 (2.0F*pi*f0)
#define Ki (w0*w0)
#define Kp (4.0F*pi*xi*f0)

#include "amclib.h"

AMCLIB_TRACK_OBSRV_T_F32 trMyTrObsrv;
tFrac32 f32PhaseErr;
tFrac32 f32PosEstim;
tFrac32 f32VelocityEstim;
tFrac32 f32PosOut;
tFrac32 f32VelocityOut;

void main (void)
{
    // controller parameters
    trMyTrObsrv.pParamPI.f32CC1sc        = FRAC32((Kp+(Ki*Ts)/2)*pi/Wmax);
    trMyTrObsrv.pParamPI.f32CC2sc        = FRAC32((-Kp+(Ki*Ts)/2)*pi/Wmax);
    trMyTrObsrv.pParamPI.f32UpperLimit   = FRAC32(1.0);
    trMyTrObsrv.pParamPI.f32LowerLimit   = FRAC32(-1.0);
    trMyTrObsrv.pParamPI.u16NShift       = (tU16)0;

    // Setting parameters for integrator
    trMyTrObsrv.pParamInteg.f32C1     = FRAC32((Ts/2)*Wmax/pi);
    trMyTrObsrv.pParamInteg.u16NShift = (tU16)0;

    // Setting of input phase error
    f32PhaseErr = FRAC32(0.25);

    // Initialization of observer internal states to zero:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInit_F32(&trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInit(&trMyTrObsrv, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_TrackObsrvInit(&trMyTrObsrv);

    // Use #AMCLIB_Windmilling_F32 to obtain the estimated rotor position
    // and velocity.
    // Estimated position: f32PosOut
    // Estimated velocity: f32VelocityOut

    // Initialization of observer internal states for the required output
    // position and velocity:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInitSetState_F32(f32PosOut, f32VelocityOut,
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
      &trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvSetState(f32PosOut, f32VelocityOut,
      &trMyTrObsrv, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_TrackObsrvSetState(f32PosOut, f32VelocityOut, &trMyTrObsrv);

    // Calculation of one iteration of the observer:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrv_F32(f32PhaseErr, &f32PosEstim, &f32VelocityEstim,
      &trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrv(f32PhaseErr, &f32PosEstim, &f32VelocityEstim, &trMyTrObsrv,
      F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_TrackObsrv(f32PhaseErr, &f32PosEstim, &f32VelocityEstim,
      &trMyTrObsrv);
}
```

2.11.4 Function AMCLIB_TrackObsrv_F16

2.11.4.1 Declaration

```
void AMCLIB_TrackObsrv_F16(tFrac16 f16PhaseErr, tFrac16 *pPosEst,
tFrac16 *pVelocityEst, AMCLIB_TRACK_OBSRV_T_F16 *pCtrl);
```

2.11.4.2 Arguments

Table 90. AMCLIB_TrackObsrv_F16 arguments

Type	Name	Direction	Description
tFrac16	f16PhaseErr	**input**	Input signal representing phase error of system to be estimated.
tFrac16 *	pPosEst	**output**	Estimated output position.
tFrac16 *	pVelocityEst	**output**	Estimated output velocity.
AMCLIB_TRACK_OBSRV_T_F16 *	pCtrl	**input, output**	Pointer to a tracking observer structure AMCLIB_TRACK_OBSRV_T_F16, which contains algorithm coefficients.

2.11.4.3 Implementation details

Prior to calculating the tracking observer parameters, it is necessary to set the scaling constants. All inputs and outputs of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

Table 91. Scaling constants

Scaling constant	Symbol	Calculation

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 91. Scaling constants...*continued*

Maximum angle [radians]	Θ^{MAX}	Must be equal to π to allow continuous integration
Maximum speed [rad/s]	Ω^{MAX}	Maximum application required speed, at least the motor electrical rated speed.

The tracking observer contains a PI controller (GFLIB_ControllerPIrAW_F16) and an integrator (GFLIB_IntegratorTR_F16). The proportional gain K_P and the integral gain K_I of the controller can be calculated according to the AMCLIB_TrackObsrv_eq2.

The continuous-time controller parameters K_P and K_I and the integrator constant are then discretized using the bilinear transform. The resulting difference equation coefficients *CC1*, *CC2*, and *C1* (see AMCLIB_TrackObsrv_eq7) must be scaled so that the input and output values of the controller and integrator fit in the fractional range.

$$CC1_{SC} = CC1 \cdot \frac{\theta^{MAX}}{\omega^{MAX}}$$
$$CC2_{SC} = CC2 \cdot \frac{\theta^{MAX}}{\omega^{MAX}}$$
$$C1_{SC} = C1 \cdot \frac{\omega^{MAX}}{\theta^{MAX}}$$

Equation AMCLIB_TrackObsrv_F16_Eq1

In case the coefficients $CC1_{SC}$ or $CC2_{SC}$ exceed the fractional range, their value must be right-shifted by *NShift*. Similarly, if $C1_{SC}$ exceeds the fractional range, its value must be right-shifted by *NIntegSh*. The binary shift amount corresponds to

$$NShift = \max\left(0, \left\lceil \log_2\left(\max\left(\left|CC1_{SC}\right|, \left|CC2_{SC}\right|\right)\right)\right\rceil\right)$$
$$NIntegSh = \max\left(0, \left\lceil \log_2\left(\left|C1_{SC}\right|\right)\right\rceil\right)$$

Equation AMCLIB_TrackObsrv_F16_Eq2

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 15th power of two (corresponds to the Q1.15 encoding):

$$pParamPI.f16CC1sc = \left(CC1_{SC} \cdot 2^{-NShift}\right) \cdot 2^{15}$$
$$pParamPI.f16CC2sc = \left(CC2_{SC} \cdot 2^{-NShift}\right) \cdot 2^{15}$$
$$pParamPI.u16NShift = NShift$$
$$pParamInteg.f16C1 = \left(C1_{SC} \cdot 2^{-NIntegSh}\right) \cdot 2^{15}$$
$$pParamInteg.u16NShift = NIntegSh$$

Equation AMCLIB_TrackObsrv_F16_Eq3

The output estimated angular velocity is limited according to the PI controller limits *pParamPI.f16UpperLimit* and *pParamPI.f16LowerLimit*.

The output estimated position Θ_{estim} wraps around to allow continuous integration of the rotor position. Make sure the Θ^{MAX} scaling constant is equal to π so that the fractional range corresponds to one complete rotor revolution.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The accuracy of f16VelocityEstim is guaranteed only if pCtrl.pParamPI.u16NShift <= 15. The accuracy of f16PosEstim is guaranteed if (pCtrl.pParamPI.u16NShift <= 13 and pCtrl.pParaminteg.u16NShift = 0) or (pCtrl.pParamPI.u16NShift = 0 and pCtrl.pParaminteg.u16NShift <= 1). In other cases the worst case error may rise above the guaranteed limits.

2.11.4.4 Code example

```
#define Wmax (2618.0F)
#define pi (3.1415927F)
#define Ts (1e-4F)
#define f0 (15.0F)
#define xi (0.707F)
#define w0 (2.0F*pi*f0)
#define Ki (w0*w0)
#define Kp (4.0F*pi*xi*f0)

#include "amclib.h"

AMCLIB_TRACK_OBSRV_T_F16 trMyTrObsrv;
tFrac16 f16PhaseErr;
tFrac16 f16PosEstim;
tFrac16 f16VelocityEstim;
tFrac16 f16PosOut;
tFrac16 f16VelocityOut;

void main (void)
{
    // controller parameters
    trMyTrObsrv.pParamPI.f16CC1sc        = FRAC16((Kp+(Ki*Ts)/2)*pi/Wmax);
    trMyTrObsrv.pParamPI.f16CC2sc        = FRAC16((-Kp+(Ki*Ts)/2)*pi/Wmax);
    trMyTrObsrv.pParamPI.f16UpperLimit   = FRAC16(1.0);
    trMyTrObsrv.pParamPI.f16LowerLimit   = FRAC16(-1.0);
    trMyTrObsrv.pParamPI.u16NShift       = (tU16)0;

    // Setting parameters for integrator
    trMyTrObsrv.pParamInteg.f16C1     = FRAC16((Ts/2)*Wmax/pi);
    trMyTrObsrv.pParamInteg.u16NShift = (tU16)0;

    // Setting of input phase error
    f16PhaseErr = FRAC16(0.25);

    // Initialization of observer internal states to zero:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInit_F16(&trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInit(&trMyTrObsrv, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_TrackObsrvInit(&trMyTrObsrv);

    // Use #AMCLIB_Windmilling_F16 to obtain the estimated rotor position
    // and velocity.
    // Estimated position: f16PosOut
    // Estimated velocity: f16VelocityOut

    // Initialization of observer internal states for the required output
    // position and velocity:
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInitSetState_F16(f16PosOut, f16VelocityOut,
      &trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvSetState(f16PosOut, f16VelocityOut,
      &trMyTrObsrv, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_TrackObsrvSetState(f16PosOut, f16VelocityOut, &trMyTrObsrv);

    // Calculation of one iteration of the observer:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrv_F16(f16PhaseErr, &f16PosEstim, &f16VelocityEstim,
      &trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrv(f16PhaseErr, &f16PosEstim, &f16VelocityEstim, &trMyTrObsrv,
      F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_TrackObsrv(f16PhaseErr, &f16PosEstim, &f16VelocityEstim,
      &trMyTrObsrv);
}
```

2.11.5 Function AMCLIB_TrackObsrv_FLT

2.11.5.1 Declaration

```
void AMCLIB_TrackObsrv_FLT(tFloat fltPhaseErr, tFloat *pPosEst,
tFloat *pVelocityEst, AMCLIB_TRACK_OBSRV_T_FLT *pCtrl);
```

2.11.5.2 Arguments

Table 92. AMCLIB_TrackObsrv_FLT arguments

Type	Name	Direction	Description
tFloat	fltPhaseErr	**input**	Input signal representing phase error of system to be estimated.
tFloat *	pPosEst	**output**	Estimated output position.
tFloat *	pVelocityEst	**output**	Estimated output velocity.
AMCLIB_TRACK_OBSRV_T_FLT *	pCtrl	**input, output**	Pointer to a tracking observer structure AMCLIB_TRACK_OBSRV_T_FLT, which contains algorithm coefficients.

2.11.5.3 Implementation details

The tracking observer contains a PI controller (GFLIB_ControllerPIrAW_FLT) and an integrator (GFLIB_IntegratorTR_FLT). The proportional gain K_P and the integral gain K_I of the controller can be calculated according to the AMCLIB_TrackObsrv_eq2.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The continuous-time controller parameters K_P and K_I and the integrator constant are then discretized using the bilinear transform. The resulting difference equation coefficients *CC1*, *CC2*, and *C1* (see AMCLIB_TrackObsrv_eq7) can be recorded into the parameters structure.

$$pParamPI.fltCC1sc = CC1$$
$$pParamPI.fltCC2sc = CC2$$
$$pParamInteg.fltC1 = C1$$

Equation AMCLIB_TrackObsrv_FLT_Eq1

The output estimated angular velocity is limited according to the PI controller limits *pParamPI.fltUpperLimit* and *pParamPI.fltLowerLimit*.

The output estimated position Θ_{estim} wraps around to allow continuous integration of the rotor position in the range [- π, π).

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.11.5.4 Code example:

```
#define pi (3.1415927f)
#define Ts (1e-4f)
#define f0 (15.0f)
#define xi (0.707f)
#define w0 (2.0f*pi*f0)
#define Ki (w0*w0)
#define Kp (4.0f*pi*xi*f0)

#include "amclib.h"

AMCLIB_TRACK_OBSRV_T_FLT trMyTrObsrv;
tFloat fltPhaseErr;
tFloat fltPosEstim;
tFloat fltVelocityEstim;
tFloat fltPosOut;
tFloat fltVelocityOut;

void main (void)
{
    // controller parameters
    trMyTrObsrv.pParamPI.fltCC1sc = Kp+(Ki*Ts)/2.0f;
    trMyTrObsrv.pParamPI.fltCC2sc = -Kp+(Ki*Ts)/2.0f;
    trMyTrObsrv.pParamPI.fltUpperLimit = 1000.0f;
    trMyTrObsrv.pParamPI.fltLowerLimit = -1000.0f;

    // Setting parameters for integrator
    trMyTrObsrv.pParamInteg.fltC1 = Ts/2.0f;

    // example input phase error
    fltPhaseErr = 0.25f;

    // Initialization of observer internal states to zero:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInit_FLT(&trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    AMCLIB_TrackObsrvInit(&trMyTrObsrv, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    AMCLIB_TrackObsrvInit(&trMyTrObsrv);

    // Use #AMCLIB_Windmilling_FLT to obtain the estimated rotor position
    // and velocity.
    // Estimated position: fltPosOut
    // Estimated velocity: fltVelocityOut

    // Initialization of observer internal states for the required output
    // position and velocity:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInitSetState_FLT(fltPosOut, fltVelocityOut,
      &trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvSetState(fltPosOut, fltVelocityOut,
      &trMyTrObsrv, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    AMCLIB_TrackObsrvSetState(fltPosOut, fltVelocityOut, &trMyTrObsrv);

    // Calculation of one iteration of the observer:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrv_FLT(fltPhaseErr, &fltPosEstim, &fltVelocityEstim,
      &trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrv(fltPhaseErr, &fltPosEstim, &fltVelocityEstim, &trMyTrObsrv,
      FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    AMCLIB_TrackObsrv(fltPhaseErr, &fltPosEstim, &fltVelocityEstim,
      &trMyTrObsrv);
}
```

2.11.6 Function AMCLIB_TrackObsrvInit

2.11.6.1 Description

This function initializes all internal states of the tracking observer to zero.

Note: *The input/output pointers must contain valid addresses otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).*

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.11.6.2 Re-entrancy:

The function is re-entrant for a different `pCtrl`.

2.11.6.3 Function AMCLIB_TrackObsrvInit_F32

2.11.6.3.1 Declaration

```
void AMCLIB_TrackObsrvInit_F32(AMCLIB_TRACK_OBSRV_T_F32 *pCtrl);
```

2.11.6.3.2 Arguments

Table 93. AMCLIB_TrackObsrvInit_F32 arguments

Type	Name	Direction	Description
AMCLIB_TRACK_OBSRV_T_F32 *	pCtrl	**input, output**	Pointer to a tracking observer structure AMCLIB_TRACK_OBSRV_T_F32, which contains algorithm coefficients.

2.11.6.3.3 Code example

```
#define Wmax (2618.0F)
#define pi (3.1415927F)
#define Ts (1e-4F)
#define f0 (15.0F)
#define xi (0.707F)
#define w0 (2.0F*pi*f0)
#define Ki (w0*w0)
#define Kp (4.0F*pi*xi*f0)

#include "amclib.h"

AMCLIB_TRACK_OBSRV_T_F32 trMyTrObsrv;
tFrac32 f32PhaseErr;
tFrac32 f32PosEstim;
tFrac32 f32VelocityEstim;
tFrac32 f32PosOut;
tFrac32 f32VelocityOut;

void main (void)
{
    // controller parameters
    trMyTrObsrv.pParamPI.f32CC1sc        = FRAC32((Kp+(Ki*Ts)/2)*pi/Wmax);
    trMyTrObsrv.pParamPI.f32CC2sc        = FRAC32((-Kp+(Ki*Ts)/2)*pi/Wmax);
    trMyTrObsrv.pParamPI.f32UpperLimit   = FRAC32(1.0);
    trMyTrObsrv.pParamPI.f32LowerLimit   = FRAC32(-1.0);
    trMyTrObsrv.pParamPI.u16NShift       = (tU16)0;

    // Setting parameters for integrator
    trMyTrObsrv.pParamInteg.f32C1     = FRAC32((Ts/2)*Wmax/pi);
    trMyTrObsrv.pParamInteg.u16NShift = (tU16)0;

    // Setting of input phase error
    f32PhaseErr = FRAC32(0.25);

    // Initialization of observer internal states to zero:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInit_F32(&trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    AMCLIB_TrackObsrvInit(&trMyTrObsrv, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_TrackObsrvInit(&trMyTrObsrv);

    // Use #AMCLIB_Windmilling_F32 to obtain the estimated rotor position
    // and velocity.
    // Estimated position: f32PosOut
    // Estimated velocity: f32VelocityOut

    // Initialization of observer internal states for the required output
    // position and velocity:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInitSetState_F32(f32PosOut, f32VelocityOut,
      &trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvSetState(f32PosOut, f32VelocityOut,
      &trMyTrObsrv, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_TrackObsrvSetState(f32PosOut, f32VelocityOut, &trMyTrObsrv);

    // Calculation of one iteration of the observer:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrv_F32(f32PhaseErr, &f32PosEstim, &f32VelocityEstim,
      &trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrv(f32PhaseErr, &f32PosEstim, &f32VelocityEstim, &trMyTrObsrv,
      F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_TrackObsrv(f32PhaseErr, &f32PosEstim, &f32VelocityEstim,
      &trMyTrObsrv);
}
```

2.11.6.4 Function AMCLIB_TrackObsrvInit_F16

2.11.6.4.1 Declaration

```
void AMCLIB_TrackObsrvInit_F16(AMCLIB_TRACK_OBSRV_T_F16 *pCtrl);
```

2.11.6.4.2 Arguments

Table 94. AMCLIB_TrackObsrvInit_F16 arguments

Type	Name	Direction	Description
AMCLIB_TRACK_OBSRV_T_F16 *	pCtrl	**input, output**	Pointer to a tracking observer structure AMCLIB_TRACK_OBSRV_T_F16, which contains algorithm coefficients.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.11.6.4.3 Code example

```
#define Wmax (2618.0F)
#define pi (3.1415927F)
#define Ts (1e-4F)
#define f0 (15.0F)
#define xi (0.707F)
#define w0 (2.0F*pi*f0)
#define Ki (w0*w0)
#define Kp (4.0F*pi*xi*f0)

#include "amclib.h"

AMCLIB_TRACK_OBSRV_T_F16 trMyTrObsrv;
tFrac16 f16PhaseErr;
tFrac16 f16PosEstim;
tFrac16 f16VelocityEstim;
tFrac16 f16PosOut;
tFrac16 f16VelocityOut;

void main (void)
{
    // controller parameters
    trMyTrObsrv.pParamPI.f16CC1sc        = FRAC16((Kp+(Ki*Ts)/2)*pi/Wmax);
    trMyTrObsrv.pParamPI.f16CC2sc        = FRAC16((-Kp+(Ki*Ts)/2)*pi/Wmax);
    trMyTrObsrv.pParamPI.f16UpperLimit   = FRAC16(1.0);
    trMyTrObsrv.pParamPI.f16LowerLimit   = FRAC16(-1.0);
    trMyTrObsrv.pParamPI.u16NShift       = (tU16)0;

    // Setting parameters for integrator
    trMyTrObsrv.pParamInteg.f16C1     = FRAC16((Ts/2)*Wmax/pi);
    trMyTrObsrv.pParamInteg.u16NShift = (tU16)0;

    // Setting of input phase error
    f16PhaseErr = FRAC16(0.25);

    // Initialization of observer internal states to zero:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInit_F16(&trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInit(&trMyTrObsrv, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_TrackObsrvInit(&trMyTrObsrv);

    // Use #AMCLIB_Windmilling_F16 to obtain the estimated rotor position
    // and velocity.
    // Estimated position: f16PosOut
    // Estimated velocity: f16VelocityOut

    // Initialization of observer internal states for the required output
    // position and velocity:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInitSetState_F16(f16PosOut, f16VelocityOut,
      &trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    AMCLIB_TrackObsrvSetState(f16PosOut, f16VelocityOut,
      &trMyTrObsrv, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_TrackObsrvSetState(f16PosOut, f16VelocityOut, &trMyTrObsrv);

    // Calculation of one iteration of the observer:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrv_F16(f16PhaseErr, &f16PosEstim, &f16VelocityEstim,
      &trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrv(f16PhaseErr, &f16PosEstim, &f16VelocityEstim, &trMyTrObsrv,
      F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_TrackObsrv(f16PhaseErr, &f16PosEstim, &f16VelocityEstim,
      &trMyTrObsrv);
}
```

2.11.6.5 Function AMCLIB_TrackObsrvInit_FLT

2.11.6.5.1 Declaration

```
void AMCLIB_TrackObsrvInit_FLT(AMCLIB_TRACK_OBSRV_T_FLT *pCtrl);
```

2.11.6.5.2 Arguments

Table 95. AMCLIB_TrackObsrvInit_FLT arguments

Type	Name	Direction	Description
AMCLIB_TRACK_OBSRV_T_FLT *	pCtrl	**input, output**	Pointer to a tracking observer structure AMCLIB_TRACK_OBSRV_T_FLT, which contains algorithm coefficients.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.11.6.5.3 Code example:

```
#define pi (3.1415927F)
#define Ts (1e-4F)
#define f0 (15.0F)
#define xi (0.707F)
#define w0 (2.0F*pi*f0)
#define Ki (w0*w0)
#define Kp (4.0F*pi*xi*f0)

#include "amclib.h"

AMCLIB_TRACK_OBSRV_T_FLT trMyTrObsrv;
tFloat fltPhaseErr;
tFloat fltPosEstim;
tFloat fltVelocityEstim;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
tFloat fltPosOut;
tFloat fltVelocityOut;

void main (void)
{
    // controller parameters
    trMyTrObsrv.pParamPI.fltCC1sc        = (tFloat)(Kp+(Ki*Ts)/2);
    trMyTrObsrv.pParamPI.fltCC2sc        = (tFloat)(-Kp+(Ki*Ts)/2);
    trMyTrObsrv.pParamPI.fltUpperLimit   = (tFloat)(1.0);
    trMyTrObsrv.pParamPI.fltLowerLimit   = (tFloat)(-1.0);

    // Setting parameters for integrator
    trMyTrObsrv.pParamInteg.fltC1     = (tFloat)(Ts/2);

    // Setting of input phase error
    fltPhaseErr = (tFloat)(0.25);

    // Initialization of observer internal states to zero:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInit_FLT(&trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInit(&trMyTrObsrv, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    AMCLIB_TrackObsrvInit(&trMyTrObsrv);

    // Use #AMCLIB_Windmilling_FLT to obtain the estimated rotor position
    // and velocity.
    // Estimated position: fltPosOut
    // Estimated velocity: fltVelocityOut

    // Initialization of observer internal states for the required output
    // position and velocity:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInitSetState_FLT(fltPosOut, fltVelocityOut,
      &trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvSetState(fltPosOut, fltVelocityOut,
      &trMyTrObsrv, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    AMCLIB_TrackObsrvSetState(fltPosOut, fltVelocityOut, &trMyTrObsrv);

    // Calculation of one iteration of the observer:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrv_FLT(fltPhaseErr, &fltPosEstim, &fltVelocityEstim,
      &trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrv(fltPhaseErr, &fltPosEstim, &fltVelocityEstim, &trMyTrObsrv,
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
      FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    AMCLIB_TrackObsrv(fltPhaseErr, &fltPosEstim, &fltVelocityEstim,
      &trMyTrObsrv);
}
```

2.11.7 Function AMCLIB_TrackObsrvSetState

2.11.7.1 Description

This function sets the tracking observer internal states to attain the required angular velocity and position outputs. Setting the required values to the actual estimated position and velocity of the rotor enables seamless transition from an uncontrolled rotation of the rotor into a controlled state. A robust set of initial estimates can be obtained from function AMCLIB_Windmilling.

Note: *The observer parameters must be already set in the state structure pointed to by `pCtrl` before this function can be called.*

Note: *The input/output pointers must contain valid addresses otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).*

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.11.7.2 Re-entrancy:

The function is re-entrant for a different `pCtrl`.

2.11.7.3 Function AMCLIB_TrackObsrvSetState_F32

2.11.7.3.1 Declaration

```
void AMCLIB_TrackObsrvSetState_F32(tFrac32 f32PosOut, tFrac32
f32VelocityOut, AMCLIB_TRACK_OBSRV_T_F32 *pCtrl);
```

2.11.7.3.2 Arguments

Table 96. AMCLIB_TrackObsrvSetState_F32 arguments

Type	Name	Direction	Description
tFrac32	f32PosOut	**input**	Required output of the GFLIB_IntegratorTR, i.e., what rotor position shall be outputted in the next iteration.
tFrac32	f32VelocityOut	**input**	Required output of the GFLIB_ControllerPIrAW, i.e., what velocity shall be outputted in the next iteration.
AMCLIB_TRACK_OBSRV_T_F32 *	pCtrl	**input, output**	Pointer to a tracking observer structure AMCLIB_TRACK_OBSRV_T_F32, which contains algorithm coefficients.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.11.7.3.3 Implementation details

The scaling of the inputs f32PosOut and f32VelocityOut follows the same rules as the corresponding outputs of AMCLIB_TrackObsrv_F32.

2.11.7.3.4 Code example

```
#define Wmax (2618.0F)
#define pi (3.1415927F)
#define Ts (1e-4F)
#define f0 (15.0F)
#define xi (0.707F)
#define w0 (2.0F*pi*f0)
#define Ki (w0*w0)
#define Kp (4.0F*pi*xi*f0)

#include "amclib.h"

AMCLIB_TRACK_OBSRV_T_F32 trMyTrObsrv;
tFrac32 f32PhaseErr;
tFrac32 f32PosEstim;
tFrac32 f32VelocityEstim;
tFrac32 f32PosOut;
tFrac32 f32VelocityOut;

void main (void)
{
    // controller parameters
    trMyTrObsrv.pParamPI.f32CC1sc        = FRAC32((Kp+(Ki*Ts)/2)*pi/Wmax);
    trMyTrObsrv.pParamPI.f32CC2sc        = FRAC32((-Kp+(Ki*Ts)/2)*pi/Wmax);
    trMyTrObsrv.pParamPI.f32UpperLimit   = FRAC32(1.0);
    trMyTrObsrv.pParamPI.f32LowerLimit   = FRAC32(-1.0);
    trMyTrObsrv.pParamPI.u16NShift       = (tU16)0;

    // Setting parameters for integrator
    trMyTrObsrv.pParamInteg.f32C1     = FRAC32((Ts/2)*Wmax/pi);
    trMyTrObsrv.pParamInteg.u16NShift = (tU16)0;

    // Setting of input phase error
    f32PhaseErr = FRAC32(0.25);

    // Initialization of observer internal states to zero:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInit_F32(&trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInit(&trMyTrObsrv, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_TrackObsrvInit(&trMyTrObsrv);

    // Use #AMCLIB_Windmilling_F32 to obtain the estimated rotor position
    // and velocity.
    // Estimated position: f32PosOut
    // Estimated velocity: f32VelocityOut

    // Initialization of observer internal states for the required output
    // position and velocity:

    // Alternative 1: API call with postfix
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInitSetState_F32(f32PosOut, f32VelocityOut,
      &trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvSetState(f32PosOut, f32VelocityOut,
      &trMyTrObsrv, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_TrackObsrvSetState(f32PosOut, f32VelocityOut, &trMyTrObsrv);

    // Calculation of one iteration of the observer:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrv_F32(f32PhaseErr, &f32PosEstim, &f32VelocityEstim,
      &trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrv(f32PhaseErr, &f32PosEstim, &f32VelocityEstim, &trMyTrObsrv,
      F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_TrackObsrv(f32PhaseErr, &f32PosEstim, &f32VelocityEstim,
      &trMyTrObsrv);
}
```

2.11.7.4 Function AMCLIB_TrackObsrvSetState_F16

2.11.7.4.1 Declaration

```
void AMCLIB_TrackObsrvSetState_F16(tFrac16 f16PosOut, tFrac16
f16VelocityOut, AMCLIB_TRACK_OBSRV_T_F16 *pCtrl);
```

2.11.7.4.2 Arguments

Table 97. AMCLIB_TrackObsrvSetState_F16 arguments

Type	Name	Direction	Description
tFrac16	f16PosOut	**input**	Required output of the GFLIB_IntegratorTR, i.e., what rotor position shall be outputted in the next iteration.
tFrac16	f16VelocityOut	**input**	Required output of the GFLIB_ControllerPIrAW, i.e., what velocity shall be outputted in the next iteration.
AMCLIB_TRACK_OBSRV_T_F16 *	pCtrl	**input, output**	Pointer to a tracking observer structure AMCLIB_TRACK_OBSRV_T_F16, which contains algorithm coefficients.

2.11.7.4.3 Implementation details

The scaling of the inputs f16PosOut and f16VelocityOut follows the same rules as the corresponding outputs of AMCLIB_TrackObsrv_F16.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.11.7.4.4 Code example

```
#define Wmax (2618.0F)
#define pi (3.1415927F)
#define Ts (1e-4F)
#define f0 (15.0F)
#define xi (0.707F)
#define w0 (2.0F*pi*f0)
#define Ki (w0*w0)
#define Kp (4.0F*pi*xi*f0)

#include "amclib.h"

AMCLIB_TRACK_OBSRV_T_F16 trMyTrObsrv;
tFrac16 f16PhaseErr;
tFrac16 f16PosEstim;
tFrac16 f16VelocityEstim;
tFrac16 f16PosOut;
tFrac16 f16VelocityOut;

void main (void)
{
    // controller parameters
    trMyTrObsrv.pParamPI.f16CC1sc        = FRAC16((Kp+(Ki*Ts)/2)*pi/Wmax);
    trMyTrObsrv.pParamPI.f16CC2sc        = FRAC16((-Kp+(Ki*Ts)/2)*pi/Wmax);
    trMyTrObsrv.pParamPI.f16UpperLimit   = FRAC16(1.0);
    trMyTrObsrv.pParamPI.f16LowerLimit   = FRAC16(-1.0);
    trMyTrObsrv.pParamPI.u16NShift       = (tU16)0;

    // Setting parameters for integrator
    trMyTrObsrv.pParamInteg.f16C1     = FRAC16((Ts/2)*Wmax/pi);
    trMyTrObsrv.pParamInteg.u16NShift = (tU16)0;

    // Setting of input phase error
    f16PhaseErr = FRAC16(0.25);

    // Initialization of observer internal states to zero:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInit_F16(&trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInit(&trMyTrObsrv, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_TrackObsrvInit(&trMyTrObsrv);

    // Use #AMCLIB_Windmilling_F16 to obtain the estimated rotor position
    // and velocity.
    // Estimated position: f16PosOut
    // Estimated velocity: f16VelocityOut

    // Initialization of observer internal states for the required output
    // position and velocity:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInitSetState_F16(f16PosOut, f16VelocityOut,
      &trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    AMCLIB_TrackObsrvSetState(f16PosOut, f16VelocityOut,
      &trMyTrObsrv, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_TrackObsrvSetState(f16PosOut, f16VelocityOut, &trMyTrObsrv);

    // Calculation of one iteration of the observer:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrv_F16(f16PhaseErr, &f16PosEstim, &f16VelocityEstim,
      &trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrv(f16PhaseErr, &f16PosEstim, &f16VelocityEstim, &trMyTrObsrv,
      F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_TrackObsrv(f16PhaseErr, &f16PosEstim, &f16VelocityEstim,
      &trMyTrObsrv);
}
```

2.11.7.5 Function AMCLIB_TrackObsrvSetState_FLT

2.11.7.5.1 Declaration

```
void AMCLIB_TrackObsrvSetState_FLT(tFloat fltPosOut, tFloat
fltVelocityOut, AMCLIB_TRACK_OBSRV_T_FLT *pCtrl);
```

2.11.7.5.2 Arguments

Table 98. AMCLIB_TrackObsrvSetState_FLT arguments

Type	Name	Direction	Description
tFloat	fltPosOut	**input**	Required output of the GFLIB_IntegratorTR, i.e., what rotor position shall be outputted in the next iteration. The value must be in the range [-π, π].
tFloat	fltVelocityOut	**input**	Required output of the GFLIB_ControllerPIrAW, i.e., what velocity shall be outputted in the next iteration.
AMCLIB_TRACK_OBSRV_T_FLT *	pCtrl	**input, output**	Pointer to a tracking observer structure AMCLIB_TRACK_OBSRV_T_FLT, which contains algorithm coefficients.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.11.7.5.3 Code example:

```
#define pi (3.1415927F)
#define Ts (1e-4F)
#define f0 (15.0F)
#define xi (0.707F)
#define w0 (2.0F*pi*f0)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
#define Ki (w0*w0)
#define Kp (4.0F*pi*xi*f0)

#include "amclib.h"

AMCLIB_TRACK_OBSRV_T_FLT trMyTrObsrv;
tFloat fltPhaseErr;
tFloat fltPosEstim;
tFloat fltVelocityEstim;
tFloat fltPosOut;
tFloat fltVelocityOut;

void main (void)
{
    // controller parameters
    trMyTrObsrv.pParamPI.fltCC1sc        = (tFloat)(Kp+(Ki*Ts)/2);
    trMyTrObsrv.pParamPI.fltCC2sc        = (tFloat)(-Kp+(Ki*Ts)/2);
    trMyTrObsrv.pParamPI.fltUpperLimit   = (tFloat)(1.0);
    trMyTrObsrv.pParamPI.fltLowerLimit   = (tFloat)(-1.0);

    // Setting parameters for integrator
    trMyTrObsrv.pParamInteg.fltC1     = (tFloat)(Ts/2);

    // Setting of input phase error
    fltPhaseErr = (tFloat)(0.25);

    // Initialization of observer internal states to zero:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInit_FLT(&trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInit(&trMyTrObsrv, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    AMCLIB_TrackObsrvInit(&trMyTrObsrv);

    // Use #AMCLIB_Windmilling_FLT to obtain the estimated rotor position
    // and velocity.
    // Estimated position: fltPosOut
    // Estimated velocity: fltVelocityOut

    // Initialization of observer internal states for the required output
    // position and velocity:

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvInitSetState_FLT(fltPosOut, fltVelocityOut,
      &trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrvSetState(fltPosOut, fltVelocityOut,
      &trMyTrObsrv, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    AMCLIB_TrackObsrvSetState(fltPosOut, fltVelocityOut, &trMyTrObsrv);

    // Calculation of one iteration of the observer:
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_TrackObsrv_FLT(fltPhaseErr, &fltPosEstim, &fltVelocityEstim,
      &trMyTrObsrv);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_TrackObsrv(fltPhaseErr, &fltPosEstim, &fltVelocityEstim, &trMyTrObsrv,
      FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    AMCLIB_TrackObsrv(fltPhaseErr, &fltPosEstim, &fltVelocityEstim,
      &trMyTrObsrv);
}
```

2.12 Function AMCLIB_Windmilling

This function detects spontaneous rotation of an unpowered 3-phase permanent magnet synchronous motor by measuring the generated back electromotive force.

2.12.1 Description

This function detects spontaneous rotation (windmilling) of an unpowered 3-phase permanent magnet synchronous motor (PMSM) by measuring the generated back electromotive force (BEMF). Assuming that the motor is connected to a 3-phase bridge power stage, it is necessary to add three voltage measuring points (u_a, u_b, u_c) as indicated in the following figure.

Figure 81. Power stage with BEMF measurement points

The 3-phase voltages can be measured by single-ended A/D converters. The patent-pending algorithm employed in AMCLIB_Windmilling can accept measurements distorted

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

by variable DC offset, limitation, noise, and certain higher harmonics. The function will decide whether the motor is spinning or not and calculate the estimated position and angular velocity of the rotor. The following figure shows the internal structure of AMCLIB_Windmilling.

Figure 82. AMCLIB_Windmilling block diagram

The Beta-projection block is also available as a stand-alone function (see GMCLIB_BetaProjection). AMCLIB_Windmilling internally calls the AMCLIB_TrackObsrv function to realize the phase-locked loop (PLL) feedback structure with a PI controller and an integrator. Refer to the tracking observer chapter in this manual to learn how to set the parameters of the PI controller and the integrator. AMCLIB_Windmilling provides an automatic detector of the PLL lock, which is used as an indicator whether the rotor is spinning or not. The PLL takes a while to achieve a lock, so the function will return **UNDECIDED** for the first several iterations and then either **SPINNING** or **STOPPED**, depending on the actual state. It is necessary to call the function AMCLIB_WindmillingInit before the first use of AMCLIB_Windmilling to initialize the state variables.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.12.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.12.3 Function AMCLIB_Windmilling_F32

2.12.3.1 Declaration

```
AMCLIB_WINDMILLING_RET_T AMCLIB_Windmilling_F32(const
SWLIBS_3Syst_F32 *pUabcIn, tFrac32 *pPosEst, tFrac32
*pVelocityEst, AMCLIB_WINDMILLING_T_F32 *const pCtrl);
```

2.12.3.2 Arguments

Table 99. AMCLIB_Windmilling_F32 arguments

Type	Name	Direction	Description
const SWLIBS_3Syst_F32 *	pUabcIn	**input**	Pointer to the structure with the measured 3-phase A/B/C voltages.
tFrac32 *	pPosEst	**output**	Estimated rotor flux position in the interval <-1; 1) which corresponds to the angle range <-π; π) radians.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 99. AMCLIB_Windmilling_F32 arguments...*continued*

Type	Name	Direction	Description
tFrac32 *	pVelocityEst	**output**	Estimated electrical angular velocity of the rotor. The velocity estimates are noisy, especially for low speeds; therefore, it is recommended to use a GDFLIB_FilterMA_F32 to further filter the results. The filter should be engaged only after the AMCLIB_Windmilling has returned **SPIN NING**; use GDFLIB_FilterMASetState_F32 to initialize the filter state with the first estimated velocity value.
AMCLIB_ WINDMILLING_ T_F32 *const	pCtrl	**input, output**	Pointer to the structure with AMCLIB_ Windmilling parameters and state variables.

2.12.3.3 Return

The function returns **UNDECIDED** if it is not yet clear whether the rotor is spinning or not. The AMCLIB_Windmilling must then be called again in the next sampling period. The function returns **SPINNING** if the rotor is spinning. If the motor spin is not detected within 2000 iterations, the function will return **STOPPED**.

2.12.3.4 Implementation details

This function internally calls the Tracking Observer function which calculates the PI controller and the integrator subblocks. Refer to the documentation of function AMCLIB_TrackObsrv_F32 (and its parent chapter describing the AMCLIB_TrackObsrv functionality) for a description of the necessary scaling and setup of the parameters of the PI controller and integrator blocks used by the AMCLIB_Windmilling_F32. It is necessary to call the AMCLIB_WindmillingInit_F32 before the first use of AMCLIB_Windmilling_F32 to initialize the state variables.

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

2.12.3.5 Code Example

```
#define U_MAX 12
#define ADC_MAX_OFFSET_ERROR FRAC32(0.002/U_MAX)
#define Wmax (2618.0F)
#define pi (3.1415927F)
#define Ts (1e-4F)
#define f0 (30.0F)
#define xi (0.707F)
#define w0 (2.0F*pi*f0)
#define Ki (w0*w0)
#define Kp (4.0F*pi*xi*f0)

#include "amclib.h"

SWLIBS_3Syst_F32              f32Uabc;
AMCLIB_WINDMILLING_T_F32      windStruct;
tFrac32                       f32Position;
tFrac32                       f32Velocity;
AMCLIB_WINDMILLING_RET_T      retVal = UNDECIDED;

void main (void)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
{
    // Controller parameters
    windStruct.pParamATO.pParamPI.f32CC1sc = FRAC32((Kp+(Ki*Ts)/2)*pi/Wmax);
    windStruct.pParamATO.pParamPI.f32CC2sc = FRAC32((-Kp+(Ki*Ts)/2)*pi/Wmax);
    windStruct.pParamATO.pParamPI.f32UpperLimit = FRAC32(1.0);
    windStruct.pParamATO.pParamPI.f32LowerLimit = FRAC32(-1.0);
    windStruct.pParamATO.pParamPI.u16NShift = (tU16)0;
    // Integrator parameters
    windStruct.pParamATO.pParamInteg.f32C1 = FRAC32((Ts/2)*Wmax/pi);
    windStruct.pParamATO.pParamInteg.u16NShift = (tU16)0;

    // Initialize AMCLIB_Windmilling
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_WindmillingInit_F32(ADC_MAX_OFFSET_ERROR, &windStruct);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_WindmillingInit(ADC_MAX_OFFSET_ERROR, &windStruct, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_WindmillingInit(ADC_MAX_OFFSET_ERROR, &windStruct);

    while(1);
}

// Periodical function or interrupt
void ISR(void)
{
    // Measure the 3-phase voltages f32Uabc
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    retVal = AMCLIB_Windmilling_F32(&f32Uabc, &f32Position,
                                    &f32Velocity, &windStruct);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    retVal = AMCLIB_Windmilling(&f32Uabc, &f32Position,
                                &f32Velocity, &windStruct, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    retVal = AMCLIB_Windmilling(&f32Uabc, &f32Position,
                                &f32Velocity, &windStruct);
}
```

2.12.4 Function AMCLIB_Windmilling_F16

2.12.4.1 Declaration

```
AMCLIB_WINDMILLING_RET_T AMCLIB_Windmilling_F16(const
SWLIBS_3Syst_F16 *pUabcIn, tFrac16 *pPosEst, tFrac16
*pVelocityEst, AMCLIB_WINDMILLING_T_F16 *const pCtrl);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.12.4.2 Arguments

Table 100. AMCLIB_Windmilling_F16 arguments

Type	Name	Direction	Description
const SWLIBS_3Syst_F16 *	pUabcIn	**input**	Pointer to the structure with the measured 3-phase A/B/C voltages.
tFrac16 *	pPosEst	**output**	Estimated rotor flux position in the interval <-1; 1) which corresponds to the angle range <-π; π) radians.
tFrac16 *	pVelocityEst	**output**	Estimated electrical angular velocity of the rotor. The velocity estimates are noisy, especially for low speeds; therefore, it is recommended to use a GDFLIB_FilterMA_F16 to further filter the results. The filter should be engaged only after the AMCLIB_Windmilling has returned **SPINNING**; use GDFLIB_FilterMASetState_F16 to initialize the filter state with the first estimated velocity value.
AMCLIB_WINDMILLING_T_F16 *const	pCtrl	**input, output**	Pointer to the structure with AMCLIB_Windmilling parameters and state variables.

2.12.4.3 Return

The function returns **UNDECIDED** if it is not yet clear whether the rotor is spinning or not. The AMCLIB_Windmilling must then be called again in the next sampling period. The function returns **SPINNING** if the rotor is spinning. If the motor spin is not detected within 2000 iterations, the function will return **STOPPED**.

2.12.4.4 Implementation details

This function internally calls the Tracking Observer function which calculates the PI controller and the integrator subblocks. Refer to the documentation of function AMCLIB_TrackObsrv_F16 (and its parent chapter describing the AMCLIB_TrackObsrv functionality) for a description of the necessary scaling and setup of the parameters of the PI controller and integrator blocks used by the AMCLIB_Windmilling_F16. It is necessary to call the AMCLIB_WindmillingInit_F16 before the first use of AMCLIB_Windmilling_F16 to initialize the state variables.

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

2.12.4.5 Code Example

```
#define U_MAX 12
#define ADC_MAX_OFFSET_ERROR FRAC16(0.002/U_MAX)
#define Wmax (2618.0F)
#define pi (3.1415927F)
#define Ts (1e-4F)
#define f0 (30.0F)
#define xi (0.707F)
#define w0 (2.0F*pi*f0)
#define Ki (w0*w0)
#define Kp (4.0F*pi*xi*f0)

#include "amclib.h"
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
SWLIBS_3Syst_F16                f16Uabc;
AMCLIB_WINDMILLING_T_F16        windStruct;
tFrac16                         f16Position;
tFrac16                         f16Velocity;
AMCLIB_WINDMILLING_RET_T        retVal = UNDECIDED;

void main (void)
{
    // Controller parameters
    windStruct.pParamATO.pParamPI.f16CC1sc = FRAC16((Kp+(Ki*Ts)/2)*pi/Wmax);
    windStruct.pParamATO.pParamPI.f16CC2sc = FRAC16((-Kp+(Ki*Ts)/2)*pi/Wmax);
    windStruct.pParamATO.pParamPI.f16UpperLimit = FRAC16(1.0);
    windStruct.pParamATO.pParamPI.f16LowerLimit = FRAC16(-1.0);
    windStruct.pParamATO.pParamPI.u16NShift = (tU16)0;
    // Integrator parameters
    windStruct.pParamATO.pParamInteg.f16C1 = FRAC16((Ts/2)*Wmax/pi);
    windStruct.pParamATO.pParamInteg.u16NShift = (tU16)0;

    // Initialize AMCLIB_Windmilling
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_WindmillingInit_F16(ADC_MAX_OFFSET_ERROR, &windStruct);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_WindmillingInit(ADC_MAX_OFFSET_ERROR, &windStruct, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_WindmillingInit(ADC_MAX_OFFSET_ERROR, &windStruct);

    while(1);
}

// Periodical function or interrupt
void ISR(void)
{
    // Measure the 3-phase voltages f16Uabc
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    retVal = AMCLIB_Windmilling_F16(&f16Uabc, &f16Position,
                                    &f16Velocity, &windStruct);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    retVal = AMCLIB_Windmilling(&f16Uabc, &f16Position,
                                &f16Velocity, &windStruct, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    retVal = AMCLIB_Windmilling(&f16Uabc, &f16Position,
                                &f16Velocity, &windStruct);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.12.5 Function AMCLIB_Windmilling_FLT

2.12.5.1 Declaration

```
AMCLIB_WINDMILLING_RET_T AMCLIB_Windmilling_FLT(const
SWLIBS_3Syst_FLT *pUabcIn, tFloat *pPosEst, tFloat *pVelocityEst,
AMCLIB_WINDMILLING_T_FLT *const pCtrl);
```

2.12.5.2 Arguments

Table 101. AMCLIB_Windmilling_FLT arguments

Type	Name	Direction	Description
const SWLIBS_3Syst_FLT *	pUabcIn	**input**	Pointer to the structure with the measured 3-phase A/B/C voltages.
tFloat *	pPosEst	**output**	Estimated rotor flux position in the interval <-π; π) radians.
tFloat *	pVelocityEst	**output**	Estimated electrical angular velocity of the rotor. The velocity estimates are noisy, especially for low speeds; therefore, it is recommended to use a GDFLIB_FilterMA_FLT to further filter the results. The filter should be engaged only after the AMCLIB_Windmilling has returned **SPINNING**; use GDFLIB_FilterMASetState_FLT to initialize the filter state with the first estimated velocity value.
AMCLIB_WINDMILLING_T_FLT *const	pCtrl	**input, output**	Pointer to the structure with AMCLIB_Windmilling parameters and state variables.

2.12.5.3 Return

The function returns **UNDECIDED** if it is not yet clear whether the rotor is spinning or not. The AMCLIB_Windmilling must then be called again in the next sampling period. The function returns **SPINNING** if the rotor is spinning. If the motor spin is not detected within 2000 iterations, the function will return **STOPPED**.

2.12.5.4 Implementation details

This function internally calls the Tracking Observer function which calculates the PI controller and the integrator subblocks. Refer to the documentation of function AMCLIB_TrackObsrv_FLT (and its parent chapter describing the AMCLIB_TrackObsrv functionality) for a description of the setup of the parameters of the PI controller and integrator blocks used by the AMCLIB_Windmilling_FLT. It is necessary to call the AMCLIB_WindmillingInit_FLT before the first use of AMCLIB_Windmilling_FLT to initialize the state variables.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.12.5.5 Code Example

```
#define ADC_MAX_OFFSET_ERROR 0.002
#define pi (3.1415927F)
#define Ts (1e-4F)
#define f0 (30.0F)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
#define xi (0.707F)
#define w0 (2.0F*pi*f0)
#define Ki (w0*w0)
#define Kp (4.0F*pi*xi*f0)

#include "amclib.h"

SWLIBS_3Syst_FLT              fltUabc;
AMCLIB_WINDMILLING_T_FLT      windStruct;
tFloat                        fltPosition;
tFloat                        fltVelocity;
AMCLIB_WINDMILLING_RET_T      retVal = UNDECIDED;

void main (void)
{
    // Controller parameters
    windStruct.pParamATO.pParamPI.fltCC1sc = (tFloat)(Kp+(Ki*Ts)/2);
    windStruct.pParamATO.pParamPI.fltCC2sc = (tFloat)(-Kp+(Ki*Ts)/2);
    windStruct.pParamATO.pParamPI.fltUpperLimit = (tFloat)(20000.0);
    windStruct.pParamATO.pParamPI.fltLowerLimit = (tFloat)(-20000.0);
    // Integrator parameters
    windStruct.pParamATO.pParamInteg.fltC1 = (tFloat)(Ts/2);

    // Initialize AMCLIB_Windmilling
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_WindmillingInit_FLT(ADC_MAX_OFFSET_ERROR, &windStruct);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_WindmillingInit(ADC_MAX_OFFSET_ERROR, &windStruct, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    AMCLIB_WindmillingInit(ADC_MAX_OFFSET_ERROR, &windStruct);

    while(1);
}

// Periodical function or interrupt
void ISR(void)
{
    // Measure the 3-phase voltages fltUabc
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    retVal = AMCLIB_Windmilling_FLT(&fltUabc, &fltPosition,
                                    &fltVelocity, &windStruct);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    retVal = AMCLIB_Windmilling(&fltUabc, &fltPosition,
                                &fltVelocity, &windStruct, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // as default.
    retVal = AMCLIB_Windmilling(&fltUabc, &fltPosition,
                                &fltVelocity, &windStruct);
}
```

2.12.6 Function AMCLIB_WindmillingInit

2.12.6.1 Description

This function initializes the internal state variables used by AMCLIB_Windmilling. AMCLIB_WindmillingInit must be called before the first use of AMCLIB_Windmilling.

Note: *The input/output pointer must contain valid addresses otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).*

Note: *The input/output pointer must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.12.6.2 Re-entrancy:

The function is re-entrant for a different `pCtrl`.

2.12.6.3 Function AMCLIB_WindmillingInit_F32

2.12.6.3.1 Declaration

```
void AMCLIB_WindmillingInit_F32(tFrac32 f32ADCMaxError,
AMCLIB_WINDMILLING_T_F32 *const pCtrl);
```

2.12.6.3.2 Arguments

Table 102. AMCLIB_WindmillingInit_F32 arguments

Type	Name	Direction	Description
tFrac32	f32ADCMaxError	**input**	Maximum differential offset error voltage between any two phases of the 3-phase voltage-sensing A/D converters. This value must be scaled the same way as the measured 3-phase voltages. If the provided value is lower than the actual offset, the AMCLIB_Windmilling may return false positive detection even if the rotor is not spinning.
AMCLIB_WINDMILLING_T_F32 *const	pCtrl	**input, output**	Pointer to the structure with parameters and state variables.

2.12.6.3.3 Implementation details

Function AMCLIB_WindmillingInit_F32 must be called before the first use of AMCLIB_Windmilling_F32 to initialize the state variables.

2.12.6.3.4 Code Example

```
#define U_MAX 12
#define ADC_MAX_OFFSET_ERROR FRAC32(0.002/U_MAX)
#define Wmax (2618.0F)
#define pi (3.1415927F)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
#define Ts (1e-4F)
#define f0 (30.0F)
#define xi (0.707F)
#define w0 (2.0F*pi*f0)
#define Ki (w0*w0)
#define Kp (4.0F*pi*xi*f0)

#include "amclib.h"

SWLIBS_3Syst_F32              f32Uabc;
AMCLIB_WINDMILLING_T_F32      windStruct;
tFrac32                       f32Position;
tFrac32                       f32Velocity;
AMCLIB_WINDMILLING_RET_T      retVal = UNDECIDED;

void main (void)
{
    // Controller parameters
    windStruct.pParamATO.pParamPI.f32CC1sc = FRAC32((Kp+(Ki*Ts)/2)*pi/Wmax);
    windStruct.pParamATO.pParamPI.f32CC2sc = FRAC32((-Kp+(Ki*Ts)/2)*pi/Wmax);
    windStruct.pParamATO.pParamPI.f32UpperLimit = FRAC32(1.0);
    windStruct.pParamATO.pParamPI.f32LowerLimit = FRAC32(-1.0);
    windStruct.pParamATO.pParamPI.u16NShift = (tU16)0;
    // Integrator parameters
    windStruct.pParamATO.pParamInteg.f32C1 = FRAC32((Ts/2)*Wmax/pi);
    windStruct.pParamATO.pParamInteg.u16NShift = (tU16)0;

    // Initialize AMCLIB_Windmilling
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_WindmillingInit_F32(ADC_MAX_OFFSET_ERROR, &windStruct);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_WindmillingInit(ADC_MAX_OFFSET_ERROR, &windStruct, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    AMCLIB_WindmillingInit(ADC_MAX_OFFSET_ERROR, &windStruct);

    while(1);
}

// Periodical function or interrupt
void ISR(void)
{
    // Measure the 3-phase voltages f32Uabc
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    retVal = AMCLIB_Windmilling_F32(&f32Uabc, &f32Position,
                                    &f32Velocity, &windStruct);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    retVal = AMCLIB_Windmilling(&f32Uabc, &f32Position,
                                &f32Velocity, &windStruct, F32);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    retVal = AMCLIB_Windmilling(&f32Uabc, &f32Position,
                                &f32Velocity, &windStruct);
}
```

2.12.6.4 Function AMCLIB_WindmillingInit_F16

2.12.6.4.1 Declaration

```
void AMCLIB_WindmillingInit_F16(tFrac16 f16ADCMaxError,
AMCLIB_WINDMILLING_T_F16 *const pCtrl);
```

2.12.6.4.2 Arguments

Table 103. AMCLIB_WindmillingInit_F16 arguments

Type	Name	Direction	Description
tFrac16	f16ADCMaxError	**input**	Maximum differential offset error voltage between any two phases of the 3-phase voltage-sensing A/D converters. This value must be scaled the same way as the measured 3-phase voltages. If the provided value is lower than the actual offset, the AMCLIB_Windmilling may return false positive detection even if the rotor is not spinning.
AMCLIB_WINDMILLING_T_F16 *const	pCtrl	**input, output**	Pointer to the structure with parameters and state variables.

2.12.6.4.3 Implementation details

Function AMCLIB_WindmillingInit_F16 must be called before the first use of AMCLIB_Windmilling_F16 to initialize the state variables.

2.12.6.4.4 Code Example

```
#define U_MAX 12
#define ADC_MAX_OFFSET_ERROR FRAC16(0.002/U_MAX)
#define Wmax (2618.0F)
#define pi (3.1415927F)
#define Ts (1e-4F)
#define f0 (30.0F)
#define xi (0.707F)
#define w0 (2.0F*pi*f0)
#define Ki (w0*w0)
#define Kp (4.0F*pi*xi*f0)

#include "amclib.h"

SWLIBS_3Syst_F16              f16Uabc;
AMCLIB_WINDMILLING_T_F16      windStruct;
tFrac16                       f16Position;
tFrac16                       f16Velocity;
AMCLIB_WINDMILLING_RET_T      retVal = UNDECIDED;

void main (void)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
{
    // Controller parameters
    windStruct.pParamATO.pParamPI.f16CC1sc = FRAC16((Kp+(Ki*Ts)/2)*pi/Wmax);
    windStruct.pParamATO.pParamPI.f16CC2sc = FRAC16((-Kp+(Ki*Ts)/2)*pi/Wmax);
    windStruct.pParamATO.pParamPI.f16UpperLimit = FRAC16(1.0);
    windStruct.pParamATO.pParamPI.f16LowerLimit = FRAC16(-1.0);
    windStruct.pParamATO.pParamPI.u16NShift = (tU16)0;
    // Integrator parameters
    windStruct.pParamATO.pParamInteg.f16C1 = FRAC16((Ts/2)*Wmax/pi);
    windStruct.pParamATO.pParamInteg.u16NShift = (tU16)0;

    // Initialize AMCLIB_Windmilling
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_WindmillingInit_F16(ADC_MAX_OFFSET_ERROR, &windStruct);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_WindmillingInit(ADC_MAX_OFFSET_ERROR, &windStruct, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    AMCLIB_WindmillingInit(ADC_MAX_OFFSET_ERROR, &windStruct);

    while(1);
}

// Periodical function or interrupt
void ISR(void)
{
    // Measure the 3-phase voltages f16Uabc
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    retVal = AMCLIB_Windmilling_F16(&f16Uabc, &f16Position,
                                    &f16Velocity, &windStruct);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    retVal = AMCLIB_Windmilling(&f16Uabc, &f16Position,
                                &f16Velocity, &windStruct, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    retVal = AMCLIB_Windmilling(&f16Uabc, &f16Position,
                                &f16Velocity, &windStruct);
}
```

2.12.6.5 Function AMCLIB_WindmillingInit_FLT

2.12.6.5.1 Declaration

```
void AMCLIB_WindmillingInit_FLT(tFloat fltADCMaxError,
AMCLIB_WINDMILLING_T_FLT *const pCtrl);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.12.6.5.2 Arguments

Table 104. AMCLIB_WindmillingInit_FLT arguments

Type	Name	Direction	Description
tFloat	fltADCMaxError	**input**	Maximum differential offset error voltage between any two phases of the 3-phase voltage-sensing A/D converters. If the provided value is lower than the actual offset, the AMCLIB_Windmilling may return false positive detection even if the rotor is not spinning. This value must not be zero.
AMCLIB_WINDMILLING_T_FLT *const	pCtrl	**input, output**	Pointer to the structure with parameters and state variables.

2.12.6.5.3 Implementation details

Function AMCLIB_WindmillingInit_FLT must be called before the first use of AMCLIB_Windmilling_FLT to initialize the state variables.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.12.6.5.4 Code Example

```
#define ADC_MAX_OFFSET_ERROR 0.002
#define pi (3.1415927F)
#define Ts (1e-4F)
#define f0 (30.0F)
#define xi (0.707F)
#define w0 (2.0F*pi*f0)
#define Ki (w0*w0)
#define Kp (4.0F*pi*xi*f0)

#include "amclib.h"

SWLIBS_3Syst_FLT              fltUabc;
AMCLIB_WINDMILLING_T_FLT      windStruct;
tFloat                        fltPosition;
tFloat                        fltVelocity;
AMCLIB_WINDMILLING_RET_T      retVal = UNDECIDED;

void main (void)
{
    // Controller parameters
    windStruct.pParamATO.pParamPI.fltCC1sc = (tFloat)(Kp+(Ki*Ts)/2);
    windStruct.pParamATO.pParamPI.fltCC2sc = (tFloat)(-Kp+(Ki*Ts)/2);
    windStruct.pParamATO.pParamPI.fltUpperLimit = (tFloat)(1.0);
    windStruct.pParamATO.pParamPI.fltLowerLimit = (tFloat)(-1.0);
    // Integrator parameters
    windStruct.pParamATO.pParamInteg.fltC1 = (tFloat)(Ts/2);

    // Initialize AMCLIB_Windmilling
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    AMCLIB_WindmillingInit_FLT(ADC_MAX_OFFSET_ERROR, &windStruct);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    AMCLIB_WindmillingInit(ADC_MAX_OFFSET_ERROR, &windStruct, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    AMCLIB_WindmillingInit(ADC_MAX_OFFSET_ERROR, &windStruct);

    while(1);
}

// Periodical function or interrupt
void ISR(void)
{
    // Measure the 3-phase voltages fltUabc
    // (...)

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    retVal = AMCLIB_Windmilling_FLT(&fltUabc, &fltPosition,
                                    &fltVelocity, &windStruct);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    retVal = AMCLIB_Windmilling(&fltUabc, &fltPosition,
                                &fltVelocity, &windStruct, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    retVal = AMCLIB_Windmilling(&fltUabc, &fltPosition,
                                &fltVelocity, &windStruct);
}
```

2.13 Function GDFLIB_FilterFIR

The function performs a single iteration of an finite impulse response (FIR) filter.

2.13.1 Description

The function calculates the response of a direct-form FIR filter for one input sample. The FIR filter is defined by the difference equation

$$y(n) = \sum_{k=0}^{N} h_k x(n-k)$$

Equation GDFLIB_FilterFIR_Eq1

where *x* is the input signal, *y* is the output signal, *h* are the filter coefficients, and *N* is the filter order. An *N*-th order FIR filter requires *N* + 1 coefficients.

Function GDFLIB_FilterFIRInit should be called before the first use of the filter to clear the state buffer.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.13.2 Re-entrancy

The function is re-entrant for a different `pState`.

2.13.3 Function GDFLIB_FilterFIR_F32

2.13.3.1 Declaration

```
tFrac32 GDFLIB_FilterFIR_F32(tFrac32 f32In,
const GDFLIB_FILTERFIR_PARAM_T_F32 *const pParam,
GDFLIB_FILTERFIR_STATE_T_F32 *const pState);
```

2.13.3.2 Arguments

Table 105. GDFLIB_FilterFIR_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input sample.
const GDFLIB_FILTERFIR_PARAM_T_F32 *const	pParam	**input**	Pointer to the parameter structure.
GDFLIB_FILTERFIR_STATE_T_F32 *const	pState	**input, output**	Pointer to the filter state structure.

2.13.3.3 Return

Filter response for the input sample.

2.13.3.4 Implementation details

The implementation uses a 64-bit accumulator in 32.31 format. Multiplication results are calculated in a precision limited to 31 fractional bits. The accumulator is allowed to wrap-around during calculation, the final accumulator state is saturated to the Q1.31 format.

2.13.3.5 Code Example

```
#include "gdflib.h"

#define FIR_ORDER 15
#define FIR_NUMTAPS ((FIR_ORDER) + 1)

GDFLIB_FILTERFIR_PARAM_T_F32 Param;
GDFLIB_FILTERFIR_STATE_T_F32 State0, State1, State2;

tFrac32 f32InBuf[FIR_NUMTAPS];
tFrac32 f32CoefBuf[FIR_NUMTAPS];

#define OUT_LEN 16

void main(void)
{
    int ii;
    tFrac32 f32OutBuf0[OUT_LEN];
    tFrac32 f32OutBuf1[OUT_LEN];
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    tFrac32 f32OutBuf2[OUT_LEN];

    // Define a simple low-pass filter
    // The filter coefficients were calculated by the following
    // Matlab function (coefficients are contained in Hd.Numerator):
    //
    // function Hd = fir_example
    // FIR_EXAMPLE Returns a discrete-time filter object.
    // N    = 15;
    // F6dB = 0.5;
    //
    // h = fdesign.lowpass('n,fc', N, F6dB);
    //
    // Hd = design(h, 'window');
    // return;
    ii = 0;
    f32CoefBuf[ii++] = 0xFFB10C14;
    f32CoefBuf[ii++] = 0xFF779D25;
    f32CoefBuf[ii++] = 0x01387DD7;
    f32CoefBuf[ii++] = 0x028E6845;
    f32CoefBuf[ii++] = 0xFB245142;
    f32CoefBuf[ii++] = 0xF7183CC7;
    f32CoefBuf[ii++] = 0x11950A3C;
    f32CoefBuf[ii++] = 0x393ED867;
    f32CoefBuf[ii++] = 0x393ED867;
    f32CoefBuf[ii++] = 0x11950A3C;
    f32CoefBuf[ii++] = 0xF7183CC7;
    f32CoefBuf[ii++] = 0xFB245142;
    f32CoefBuf[ii++] = 0x028E6845;
    f32CoefBuf[ii++] = 0x01387DD7;
    f32CoefBuf[ii++] = 0xFF779D25;
    f32CoefBuf[ii++] = 0xFFB10C14;

    Param.u32Order = FIR_ORDER;
    Param.pCoefBuf = &f32CoefBuf[0];

    // Initialize FIR filter
    GDFLIB_FilterFIRInit_F32(&Param, &State0, &f32InBuf[0]);

    // Initialize FIR filter
    GDFLIB_FilterFIRInit(&Param, &State1, &f32InBuf[0], F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // Initialize FIR filter
    GDFLIB_FilterFIRInit(&Param, &State2, &f32InBuf[0]);

    // Compute step response of the filter
    for(ii=0; ii < OUT_LEN; ii++)
    {
      // f32OutBuf0 contains step response of the filter
      f32OutBuf0[ii] = GDFLIB_FilterFIR_F32(0x7FFFFFFF, &Param, &State0);

      // f32OutBuf1 contains step response of the filter
      f32OutBuf1[ii] = GDFLIB_FilterFIR(0x7FFFFFFF, &Param, &State1, F32);

      // ##############################################################
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
        // Available only if 32-bit fractional implementation selected
        // as default
        // ##############################################################

        // f32OutBuf2 contains step response of the filter
        f32OutBuf2[ii] = GDFLIB_FilterFIR(0x7FFFFFFF, &Param, &State2);
      }
      // After the loop the f32OutBuf0, f32OutBuf1, f32OutBuf2 shall contains
      // the following values:
      // {0xFFB1009E, 0xFF2801B0, 0x005FFF40, 0x02EDFA24, 0xFE1203DC,
      //  0xF52A15AC, 0x06BEF282, 0x3FFC8006, 0x793A0D8A, 0x7FFFFFFF,
      //  0x7FFFFFFF, 0x7D0B05E8, 0x7F9900CC, 0x7FFFFFFF, 0x7FFFFFFF,
      //  0x7FF9000C}
}
```

2.13.4 Function GDFLIB_FilterFIR_F16

2.13.4.1 Declaration

```
tFrac16 GDFLIB_FilterFIR_F16(tFrac16 f16In,
const GDFLIB_FILTERFIR_PARAM_T_F16 *const pParam,
GDFLIB_FILTERFIR_STATE_T_F16 *const pState);
```

2.13.4.2 Arguments

Table 106. GDFLIB_FilterFIR_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input sample.
const GDFLIB_FILTERFIR_PARAM_T_F16 *const	pParam	**input**	Pointer to the parameter structure.
GDFLIB_FILTERFIR_STATE_T_F16 *const	pState	**input, output**	Pointer to the filter state structure.

2.13.4.3 Return

Filter response for the input sample.

2.13.4.4 Implementation details

The implementation uses a 64-bit accumulator in 32.31 format. Multiplication results are calculated in full precision. The accumulator is allowed to wrap-around during calculation, the final accumulator state is saturated to the Q1.15 range.

2.13.4.5 Code Example

```
#include "gdflib.h"

#define FIR_ORDER 15
#define FIR_NUMTAPS ((FIR_ORDER) + 1)

GDFLIB_FILTERFIR_PARAM_T_F16 Param;
GDFLIB_FILTERFIR_STATE_T_F16 State0, State1, State2;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
tFrac16 f16InBuf[FIR_NUMTAPS + 1] __attribute__((aligned(4)));
tFrac16 f16CoefBuf[FIR_NUMTAPS + 1] __attribute__((aligned(4)));

#define OUT_LEN 16

void main(void)
{
    int ii;
    tFrac16 f16OutBuf0[OUT_LEN];
    tFrac16 f16OutBuf1[OUT_LEN];
    tFrac16 f16OutBuf2[OUT_LEN];

    // Define a simple low-pass filter
    // The filter coefficients were calculated by the following
    // Matlab function (coefficients are contained in Hd.Numerator):
    //
    // function Hd = fir_example
    // FIR_EXAMPLE Returns a discrete-time filter object.
    // N    = 15;
    // F6dB = 0.5;
    //
    // h = fdesign.lowpass('n,fc', N, F6dB);
    //
    // Hd = design(h, 'window');
    // return;
    ii = 0;
    f16CoefBuf[ii++] = 0xFFB1;
    f16CoefBuf[ii++] = 0xFF77;
    f16CoefBuf[ii++] = 0x0138;
    f16CoefBuf[ii++] = 0x028E;
    f16CoefBuf[ii++] = 0xFB24;
    f16CoefBuf[ii++] = 0xF718;
    f16CoefBuf[ii++] = 0x1195;
    f16CoefBuf[ii++] = 0x393E;
    f16CoefBuf[ii++] = 0x393E;
    f16CoefBuf[ii++] = 0x1195;
    f16CoefBuf[ii++] = 0xF718;
    f16CoefBuf[ii++] = 0xFB24;
    f16CoefBuf[ii++] = 0x028E;
    f16CoefBuf[ii++] = 0x0138;
    f16CoefBuf[ii++] = 0xFF77;
    f16CoefBuf[ii++] = 0xFFB1;

    Param.u16Order = FIR_ORDER;
    Param.pCoefBuf = &f16CoefBuf[0];

    // Initialize FIR filter
    GDFLIB_FilterFIRInit_F16(&Param, &State0, &f16InBuf[0]);

    // Initialize FIR filter
    GDFLIB_FilterFIRInit(&Param, &State1, &f16InBuf[0], F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // Initialize FIR filter
    GDFLIB_FilterFIRInit(&Param, &State2, &f16InBuf[0]);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Compute step response of the filter
    for(ii=0; ii < OUT_LEN; ii++)
    {
      // f16OutBuf0 contains step response of the filter
      f16OutBuf0[ii] = GDFLIB_FilterFIR_F16(0x7FFF, &Param, &State0);

      // f16OutBuf1 contains step response of the filter
      f16OutBuf1[ii] = GDFLIB_FilterFIR(0x7FFF, &Param, &State1, F16);

      // ##############################################################
      // Available only if 16-bit fractional implementation selected
      // as default
      // ##############################################################

      // f16OutBuf2 contains step response of the filter
      f16OutBuf2[ii] = GDFLIB_FilterFIR(0x7FFF, &Param, &State2);
    }
    // After the loop the f16OutBuf0, f16OutBuf1, f16OutBuf2 shall
    // contains the following values:
    // {0xFFB1, 0xFF28, 0x005F, 0x02ED, 0xFE12, 0xF52A, 0x06BE, 0x3FFC,
    //  0x793A, 0x7FFF, 0x7FFF, 0x7D0B, 0x7F99, 0x7FFF, 0x7FFF, 0x7FF9}
}
```

2.13.5 Function GDFLIB_FilterFIR_FLT

2.13.5.1 Declaration

```
tFloat GDFLIB_FilterFIR_FLT(tFloat fltIn, const
GDFLIB_FILTERFIR_PARAM_T_FLT *const pParam,
GDFLIB_FILTERFIR_STATE_T_FLT *const pState);
```

2.13.5.2 Arguments

Table 107. GDFLIB_FilterFIR_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input sample.
const GDFLIB_FILTERFIR_PARAM_T_FLT *const	pParam	**input**	Pointer to a parameter structure.
GDFLIB_FILTERFIR_STATE_T_FLT *const	pState	**input, output**	Pointer to a filter state structure.

2.13.5.3 Return

Filter response for the input sample.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.13.5.4 Code Example

```
#include "gdflib.h"
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
#define FIR_ORDER 15
#define FIR_NUMTAPS ((FIR_ORDER) + 1)

GDFLIB_FILTERFIR_PARAM_T_FLT Param;
GDFLIB_FILTERFIR_STATE_T_FLT State0, State1, State2;

tFloat fltInBuf[FIR_NUMTAPS];
tFloat fltCoefBuf[FIR_NUMTAPS];

#define OUT_LEN 16

void main(void)
{
    int ii;
    tFloat fltOutBuf0[OUT_LEN];
    tFloat fltOutBuf1[OUT_LEN];
    tFloat fltOutBuf2[OUT_LEN];

    // Define a simple low-pass filter
    // The filter coefficients were calculated by the following
    // Matlab function (coefficients are contained in Hd.Numerator):
    //
    // function Hd = fir_example
    // FIR_EXAMPLE Returns a discrete-time filter object.
    // N    = 15;
    // F6dB = 0.5;
    //
    // h = fdesign.lowpass('n,fc', N, F6dB);
    //
    // Hd = design(h, 'window');
    // return;
    ii = 0;
    fltCoefBuf[ii++] = -0.997590551617365;
    fltCoefBuf[ii++] = -0.995837825348525;
    fltCoefBuf[ii++] =  0.0095364856578114;
    fltCoefBuf[ii++] =  0.0199709259997918;
    fltCoefBuf[ii++] = -0.962045819946586;
    fltCoefBuf[ii++] = -0.930427167532233;
    fltCoefBuf[ii++] =  0.137360839237161;
    fltCoefBuf[ii++] =  0.447230387221663;
    fltCoefBuf[ii++] =  0.447230387221663;
    fltCoefBuf[ii++] =  0.137360839237161;
    fltCoefBuf[ii++] = -0.30427167532233;
    fltCoefBuf[ii++] = -0.962045819946586;
    fltCoefBuf[ii++] =  0.199709259997918;
    fltCoefBuf[ii++] =  0.0095364856578114;
    fltCoefBuf[ii++] = -0.995837825348525;
    fltCoefBuf[ii++] = -0.997590551617365;

    Param.u32Order = FIR_ORDER;
    Param.pCoefBuf = &fltCoefBuf[0];

    // Initialize FIR filter
    GDFLIB_FilterFIRInit_FLT(&Param, &State0, &fltInBuf[0]);

    // Initialize FIR filter
    GDFLIB_FilterFIRInit(&Param, &State1, &fltInBuf[0], FLT);

    // ##############################################################
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // Initialize FIR filter
    GDFLIB_FilterFIRInit(&Param, &State2, &fltInBuf[0]);

    // Compute step response of the filter
    for(ii=0; ii < OUT_LEN; ii++)
    {
      // fltOutBuf0 contains step response of the filter
      fltOutBuf0[ii] = GDFLIB_FilterFIR_FLT((tFloat)(1), &Param, &State0);

      // fltOutBuf1 contains step response of the filter
      fltOutBuf1[ii] = GDFLIB_FilterFIR((tFloat)(1), &Param, &State1, FLT);

      // ##############################################################
      // Available only if single precision floating point
      // implementation selected as default
      // ##############################################################

      // fltOutBuf2 contains step response of the filter
      fltOutBuf2[ii] = GDFLIB_FilterFIR((tFloat)(1), &Param, &State2);
    }
    // After the loop the fltOutBuf0, fltOutBuf1, fltOutBuf2 shall contains
    // the following values:
    // {-0.99759054, -1.9934283, -1.9838918, -1.963921, -2.9259667,
    //  -3.8563938, -3.719033, -3.2718027, -2.8245723, -2.6872115,
    //  -2.9914832, -3.9535291, -3.7538199, -3.7442834, -4.7401214,
    //  -5.7377119}
}
```

2.13.6 Function GDFLIB_FilterFIRInit

This function initializes the FIR filter buffers.

2.13.6.1 Description

The function performs the initialization procedure for the GDFLIB_FilterFIR function. In particular, the function performs the following operations:

1. Resets the input buffer index to zero.
2. Initializes the input buffer pointer to the pointer provided as an argument.
3. Resets the input buffer.

After initialization, made by the function, the parameters and state structures should be provided as arguments to calls of the GDFLIB_FilterFIR function.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

The input buffer pointer (`State->pInBuf`) must point to a Read/Write memory region, which must be at least the number of the filter taps long. The number of taps in a filter is equal to the filter order + 1. There is no restriction as to the location of the parameters structure as long as it is readable.

CAUTION: *No check is performed for R/W capability and the length of the input buffer (`pState->pInBuf`). In case of passing incorrect pointer to the function, an unexpected*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

behavior of the function might be expected including the incorrect memory access exception.

2.13.6.2 Re-entrancy

The function is re-entrant only if the calling code is provided with a distinct instance of the structure pointed to by `pState`.

2.13.6.3 Function GDFLIB_FilterFIRInit_F32

2.13.6.3.1 Declaration

```
void GDFLIB_FilterFIRInit_F32(const GDFLIB_FILTERFIR_PARAM_T_F32
*const pParam, GDFLIB_FILTERFIR_STATE_T_F32 *const pState,
tFrac32 *pInBuf);
```

2.13.6.3.2 Arguments

Table 108. GDFLIB_FilterFIRInit_F32 arguments

Type	Name	Direction	Description
const GDFLIB_FILTERFIR_PARAM_T_F32 *const	pParam	**input**	Pointer to the parameters structure.
GDFLIB_FILTERFIR_STATE_T_F32 *const	pState	**input, output**	Pointer to the state structure.
tFrac32 *	pInBuf	**input, output**	Pointer to a buffer for storing filter input signal values, must point to a R/W memory region and must be a filter order + 1 long.

2.13.6.3.3 Implementation details

The function performs the initialization procedure for the GDFLIB_FilterFIR_F32 function. In particular, the function performs the following operations:

1. Resets the input buffer index to zero.
2. Initializes the input buffer pointer to the pointer provided as an argument.
3. Resets the input buffer.

After initialization, made by the function, the parameters and state structures should be provided as arguments to calls of the GDFLIB_FilterFIR_F32 function.

2.13.6.3.4 Code Example

```
#include "gdflib.h"

#define FIR_ORDER 15
#define FIR_NUMTAPS ((FIR_ORDER) + 1)

GDFLIB_FILTERFIR_PARAM_T_F32 Param;
GDFLIB_FILTERFIR_STATE_T_F32 State0, State1, State2;

tFrac32 f32InBuf[FIR_NUMTAPS];
tFrac32 f32CoefBuf[FIR_NUMTAPS];
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
#define OUT_LEN 16

void main(void)
{
    int ii;
    tFrac32 f32OutBuf0[OUT_LEN];
    tFrac32 f32OutBuf1[OUT_LEN];
    tFrac32 f32OutBuf2[OUT_LEN];

    // Define a simple low-pass filter
    // The filter coefficients were calculated by the following
    // Matlab function (coefficients are contained in Hd.Numerator):
    //
    // function Hd = fir_example
    // FIR_EXAMPLE Returns a discrete-time filter object.
    // N    = 15;
    // F6dB = 0.5;
    //
    // h = fdesign.lowpass('n,fc', N, F6dB);
    //
    // Hd = design(h, 'window');
    // return;
    ii = 0;
    f32CoefBuf[ii++] = 0xFFB10C14;
    f32CoefBuf[ii++] = 0xFF779D25;
    f32CoefBuf[ii++] = 0x01387DD7;
    f32CoefBuf[ii++] = 0x028E6845;
    f32CoefBuf[ii++] = 0xFB245142;
    f32CoefBuf[ii++] = 0xF7183CC7;
    f32CoefBuf[ii++] = 0x11950A3C;
    f32CoefBuf[ii++] = 0x393ED867;
    f32CoefBuf[ii++] = 0x393ED867;
    f32CoefBuf[ii++] = 0x11950A3C;
    f32CoefBuf[ii++] = 0xF7183CC7;
    f32CoefBuf[ii++] = 0xFB245142;
    f32CoefBuf[ii++] = 0x028E6845;
    f32CoefBuf[ii++] = 0x01387DD7;
    f32CoefBuf[ii++] = 0xFF779D25;
    f32CoefBuf[ii++] = 0xFFB10C14;

    Param.u32Order = FIR_ORDER;
    Param.pCoefBuf = &f32CoefBuf[0];

    // Initialize FIR filter
    GDFLIB_FilterFIRInit_F32(&Param, &State0, &f32InBuf[0]);

    // Initialize FIR filter
    GDFLIB_FilterFIRInit(&Param, &State1, &f32InBuf[0], F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // Initialize FIR filter
    GDFLIB_FilterFIRInit(&Param, &State2, &f32InBuf[0]);

    // Compute step response of the filter
    for(ii=0; ii < OUT_LEN; ii++)
    {
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
        // f32OutBuf0 contains step response of the filter
        f32OutBuf0[ii] = GDFLIB_FilterFIR_F32(0x7FFFFFFF, &Param, &State0);

        // f32OutBuf1 contains step response of the filter
        f32OutBuf1[ii] = GDFLIB_FilterFIR(0x7FFFFFFF, &Param, &State1, F32);

        // ##############################################################
        // Available only if 32-bit fractional implementation selected
        // as default
        // ##############################################################

        // f32OutBuf2 contains step response of the filter
        f32OutBuf2[ii] = GDFLIB_FilterFIR(0x7FFFFFFF, &Param, &State2);
    }
    // After the loop the f32OutBuf0, f32OutBuf1, f32OutBuf2 shall contains
    // the following values:
    // {0xFFB1009E, 0xFF2801B0, 0x005FFF40, 0x02EDFA24, 0xFE1203DC,
    //  0xF52A15AC, 0x06BEF282, 0x3FFC8006, 0x793A0D8A, 0x7FFFFFFF,
    //  0x7FFFFFFF, 0x7D0B05E8, 0x7F9900CC, 0x7FFFFFFF, 0x7FFFFFFF,
    //  0x7FF9000C}
}
```

2.13.6.4 Function GDFLIB_FilterFIRInit_F16

2.13.6.4.1 Declaration

```
void GDFLIB_FilterFIRInit_F16(const GDFLIB_FILTERFIR_PARAM_T_F16
*const pParam, GDFLIB_FILTERFIR_STATE_T_F16 *const pState,
tFrac16 *pInBuf);
```

2.13.6.4.2 Arguments

Table 109. GDFLIB_FilterFIRInit_F16 arguments

Type	Name	Direction	Description
const GDFLIB_FILTERFIR_PARAM_T_F16 *const	pParam	**input**	Pointer to the parameters structure.
GDFLIB_FILTERFIR_STATE_T_F16 *const	pState	**input, output**	Pointer to the state structure.
tFrac16 *	pInBuf	**input, output**	Pointer to a buffer for storing filter input signal values, must point to a R/W memory region and must be a filter order + 1 long.

2.13.6.4.3 Code Example

```
#include "gdflib.h"

#define FIR_ORDER 15
#define FIR_NUMTAPS ((FIR_ORDER) + 1)

GDFLIB_FILTERFIR_PARAM_T_F16 Param;
GDFLIB_FILTERFIR_STATE_T_F16 State0, State1, State2;

tFrac16 f16InBuf[FIR_NUMTAPS];
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
tFrac16 f16CoefBuf[FIR_NUMTAPS];

#define OUT_LEN 16

void main(void)
{
    int ii;
    tFrac16 f16OutBuf0[OUT_LEN];
    tFrac16 f16OutBuf1[OUT_LEN];
    tFrac16 f16OutBuf2[OUT_LEN];

    // Define a simple low-pass filter
    // The filter coefficients were calculated by the following
    // Matlab function (coefficients are contained in Hd.Numerator):
    //
    // function Hd = fir_example
    // FIR_EXAMPLE Returns a discrete-time filter object.
    // N    = 15;
    // F6dB = 0.5;
    //
    // h = fdesign.lowpass('n,fc', N, F6dB);
    //
    // Hd = design(h, 'window');
    // return;
    ii = 0;
    f16CoefBuf[ii++] = 0xFFB1;
    f16CoefBuf[ii++] = 0xFF77;
    f16CoefBuf[ii++] = 0x0138;
    f16CoefBuf[ii++] = 0x028E;
    f16CoefBuf[ii++] = 0xFB24;
    f16CoefBuf[ii++] = 0xF718;
    f16CoefBuf[ii++] = 0x1195;
    f16CoefBuf[ii++] = 0x393E;
    f16CoefBuf[ii++] = 0x393E;
    f16CoefBuf[ii++] = 0x1195;
    f16CoefBuf[ii++] = 0xF718;
    f16CoefBuf[ii++] = 0xFB24;
    f16CoefBuf[ii++] = 0x028E;
    f16CoefBuf[ii++] = 0x0138;
    f16CoefBuf[ii++] = 0xFF77;
    f16CoefBuf[ii++] = 0xFFB1;

    Param.u16Order = FIR_ORDER;
    Param.pCoefBuf = &f16CoefBuf[0];

    // Initialize FIR filter
    GDFLIB_FilterFIRInit_F16(&Param, &State0, &f16InBuf[0]);

    // Initialize FIR filter
    GDFLIB_FilterFIRInit(&Param, &State1, &f16InBuf[0], F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // Initialize FIR filter
    GDFLIB_FilterFIRInit(&Param, &State2, &f16InBuf[0]);

    // Compute step response of the filter
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    for(ii=0; ii < OUT_LEN; ii++)
    {
      // f16OutBuf0 contains step response of the filter
      f16OutBuf0[ii] = GDFLIB_FilterFIR_F16(0x7FFF, &Param, &State0);

      // f16OutBuf1 contains step response of the filter
      f16OutBuf1[ii] = GDFLIB_FilterFIR(0x7FFF, &Param, &State1, F16);

      // ##############################################################
      // Available only if 16-bit fractional implementation selected
      // as default
      // ##############################################################

      // f16OutBuf2 contains step response of the filter
      f16OutBuf2[ii] = GDFLIB_FilterFIR(0x7FFF, &Param, &State2);
    }
    // After the loop the f16OutBuf0, f16OutBuf1, f16OutBuf2 shall
    // contains the following values:
    // {0xFFB1, 0xFF28, 0x005F, 0x02ED, 0xFE12, 0xF52A, 0x06BE, 0x3FFC,
    //  0x793A, 0x7FFF, 0x7FFF, 0x7D0B, 0x7F99, 0x7FFF, 0x7FFF, 0x7FF9}
}
```

2.13.6.5 Function GDFLIB_FilterFIRInit_FLT

2.13.6.5.1 Declaration

```
void GDFLIB_FilterFIRInit_FLT(const GDFLIB_FILTERFIR_PARAM_T_FLT
*const pParam, GDFLIB_FILTERFIR_STATE_T_FLT *const pState, tFloat
*pInBuf);
```

2.13.6.5.2 Arguments

Table 110. GDFLIB_FilterFIRInit_FLT arguments

Type	Name	Direction	Description
const GDFLIB_FILTERFIR_PARAM_T_FLT *const	pParam	**input**	Pointer to a parameters structure.
GDFLIB_FILTERFIR_STATE_T_FLT *const	pState	**input, output**	Pointer to a state structure.
tFloat *	pInBuf	**input, output**	Pointer to a buffer for storing filter input signal values, must point to a R/W memory region and must be the filter order + 1 long.

Note: *The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.13.6.5.3 Code Example

```
#include "gdflib.h"

#define FIR_ORDER 15
#define FIR_NUMTAPS ((FIR_ORDER) + 1)

GDFLIB_FILTERFIR_PARAM_T_FLT Param;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
GDFLIB_FILTERFIR_STATE_T_FLT State0, State1, State2;

tFloat fltInBuf[FIR_ORDER];
tFloat fltCoefBuf[FIR_ORDER];

#define OUT_LEN 16

void main(void)
{
    int ii;
    tFloat fltOutBuf0[OUT_LEN];
    tFloat fltOutBuf1[OUT_LEN];
    tFloat fltOutBuf2[OUT_LEN];

    // Define a simple low-pass filter
    // The filter coefficients were calculated by the following
    // Matlab function (coefficients are contained in Hd.Numerator):
    //
    // function Hd = fir_example
    // FIR_EXAMPLE Returns a discrete-time filter object.
    // N    = 15;
    // F6dB = 0.5;
    //
    // h = fdesign.lowpass('n,fc', N, F6dB);
    //
    // Hd = design(h, 'window');
    // return;
    ii = 0;
    fltCoefBuf[ii++] = -0.997590551617365;
    fltCoefBuf[ii++] = -0.995837825348525;
    fltCoefBuf[ii++] =  0.0095364856578114;
    fltCoefBuf[ii++] =  0.0199709259997918;
    fltCoefBuf[ii++] = -0.962045819946586;
    fltCoefBuf[ii++] = -0.930427167532233;
    fltCoefBuf[ii++] =  0.137360839237161;
    fltCoefBuf[ii++] =  0.447230387221663;
    fltCoefBuf[ii++] =  0.447230387221663;
    fltCoefBuf[ii++] =  0.137360839237161;
    fltCoefBuf[ii++] = -0.30427167532233;
    fltCoefBuf[ii++] = -0.962045819946586;
    fltCoefBuf[ii++] =  0.199709259997918;
    fltCoefBuf[ii++] =  0.0095364856578114;
    fltCoefBuf[ii++] = -0.995837825348525;
    fltCoefBuf[ii++] = -0.997590551617365;

    Param.u32Order = FIR_ORDER;
    Param.pCoefBuf = &fltCoefBuf[0];

    // Initialize FIR filter
    GDFLIB_FilterFIRInit_FLT(&Param, &State0, &fltInBuf[0]);

    // Initialize FIR filter
    GDFLIB_FilterFIRInit(&Param, &State1, &fltInBuf[0], FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // Initialize FIR filter
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    GDFLIB_FilterFIRInit(&Param, &State2, &fltInBuf[0]);

    // Compute step response of the filter
    for(ii=0; ii < OUT_LEN; ii++)
    {
      // fltOutBuf0 contains step response of the filter
      fltOutBuf0[ii] = GDFLIB_FilterFIR_FLT((tFloat)(1), &Param, &State0);

      // fltOutBuf1 contains step response of the filter
      fltOutBuf1[ii] = GDFLIB_FilterFIR((tFloat)(1), &Param, &State1, FLT);

      // ##############################################################
      // Available only if single precision floating point
      // implementation selected as default
      // ##############################################################

      // fltOutBuf2 contains step response of the filter
      fltOutBuf2[ii] = GDFLIB_FilterFIR((tFloat)(1), &Param, &State2);
    }
    // After the loop the fltOutBuf0, fltOutBuf1, fltOutBuf2 shall contains
    // the following values:
    // {-0.99759054, -1.9934283, -1.9838918, -1.963921, -2.9259667,
    //  -3.8563938, -3.719033, -3.2718027, -2.8245723, -2.6872115,
    //  -2.9914832, -3.9535291, -3.7538199, -3.7442834, -4.7401214,
    //  -5.7377119}
}
```

2.14 Function GDFLIB_FilterIIR1

This function implements the first order IIR filter.

2.14.1 Description

This function calculates the first order infinite impulse (IIR) filter. The IIR filters are also called recursive filters because both the input and the previously calculated output values are used for calculation of the filter equation in each step. This form of feedback enables transfer of the energy from the output to the input, which theoretically leads to an infinitely long impulse response (IIR).

A general form of the IIR filter expressed as a transfer function in the Z-domain is described as follows:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}}$$

Equation GDFLIB_FilterIIR1_Eq1

where N denotes the filter order. The first order IIR filter in the Z-domain is therefore given from equation GDFLIB_FilterIIR1_Eq1 as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1}}$$

Equation GDFLIB_FilterIIR1_Eq2

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

In order to implement the first order IIR filter on a microcontroller, the discrete time domain representation of the filter, described by equation GDFLIB_FilterIIR1_Eq2, must be transformed into a time difference equation as follows:

$$y(k) = b_0 x(k) + b_1 x(k-1) - a_1 y(k-1)$$

Equation GDFLIB_FilterIIR1_Eq3

Equation GDFLIB_FilterIIR1_Eq3 represents a Direct Form I implementation of a first order IIR filter. It is well known that Direct Form I (DF-I) and Direct Form II (DF-II) implementations of an IIR filter are generally sensitive to parameter quantization if a finite precision arithmetic is considered. This, however, can be neglected when the filter transfer function is broken down into low order sections, i.e. first or second order. The main difference between DF-I and DF-II implementations of an IIR filter is in the number of delay buffers and in the number of guard bits required to handle the potential overflow (in fixed-point variants of the function). The DF-II implementation requires less delay buffers than DF-I, hence less data memory is utilized. On the other hand, since the poles come first in the DF-II realization, the signal entering the state delay-line typically requires a larger dynamic range than the output signal y(k). Therefore, overflow can occur at the delay-line input of the DF-II implementation, unlike in the DF-I implementation (considering a fixed-point implementation).

Because there are two delay buffers necessary for both DF-I and DF-II implementations of the first order IIR filter, the DF-I implementation was chosen to be used in the GDFLIB_FilterIIR1 function.

Figure 83. Direct Form 1 first order IIR filter

The coefficients of the filter depicted in Figure 83 can be designed to meet the requirements for the first order Low (LPF) or High Pass (HPF) filters. Filter coefficients can be calculated using various tools, for example, the Matlab® *butter* function (see examples below).

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.14.2 Re-entrancy

The function is re-entrant for a different `pParam`.

2.14.3 Function GDFLIB_FilterIIR1_F32

2.14.3.1 Declaration

```
tFrac32 GDFLIB_FilterIIR1_F32(tFrac32 f32In,
GDFLIB_FILTER_IIR1_T_F32 *const pParam);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.14.3.2 Arguments

Table 111. GDFLIB_FilterIIR1_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Value of input signal to be filtered in step (k). The value is a 32-bit number in the Q1.31 fractional format.
GDFLIB_FILTER_IIR1_T_F32 *const	pParam	**input, output**	Pointer to the filter structure with a filter buffer and filter parameters.

2.14.3.3 Return

The function returns a 32-bit value in the fractional format Q1.31, representing the filtered value of the input signal in step *k*.

2.14.3.4 Implementation details

In order to avoid overflow during the calculation of the GDFLIB_FilterIIR1_F32 function, filter coefficients must be divided by eight. The coefficients can be calculated using Matlab® as follows:

```
freq_cut = 100;
T_sampling  = 100e-6;

[b,a]=butter(1,[freq_cut*T_sampling*2],'low');
sys=tf(b,a,T_sampling);
bode(sys)

f32B0 = b(1);
f32B1 = b(2);
f32A1 = a(2);
disp('Coefficients for GDFLIB_FilterIIR1 function:');
disp(['f32B0 = FRAC32(' num2str(f32B0,'%1.15f') '/8);']);
disp(['f32B1 = FRAC32(' num2str(f32B1,'%1.15f') '/8);']);
disp(['f32A1 = FRAC32(' num2str(f32A1,'%1.15f') '/8);']);
```

Note: *The filter delay line includes two delay buffers which should be reset after filter initialization. This can be done by assigning to the filter instance a GDFLIB_FILTER_IIR1_DEFAULT_F32 macro during instance declaration or by calling the GDFLIB_FilterIIR1Init_F32 function.*

CAUTION: *Because of fixed point implementation, and to avoid overflow during the calculation of the GDFLIB_FilterIIR1_F32 function, filter coefficients must be divided by eight. Function output is internally multiplied by eight to correct the coefficient scaling.*

2.14.3.5 Code Example

```
#include "gdflib.h"

tFrac32 f32In;
tFrac32 f32Out;

GDFLIB_FILTER_IIR1_T_F32 f32trMyIIR1 = GDFLIB_FILTER_IIR1_DEFAULT_F32;

void main(void)
{
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // input value = 0.25
    f32In = FRAC32(0.25);

    // filter coefficients (LPF 100Hz, Ts=100e-6)
    f32trMyIIR1.trFiltCoeff.f32B0 = FRAC32(0.030468747091254/8);
    f32trMyIIR1.trFiltCoeff.f32B1 = FRAC32(0.030468747091254/8);
    f32trMyIIR1.trFiltCoeff.f32A1 = FRAC32(-0.939062505817492/8);

    // output should be 0x00F99998 ~ FRAC32(0.007617)
    GDFLIB_FilterIIR1Init_F32(&f32trMyIIR1);
    f32Out = GDFLIB_FilterIIR1_F32(f32In, &f32trMyIIR1);

    // output should be 0x00F99998 ~ FRAC32(0.007617)
    GDFLIB_FilterIIR1Init(&f32trMyIIR1, F32);
    f32Out = GDFLIB_FilterIIR1(f32In, &f32trMyIIR1, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x00F99998 ~ FRAC32(0.007617)
    GDFLIB_FilterIIR1Init(&f32trMyIIR1);
    f32Out = GDFLIB_FilterIIR1(f32In, &f32trMyIIR1);
}
```

2.14.4 Function GDFLIB_FilterIIR1_F16

2.14.4.1 Declaration

```
tFrac16 GDFLIB_FilterIIR1_F16(tFrac16 f16In,
GDFLIB_FILTER_IIR1_T_F16 *const pParam);
```

2.14.4.2 Arguments

Table 112. GDFLIB_FilterIIR1_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Value of input signal to be filtered in step (k). The value is a 16-bit number in the Q1.15 fractional format.
GDFLIB_FILTER_IIR1_T_F16 *const	pParam	**input, output**	Pointer to the filter structure with a filter buffer and filter parameters.

2.14.4.3 Return

The function returns a 16-bit value in the fractional format Q1.15, representing the filtered value of the input signal in step *k*.

2.14.4.4 Implementation details

In order to avoid overflow during the calculation of the GDFLIB_FilterIIR1_F32 function, filter coefficients must be divided by eight. The coefficients can be calculated using Matlab® as follows:

```
freq_cut     = 100;
T_sampling   = 100e-6;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
[b,a]=butter(1,[freq_cut*T_sampling*2],'low');
sys=tf(b,a,T_sampling);
bode(sys)

f16B0 = b(1);
f16B1 = b(2);
f16A1 = a(2);
disp('Coefficients for GDFLIB_FilterIIR1 function:');
disp(['f16B0 = FRAC16(' num2str(f16B0,'%1.15f') '/8);']);
disp(['f16B1 = FRAC16(' num2str(f16B1,'%1.15f') '/8);']);
disp(['f16A1 = FRAC16(' num2str(f16A1,'%1.15f') '/8);']);
```

Note: *The filter delay line includes two delay buffers which should be reset after filter initialization. This can be done by assigning to the filter instance a GDFLIB_FILTER_IIR1_DEFAULT_F16 macro during instance declaration or by calling the GDFLIB_FilterIIR1Init_F16 function.*

CAUTION: *Because of fixed point implementation, and to avoid overflow during the calculation of the GDFLIB_FilterIIR1_F16 function, filter coefficients must be divided by eight. Function output is internally multiplied by eight to correct the coefficient scaling.*

2.14.4.5 Code Example

```
#include "gdflib.h"

tFrac16 f16In;
tFrac16 f16Out;

GDFLIB_FILTER_IIR1_T_F16 f16trMyIIR1 = GDFLIB_FILTER_IIR1_DEFAULT_F16;

void main(void)
{
    // input value = 0.25
    f16In = FRAC16(0.25);

    // filter coefficients (LPF 100Hz, Ts=100e-6)
    f16trMyIIR1.trFiltCoeff.f16B0 = FRAC16(0.030468747091254/8);
    f16trMyIIR1.trFiltCoeff.f16B1 = FRAC16(0.030468747091254/8);
    f16trMyIIR1.trFiltCoeff.f16A1 = FRAC16(-0.939062505817492/8);

    // output should be 0x00F9 ~ FRAC16(0.007617)
    GDFLIB_FilterIIR1Init_F16(&f16trMyIIR1);
    f16Out = GDFLIB_FilterIIR1_F16(f16In, &f16trMyIIR1);

    // output should be 0x00F9 ~ FRAC16(0.007617)
    GDFLIB_FilterIIR1Init(&f16trMyIIR1, F16);
    f16Out = GDFLIB_FilterIIR1(f16In, &f16trMyIIR1, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x00F9 ~ FRAC16(0.007617)
    GDFLIB_FilterIIR1Init(&f16trMyIIR1);
    f16Out = GDFLIB_FilterIIR1(f16In, &f16trMyIIR1);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.14.5 Function GDFLIB_FilterIIR1_FLT

2.14.5.1 Declaration

```
tFloat GDFLIB_FilterIIR1_FLT(tFloat fltIn,
GDFLIB_FILTER_IIR1_T_FLT *const pParam);
```

2.14.5.2 Arguments

Table 113. GDFLIB_FilterIIR1_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Value of the input signal to be filtered in step (k). Input is a 32-bit number that contains a single precision floating point value.
GDFLIB_FILTER_IIR1_T_FLT *const	pParam	**input, output**	Pointer to a filter structure with a filter buffer and filter parameters. Arguments of the structure contain single precision floating point values.

2.14.5.3 Return

The function returns a 32-bit value in single precision floating point format, representing the filtered value of the input signal in step *k*.

2.14.5.4 Implementation details

The coefficients can be calculated using Matlab® as follows:

```
freq_cut = 100;
T_sampling  = 100e-6;

[b,a]=butter(1,[freq_cut*T_sampling*2],'low');
sys=tf(b,a,T_sampling);
bode(sys)

fltB0 = b(1);
fltB1 = b(2);
fltA1 = a(2);
disp('Coefficients for GDFLIB_FilterIIR1 function:');
disp(['fltB0 = (tFloat)(' num2str(fltB0,'%1.15f') ');']);
disp(['fltB1 = (tFloat)(' num2str(fltB1,'%1.15f') ');']);
disp(['fltA1 = (tFloat)(' num2str(fltA1,'%1.15f') ');']);
```

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

CAUTION: *The filter delay line includes two delay buffers which should be reset after filter initialization. This can be done by assigning to the filter instance a* GDFLIB_FILTER_IIR1_DEFAULT_FLT *macro during instance declaration or by calling the* GDFLIB_FilterIIR1Init_FLT *function.*

2.14.5.5 Code Example

```
#include "gdflib.h"

tFloat fltIn;
tFloat fltOut;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
GDFLIB_FILTER_IIR1_T_FLT flttrMyIIR1 = GDFLIB_FILTER_IIR1_DEFAULT_FLT;

void main(void)
{
    // input value = 0.25
    fltIn = (tFloat) 0.25;

    // filter coefficients (LPF 100Hz)
    flttrMyIIR1.trFiltCoeff.fltB0 = (tFloat)(0.030468747091254);
    flttrMyIIR1.trFiltCoeff.fltB1 = (tFloat)(0.030468747091254);
    flttrMyIIR1.trFiltCoeff.fltA1 = (tFloat)(-0.939062505817492);

    // output should be 0.007617
    GDFLIB_FilterIIR1Init_FLT(&flttrMyIIR1);
    fltOut = GDFLIB_FilterIIR1_FLT(fltIn, &flttrMyIIR1);

    // output should be 0.007617
    GDFLIB_FilterIIR1Init(&flttrMyIIR1, FLT);
    fltOut = GDFLIB_FilterIIR1(fltIn, &flttrMyIIR1, FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be 0.007617
    GDFLIB_FilterIIR1Init(&flttrMyIIR1);
    fltOut = GDFLIB_FilterIIR1(fltIn, &flttrMyIIR1);
}
```

2.14.6 Function GDFLIB_FilterIIR1Init

This function initializes the first order IIR filter buffers.

2.14.6.1 Description

This function clears the internal buffers of a first order IIR filter. It shall be called after filter parameter initialization and whenever the filter initialization is required.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

Note: *This function shall not be called together with the GDFLIB_FilterIIR1 function unless periodic clearing of filter buffers is required.*

2.14.6.2 Re-entrancy

The function is re-entrant for a different `pParam`.

2.14.6.3 Function GDFLIB_FilterIIR1Init_F32

2.14.6.3.1 Declaration

```
void GDFLIB_FilterIIR1Init_F32(GDFLIB_FILTER_IIR1_T_F32 *const
pParam);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.14.6.3.2 Arguments

Table 114. GDFLIB_FilterIIR1Init_F32 arguments

Type	Name	Direction	Description
GDFLIB_FILTER_IIR1_T_F32 *const	pParam	**input, output**	Pointer to filter structure with filter buffer and filter parameters.

2.14.6.3.3 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_IIR1_T_F32 f32trMyIIR1 = GDFLIB_FILTER_IIR1_DEFAULT_F32;

void main(void)
{
    // function returns no value
    GDFLIB_FilterIIR1Init_F32(&f32trMyIIR1);

    // function returns no value
    GDFLIB_FilterIIR1Init(&f32trMyIIR1, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // function returns no value
    GDFLIB_FilterIIR1Init(&f32trMyIIR1);
}
```

2.14.6.4 Function GDFLIB_FilterIIR1Init_F16

2.14.6.4.1 Declaration

```
void GDFLIB_FilterIIR1Init_F16(GDFLIB_FILTER_IIR1_T_F16 *const
pParam);
```

2.14.6.4.2 Arguments

Table 115. GDFLIB_FilterIIR1Init_F16 arguments

Type	Name	Direction	Description
GDFLIB_FILTER_IIR1_T_F16 *const	pParam	**input, output**	Pointer to filter structure with filter buffer and filter parameters.

2.14.6.4.3 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_IIR1_T_F16 f16trMyIIR1 = GDFLIB_FILTER_IIR1_DEFAULT_F16;

void main(void)
{
    // function returns no value
    GDFLIB_FilterIIR1Init_F16(&f16trMyIIR1);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // function returns no value
    GDFLIB_FilterIIR1Init(&f16trMyIIR1, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // function returns no value
    GDFLIB_FilterIIR1Init(&f16trMyIIR1);
}
```

2.14.6.5 Function GDFLIB_FilterIIR1Init_FLT

2.14.6.5.1 Declaration

```
void GDFLIB_FilterIIR1Init_FLT(GDFLIB_FILTER_IIR1_T_FLT *const
pParam);
```

2.14.6.5.2 Arguments

Table 116. GDFLIB_FilterIIR1Init_FLT arguments

Type	Name	Direction	Description
GDFLIB_FILTER_IIR1_T_FLT *const	pParam	**input, output**	Pointer to a filter structure with filter buffer and filter parameters. Arguments of the structure contain single precision floating point values.

2.14.6.5.3 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_IIR1_T_FLT flttrMyIIR1 = GDFLIB_FILTER_IIR1_DEFAULT_FLT;

void main(void)
{
    // function returns no value
    GDFLIB_FilterIIR1Init_FLT(&flttrMyIIR1);

    // function returns no value
    GDFLIB_FilterIIR1Init(&flttrMyIIR1, FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // function returns no value
    GDFLIB_FilterIIR1Init(&flttrMyIIR1);
}
```

2.15 Function GDFLIB_FilterIIR2

This function implements the second order (biquad) IIR filter.

© NXP B.V. 2023. All rights reserved.

2.15.1 Description

This function calculates the second order infinite impulse (IIR) filter. The IIR filters are also called recursive filters because both the input and the previously calculated output values are used for calculation of the filter equation in each step. This form of feedback enables transfer of the energy from the output to the input, which theoretically leads to an infinitely long impulse response (IIR).

A general form of the IIR filter expressed as a transfer function in the Z-domain is described as follows:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}}$$

Equation GDFLIB_FilterIIR2_Eq1

where N denotes the filter order. The second order IIR filter in the Z-domain is therefore given from eq. GDFLIB_FilterIIR2_Eq1 as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

Equation GDFLIB_FilterIIR2_Eq2

In order to implement the second order IIR filter on a microcontroller, the discrete time domain representation of the filter, described by eq. GDFLIB_FilterIIR2_Eq2, must be transformed into a time difference equation as follows:

$$y(k) = b_0 x(k) + b_1 x(k-1) + b_2 x(k-2) - a_1 y(k-1) - a_2 y(k-2)$$

Equation GDFLIB_FilterIIR2_Eq3

Equation GDFLIB_FilterIIR2_Eq3 represents a Direct Form I implementation of a second order IIR filter. It is well known that Direct Form I (DF-I) and Direct Form II (DF-II) implementations of an IIR filter are generally sensitive to parameter quantization if a finite precision arithmetic is considered. This, however, can be neglected when the filter transfer function is broken down into low order sections, i.e. first or second order. The main difference between DF-I and DF-II implementations of an IIR filter is in the number of delay buffers and in the number of guard bits required to handle the potential overflow (in fixed-point variants of the function). The DF-II implementation requires fewer delay buffers than DF-I, hence less data memory is utilized. On the other hand, since the poles come first in the DF-II realization, the signal entering the state delay-line typically requires a larger dynamic range than the output signal y(k). Therefore, overflow can occur at the delay-line input of the DF-II implementation, unlike in the DF-I implementation (considering a fixed-point implementation).

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 84. Direct Form 1 second order IIR filter

The coefficients of the filter depicted in Figure 84 can be designed to meet the requirements for the second order Band Pass (BPF) or Band Stop (BSF) filters. Filter coefficients can be calculated using various tools, for example, the Matlab® *butter* function (see examples below).

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.15.2 Re-entrancy

The function is re-entrant for a different `pParam`.

2.15.3 Function GDFLIB_FilterIIR2_F32

2.15.3.1 Declaration

```
tFrac32 GDFLIB_FilterIIR2_F32(tFrac32 f32In,
GDFLIB_FILTER_IIR2_T_F32 *const pParam);
```

2.15.3.2 Arguments

Table 117. GDFLIB_FilterIIR2_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Value of input signal to be filtered in step (k). The value is a 32-bit number in the Q1.31 fractional format.
GDFLIB_FILTER_IIR2_T_F32 *const	pParam	**input, output**	Pointer to the filter structure with a filter buffer and filter parameters.

2.15.3.3 Return

The function returns a 32-bit value in the fractional format Q1.31, representing the filtered value of the input signal in step *k*.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.15.3.4 Implementation details

In order to avoid overflow during the calculation of the GDFLIB_FilterIIR2_F32 function, filter coefficients must be divided by eight. The coefficients can be calculated using Matlab® as follows:

```
freq_bot    = 400;
freq_top    = 625;
T_sampling  = 100e-6;

[b,a]= butter(1,[freq_bot freq_top]*T_sampling *2, 'bandpass');
sys =tf(b,a,T_sampling);
bode(sys,[freq_bot:1:freq_top]*2*pi)

f32B0 = b(1);
f32B1 = b(2);
f32B2 = b(3);
f32A1 = a(2);
f32A2 = a(3);
disp (' Coefficients for GDFLIB_FilterIIR2 function :');
disp ([ 'f32B0 = FRAC32(' num2str( f32B0,'%1.15f' ) '/8);']);
disp ([ 'f32B1 = FRAC32(' num2str( f32B1,'%1.15f' ) '/8);']);
disp ([ 'f32B2 = FRAC32(' num2str( f32B2,'%1.15f' ) '/8);']);
disp ([ 'f32A1 = FRAC32(' num2str( f32A1,'%1.15f' ) '/8);']);
disp ([ 'f32A2 = FRAC32(' num2str( f32A2,'%1.15f' ) '/8);']);
```

Note: *The filter delay line includes four delay buffers which should be reset after filter initialization. This can be done by assigning to the filter instance a GDFLIB_FILTER_IIR2_DEFAULT_F32 macro during instance declaration or by calling the GDFLIB_FilterIIR2Init_F32 function.*

CAUTION: *Because of fixed point implementation, and to avoid overflow during the calculation of the GDFLIB_FilterIIR2_F32 function, filter coefficients must be divided by eight. Function output is internally multiplied by eight to correct the coefficient scaling.*

2.15.3.5 Code Example

```
#include "gdflib.h"

tFrac32 f32In;
tFrac32 f32Out;

GDFLIB_FILTER_IIR2_T_F32 f32trMyIIR2 = GDFLIB_FILTER_IIR2_DEFAULT_F32;

void main(void)
{
    // input value = 0.25
    f32In = FRAC32(0.25);

    // filter coefficients (BPF 400-625Hz, Ts=100e-6)
    f32trMyIIR2.trFiltCoeff.f32B0 = FRAC32(0.066122101544579/8);
    f32trMyIIR2.trFiltCoeff.f32B1 = FRAC32(0/8);
    f32trMyIIR2.trFiltCoeff.f32B2 = FRAC32(-0.066122101544579/8);
    f32trMyIIR2.trFiltCoeff.f32A1 = FRAC32(-1.776189018043779/8);
    f32trMyIIR2.trFiltCoeff.f32A2 = FRAC32(0.867755796910841/8);

    // output should be 0x021DAC18 ~ FRAC32(0.0165305)
    GDFLIB_FilterIIR2Init_F32(&f32trMyIIR2);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    f32Out = GDFLIB_FilterIIR2_F32(f32In, &f32trMyIIR2);

    // output should be 0x021DAC18 ~ FRAC32(0.0165305)
    GDFLIB_FilterIIR2Init(&f32trMyIIR2, F32);
    f32Out = GDFLIB_FilterIIR2(f32In, &f32trMyIIR2, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x021DAC18 ~ FRAC32(0.0165305)
    GDFLIB_FilterIIR2Init(&f32trMyIIR2);
    f32Out = GDFLIB_FilterIIR2(f32In, &f32trMyIIR2);
}
```

2.15.4 Function GDFLIB_FilterIIR2_F16

2.15.4.1 Declaration

```
tFrac16 GDFLIB_FilterIIR2_F16(tFrac16 f16In,
GDFLIB_FILTER_IIR2_T_F16 *const pParam);
```

2.15.4.2 Arguments

Table 118. GDFLIB_FilterIIR2_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Value of input signal to be filtered in step (k). The value is a 16-bit number in the Q1.15 fractional format.
GDFLIB_FILTER_IIR2_T_F16 *const	pParam	**input, output**	Pointer to the filter structure with a filter buffer and filter parameters.

2.15.4.3 Return

The function returns a 16-bit value in the fractional format Q1.15, representing the filtered value of the input signal in step *k*.

2.15.4.4 Implementation details

In order to avoid overflow during the calculation of the GDFLIB_FilterIIR2_F16 function, filter coefficients must be divided by eight. The coefficients can be calculated using Matlab® as follows:

```
freq_bot    = 400;
freq_top    = 625;
T_sampling  = 100e-6;

[b,a]= butter(1,[freq_bot freq_top]*T_sampling *2, 'bandpass');
sys =tf(b,a,T_sampling);
bode(sys,[freq_bot:1:freq_top]*2*pi)

f16B0 = b(1);
f16B1 = b(2);
f16B2 = b(3);
f16A1 = a(2);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
f16A2 = a(3);
disp (' Coefficients for GDFLIB_FilterIIR2 function :');
disp ([ 'f16B0 = FRAC16(' num2str( f16B0,'%1.15f' ) '/8);']);
disp ([ 'f16B1 = FRAC16(' num2str( f16B1,'%1.15f' ) '/8);']);
disp ([ 'f16B2 = FRAC16(' num2str( f16B2,'%1.15f' ) '/8);']);
disp ([ 'f16A1 = FRAC16(' num2str( f16A1,'%1.15f' ) '/8);']);
disp ([ 'f16A2 = FRAC16(' num2str( f16A2,'%1.15f' ) '/8);']);
```

Note: *The filter delay line includes four delay buffers which should be reset after filter initialization. This can be done by assigning to the filter instance a GDFLIB_FILTER_IIR2_DEFAULT_F16 macro during instance declaration or by calling the GDFLIB_FilterIIR2Init_F16 function.*

CAUTION: *Because of fixed point implementation, and to avoid overflow during the calculation of the GDFLIB_FilterIIR2_F16 function, filter coefficients must be divided by eight. Function output is internally multiplied by eight to correct the coefficient scaling.*

2.15.4.5 Code Example

```
#include "gdflib.h"

tFrac16 f16In;
tFrac16 f16Out;

GDFLIB_FILTER_IIR2_T_F16 f16trMyIIR2 = GDFLIB_FILTER_IIR2_DEFAULT_F16;

void main(void)
{
    // input value = 0.25
    f16In = FRAC16(0.25);

    // filter coefficients (BPF 400-625Hz, Ts=100e-6)
    f16trMyIIR2.trFiltCoeff.f16B0 = FRAC16(0.066122101544579/8);
    f16trMyIIR2.trFiltCoeff.f16B1 = FRAC16(0/8);
    f16trMyIIR2.trFiltCoeff.f16B2 = FRAC16(-0.066122101544579/8);
    f16trMyIIR2.trFiltCoeff.f16A1 = FRAC16(-1.776189018043779/8);
    f16trMyIIR2.trFiltCoeff.f16A2 = FRAC16(0.867755796910841/8);

    // output should be 0x021D ~ FRAC16(0.01651)
    GDFLIB_FilterIIR2Init_F16(&f16trMyIIR2);
    f16Out = GDFLIB_FilterIIR2_F16(f16In, &f16trMyIIR2);

    // output should be 0x021D ~ FRAC16(0.01651)
    GDFLIB_FilterIIR2Init(&f16trMyIIR2, F16);
    f16Out = GDFLIB_FilterIIR2(f16In, &f16trMyIIR2, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x021D ~ FRAC16(0.01651)
    GDFLIB_FilterIIR2Init(&f16trMyIIR2);
    f16Out = GDFLIB_FilterIIR2(f16In, &f16trMyIIR2);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.15.5 Function GDFLIB_FilterIIR2_FLT

2.15.5.1 Declaration

```
tFloat GDFLIB_FilterIIR2_FLT(tFloat fltIn,
GDFLIB_FILTER_IIR2_T_FLT *const pParam);
```

2.15.5.2 Arguments

Table 119. GDFLIB_FilterIIR2_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Value of the input signal to be filtered in step (k). Input is a 32-bit number that contains a single precision floating point value.
GDFLIB_FILTER_IIR2_T_FLT *const	pParam	**input, output**	Pointer to a filter structure with a filter buffer and filter parameters. Arguments of the structure contain single precision floating point values.

2.15.5.3 Return

The function returns a 32-bit value in single precision floating point format, representing the filtered value of the input signal in step *k*.

2.15.5.4 Implementation details

The coefficients can be calculated using Matlab® as follows:

```
freq_bot    = 400;
freq_top    = 625;
T_sampling  = 100e-6;

[b,a]= butter(1,[freq_bot freq_top]*T_sampling *2, 'bandpass');
sys =tf(b,a,T_sampling);
bode(sys,[freq_bot:1:freq_top]*2*pi)

fltB0 = b(1);
fltB1 = b(2);
fltB2 = b(3);
fltA1 = a(2);
fltA2 = a(3);
disp (' Coefficients for GDFLIB_FilterIIR2 function :');
disp ([ 'fltB0 = (tFloat)(' num2str( fltB0,'%1.15f' ) ');']);
disp ([ 'fltB1 = (tFloat)(' num2str( fltB1,'%1.15f' ) ');']);
disp ([ 'fltB2 = (tFloat)(' num2str( fltB2,'%1.15f' ) ');']);
disp ([ 'fltA1 = (tFloat)(' num2str( fltA1,'%1.15f' ) ');']);
disp ([ 'fltA2 = (tFloat)(' num2str( fltA2,'%1.15f' ) ');']);
```

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

CAUTION: *The filter delay line includes four delay buffers which should be reset after filter initialization. This can be done by assigning to the filter instance a GDFLIB_FILTER_IIR2_DEFAULT_FLT macro during instance declaration or by calling the GDFLIB_FilterIIR2Init_FLT function.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.15.5.5 Code Example

```
#include "gdflib.h"

tFloat fltIn;
tFloat fltOut;

GDFLIB_FILTER_IIR2_T_FLT flttrMyIIR2 = GDFLIB_FILTER_IIR2_DEFAULT_FLT;

void main(void)
{
    // input value = 0.25
    fltIn = (tFloat)(0.25);

    // filter coefficients (BPF 400-625Hz, Ts=100e-6)
    flttrMyIIR2.trFiltCoeff.fltB0 = (tFloat)(0.066122101544579);
    flttrMyIIR2.trFiltCoeff.fltB1 = (tFloat)(0.0);
    flttrMyIIR2.trFiltCoeff.fltB2 = (tFloat)(-0.066122101544579);
    flttrMyIIR2.trFiltCoeff.fltA1 = (tFloat)(-1.776189018043779);
    flttrMyIIR2.trFiltCoeff.fltA2 = (tFloat)(0.867755796910841);

    // output should be 0.01651
    GDFLIB_FilterIIR2Init_FLT(&flttrMyIIR2);
    fltOut = GDFLIB_FilterIIR2_FLT(fltIn, &flttrMyIIR2);

    // output should be 0.01651
    GDFLIB_FilterIIR2Init(&flttrMyIIR2, FLT);
    fltOut = GDFLIB_FilterIIR2(fltIn, &flttrMyIIR2, FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be 0.01651
    GDFLIB_FilterIIR2Init(&flttrMyIIR2);
    fltOut = GDFLIB_FilterIIR2(fltIn, &flttrMyIIR2);
}
```

2.15.6 Function GDFLIB_FilterIIR2Init

This function initializes the second order IIR filter buffers.

2.15.6.1 Description

This function clears the internal buffers of a second order IIR filter. It shall be called after filter parameter initialization and whenever the filter initialization is required.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

Note: *This function shall not be called together with the GDFLIB_FilterIIR2 function unless periodic clearing of filter buffers is required.*

2.15.6.2 Re-entrancy

The function is re-entrant for a different `pParam`.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.15.6.3 Function GDFLIB_FilterIIR2Init_F32

2.15.6.3.1 Declaration

```
void GDFLIB_FilterIIR2Init_F32(GDFLIB_FILTER_IIR2_T_F32 *const
pParam);
```

2.15.6.3.2 Arguments

Table 120. GDFLIB_FilterIIR2Init_F32 arguments

Type	Name	Direction	Description
GDFLIB_FILTER_IIR2_T_F32 *const	pParam	**input, output**	Pointer to the filter structure with a filter buffer and filter parameters.

2.15.6.3.3 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_IIR2_T_F32 f32trMyIIR2 = GDFLIB_FILTER_IIR2_DEFAULT_F32;

void main(void)
{
    // function returns no value
    GDFLIB_FilterIIR2Init_F32(&f32trMyIIR2);

    // function returns no value
    GDFLIB_FilterIIR2Init(&f32trMyIIR2, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // function returns no value
    GDFLIB_FilterIIR2Init(&f32trMyIIR2);
}
```

2.15.6.4 Function GDFLIB_FilterIIR2Init_F16

2.15.6.4.1 Declaration

```
void GDFLIB_FilterIIR2Init_F16(GDFLIB_FILTER_IIR2_T_F16 *const
pParam);
```

2.15.6.4.2 Arguments

Table 121. GDFLIB_FilterIIR2Init_F16 arguments

Type	Name	Direction	Description
GDFLIB_FILTER_IIR2_T_F16 *const	pParam	**input, output**	Pointer to the filter structure with a filter buffer and filter parameters.

2.15.6.4.3 Code Example

```
#include "gdflib.h"
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
GDFLIB_FILTER_IIR2_T_F16 f16trMyIIR2 = GDFLIB_FILTER_IIR2_DEFAULT_F16;

void main(void)
{
    // function returns no value
    GDFLIB_FilterIIR2Init_F16(&f16trMyIIR2);

    // function returns no value
    GDFLIB_FilterIIR2Init(&f16trMyIIR2, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // function returns no value
    GDFLIB_FilterIIR2Init(&f16trMyIIR2);
}
```

2.15.6.5 Function GDFLIB_FilterIIR2Init_FLT

2.15.6.5.1 Declaration

```
void GDFLIB_FilterIIR2Init_FLT(GDFLIB_FILTER_IIR2_T_FLT *const
pParam);
```

2.15.6.5.2 Arguments

Table 122. GDFLIB_FilterIIR2Init_FLT arguments

Type	Name	Direction	Description
GDFLIB_FILTER_IIR2_T_FLT *const	pParam	**input, output**	Pointer to a filter structure with filter buffer and filter parameters. Arguments of the structure contain single precision floating point values.

Note: *The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.15.6.5.3 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_IIR2_T_FLT flttrMyIIR2 = GDFLIB_FILTER_IIR2_DEFAULT_FLT;

void main(void)
{
    // function returns no value
    GDFLIB_FilterIIR2Init_FLT(&flttrMyIIR2);

    // function returns no value
    GDFLIB_FilterIIR2Init(&flttrMyIIR2, FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // function returns no value
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    GDFLIB_FilterIIR2Init(&flttrMyIIR2);
}
```

2.16 Function GDFLIB_FilterMA

This function implements an exponential moving average filter.

2.16.1 Description

This function calculates one iteration of an exponential moving average filter (also known as the exponentially weighted moving average, EWMA). The filter is characterized by the following difference equation:

$$y(k) = \lambda \cdot x(k) + (1 - \lambda) \cdot y(k-1)$$

Equation GDFLIB_FilterMA_Eq1

where x(k) is the filter input, y(k) is the filter output in the current step, y(k-1) is the filter output of the previous step and λ is a smoothing factor, 0 < λ < 1. Values of λ close to one lead to less smoothing and give greater weight to recent changes in the input data, while values of λ closer to zero cause greater smoothing and the filter is less responsive to recent changes.

There is no direct equivalence between the smoothing factor λ of the exponential moving average filter and the number of averaged samples N of a uniform sliding-window moving average filter. Nevertheless, the implementation uses the following approximation to relate the two filtering approaches:

$$\lambda = \frac{1}{N}$$

Equation GDFLIB_FilterMA_Eq2

When λ is set according to the above formula, the amplitude signal-to-noise ratio improvement achievable by both types of filters is the same.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.16.2 Re-entrancy

The function is re-entrant for a different `pParam`.

2.16.3 Function GDFLIB_FilterMA_F32

2.16.3.1 Declaration

```
tFrac32 GDFLIB_FilterMA_F32(tFrac32 f32In, GDFLIB_FILTER_MA_T_F32
*pParam);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.16.3.2 Arguments

Table 123. GDFLIB_FilterMA_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Value of input signal to be filtered in step (k). The value is a 32-bit number in the Q1.31 format.
GDFLIB_FILTER_MA_T_F32 *	pParam	**input, output**	Pointer to the filter structure with a filter accumulator and a smoothing factor.

2.16.3.3 Return

The function returns a 32-bit value in format Q1.31, representing the filtered value of the input signal in step (k).

2.16.3.4 Implementation details

The library function expects the smoothing factor to be supplied in the form of the u16NSamples variable stored within the filter structure. This variable represents the binary logarithm of the number of averaged samples N of the corresponding uniform sliding-window moving average filter:

$$N = 2^{u16NSamples} \quad 0 \le u16NSamples \le 31$$

Equation GDFLIB_FilterMA_F32_Eq1

Note: *The recalculated smoothing factor u16NSamples needs to be defined prior to calling this function and must be equal to or greater than 0, and equal to or smaller than 31. Incorrect setting of this parameter will yield meaningless results.*

2.16.3.5 Code Example

```
#include "gdflib.h"

tFrac32 f32Input;
tFrac32 f32Output;

GDFLIB_FILTER_MA_T_F32 f32trMyMA = GDFLIB_FILTER_MA_DEFAULT_F32;

void main(void)
{
  // input value = 0.25
  f32Input = FRAC32(0.25);

  // filter window = 2^5 = 32 samples
  f32trMyMA.u16NSamples = 5;
  GDFLIB_FilterMAInit_F32(&f32trMyMA);

  // output should be 0x1000000 = FRAC32(0.0078125)
  f32Output = GDFLIB_FilterMA_F32(f32Input,&f32trMyMA);

  // output should be 0x1000000 = FRAC32(0.0078125)
  f32Output = GDFLIB_FilterMA(f32Input,&f32trMyMA,F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // ##########################################################

  // output should be 0x1000000 = FRAC32(0.0078125)
  f32Output = GDFLIB_FilterMA(f32Input,&f32trMyMA);
}
```

2.16.4 Function GDFLIB_FilterMA_F16

2.16.4.1 Declaration

```
tFrac16 GDFLIB_FilterMA_F16(tFrac16 f16In, GDFLIB_FILTER_MA_T_F16
*pParam);
```

2.16.4.2 Arguments

Table 124. GDFLIB_FilterMA_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Value of input signal to be filtered in step (k). The value is a 16-bit number in the Q1.15 format.
GDFLIB_FILTER_MA_T_F16 *	pParam	**input, output**	Pointer to the filter structure with a filter accumulator and a smoothing factor.

2.16.4.3 Return

The function returns a 16-bit value in format Q1.15, representing the filtered value of the input signal in step (k).

2.16.4.4 Implementation details

The library function expects the smoothing factor to be supplied in the form of the u16NSamples variable stored within the filter structure. This variable represents the binary logarithm of the number of averaged samples N of the corresponding uniform sliding-window moving average filter:

$$N = 2^{u16NSamples} \quad 0 \le u16NSamples \le 15$$

Equation GDFLIB_FilterMA_F16_Eq1

Note: *The recalculated smoothing factor u16NSamples needs to be defined prior to calling this function and must be equal to or greater than 0, and equal to or smaller than 16. Incorrect setting of this parameter will yield meaningless results. In case the filter window size is greater than 8, the output error may exceed the guaranteed range.*

2.16.4.5 Code Example

```
#include "gdflib.h"

tFrac16 f16Input;
tFrac16 f16Output;

GDFLIB_FILTER_MA_T_F16 f16trMyMA = GDFLIB_FILTER_MA_DEFAULT_F16;

void main(void)
{
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // input value = 0.25
  f16Input = FRAC16(0.25);

  // filter window = 2^3 = 8 samples
  f16trMyMA.u16NSamples = 3;

  GDFLIB_FilterMAInit_F16(&f16trMyMA);

  // output should be 0x0400 = FRAC16(0.03125)
  f16Output = GDFLIB_FilterMA_F16(f16Input,&f16trMyMA);

  // output should be 0x0400 = FRAC16(0.03125)
  f16Output = GDFLIB_FilterMA(f16Input,&f16trMyMA,F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be 0x0400 = FRAC16(0.03125)
  f16Output = GDFLIB_FilterMA(f16Input,&f16trMyMA);
}
```

2.16.5 Function GDFLIB_FilterMA_FLT

2.16.5.1 Declaration

```
tFloat GDFLIB_FilterMA_FLT(tFloat fltIn, GDFLIB_FILTER_MA_T_FLT
*pParam);
```

2.16.5.2 Arguments

Table 125. GDFLIB_FilterMA_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Value of the input signal to be filtered in step (k). The value is a single precision floating point data type.
GDFLIB_FILTER_MA_T_FLT *	pParam	**input, output**	Pointer to the filter structure with a filter accumulator and a smoothing factor.

2.16.5.3 Return

The function returns a single precision floating point value, representing the filtered value of the input signal in step (k).

2.16.5.4 Implementation details

Setting the smoothing factor according to the following formula will yield results equivalent to the fixed-point variants of the GDFLIB_FilterMA function:

$$\lambda = \frac{1}{N}$$

Equation GDFLIB_FilterMA_FLT_Eq1

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The smoothing factor λ need to be entered in the input parameter fltLambda prior to calling this function. If λ < 0 or λ > 1, the output values are meaningless numbers. If any of the inputs is a subnormal value, infinity, or a NaN, the filter output for the current iteration and all future iterations is meaningless until the filter is re-initialized.

2.16.5.5 Code Example

EXAMPLE 1: Using a smoothing factor calculation equivalent to the fixed-point variants of the GDFLIB_FilterMA function:

```
#include "gdflib.h"

tFloat fltInput;
tFloat fltOutput;

GDFLIB_FILTER_MA_T_FLT flttrMyMA = GDFLIB_FILTER_MA_DEFAULT_FLT;

void main(void)
{
  // input value = 0.25
  fltInput = (tFloat)(0.25);

  // filter window = 8 samples
  flttrMyMA.fltLambda = (tFloat)(1.0/8.0);

  GDFLIB_FilterMAInit_FLT(&flttrMyMA);

  // output should be 0.031250000
  fltOutput = GDFLIB_FilterMA_FLT(fltInput,&flttrMyMA);

  // output should be 0.031250000
  fltOutput = GDFLIB_FilterMA(fltInput,&flttrMyMA,FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output should be 0.031250000
  fltOutput = GDFLIB_FilterMA(fltInput,&flttrMyMA);
}
```

EXAMPLE 2: Using a smoothing factor calculation to achieve the same signal-to-noise ratio improvement as if a uniform sliding-window moving average filter was used:

```
#include "gdflib.h"

tFloat fltInput;
tFloat fltOutput;

GDFLIB_FILTER_MA_T_FLT flttrMyMA = GDFLIB_FILTER_MA_DEFAULT_FLT;

void main(void)
{
  // input value = 0.25
  fltInput = (tFloat)(0.25);

  // filter window = 8 samples
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  flttrMyMA.fltLambda = (tFloat)(2.0/(8.0 + 1.0));

  GDFLIB_FilterMAInit_FLT(&flttrMyMA);

  // output should be 0.055555556
  fltOutput = GDFLIB_FilterMA_FLT(fltInput,&flttrMyMA);

  // output should be 0.055555556
  fltOutput = GDFLIB_FilterMA(fltInput,&flttrMyMA,FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output should be 0.055555556
  fltOutput = GDFLIB_FilterMA(fltInput,&flttrMyMA);
}
```

2.16.6 Function GDFLIB_FilterMAInit

2.16.6.1 Description

This function clears the internal accumulator of a moving average filter. It shall be called after filter parameter initialization and whenever the filter initialization is required.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

Note: *This function shall not be called together with the GDFLIB_FilterIIR1 function unless periodic clearing of filter buffers is required.*

Note: *This function shall not be called together with the GDFLIB_FilterMA function unless periodic clearing of filter buffers is required.*

2.16.6.2 Re-entrancy

The function is re-entrant for a different `pParam`.

2.16.6.3 Function GDFLIB_FilterMAInit_F32

2.16.6.3.1 Declaration

```
void GDFLIB_FilterMAInit_F32(GDFLIB_FILTER_MA_T_F32 *pParam);
```

2.16.6.3.2 Arguments

Table 126. GDFLIB_FilterMAInit_F32 arguments

Type	Name	Direction	Description
GDFLIB_FILTER_MA_T_F32 *	pParam	**input, output**	Pointer to the filter structure with a filter accumulator and a smoothing factor.

2.16.6.3.3 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_MA_T_F32 f32trMyMA = GDFLIB_FILTER_MA_DEFAULT_F32;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
void main(void)
{
  // filter window = 2^5 = 32 samples
  f32trMyMA.u16NSamples = 5;

  GDFLIB_FilterMAInit_F32(&f32trMyMA);

  GDFLIB_FilterMAInit(&f32trMyMA,F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  GDFLIB_FilterMAInit(&f32trMyMA);
}
```

2.16.6.4 Function GDFLIB_FilterMAInit_F16

2.16.6.4.1 Declaration

```
void GDFLIB_FilterMAInit_F16(GDFLIB_FILTER_MA_T_F16 *pParam);
```

2.16.6.4.2 Arguments

Table 127. GDFLIB_FilterMAInit_F16 arguments

Type	Name	Direction	Description
GDFLIB_FILTER_MA_T_F16 *	pParam	**input, output**	Pointer to the filter structure with a filter accumulator and a smoothing factor.

2.16.6.4.3 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_MA_T_F16 f16trMyMA = GDFLIB_FILTER_MA_DEFAULT_F16;

void main(void)
{
  // filter window = 2^3 = 8 samples
  f16trMyMA.u16NSamples = 3;

  GDFLIB_FilterMAInit_F16(&f16trMyMA);

  GDFLIB_FilterMAInit(&f16trMyMA,F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  GDFLIB_FilterMAInit(&f16trMyMA);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.16.6.5 Function GDFLIB_FilterMAInit_FLT

2.16.6.5.1 Declaration

```
void GDFLIB_FilterMAInit_FLT(GDFLIB_FILTER_MA_T_FLT *pParam);
```

2.16.6.5.2 Arguments

Table 128. GDFLIB_FilterMAInit_FLT arguments

Type	Name	Direction	Description
GDFLIB_FILTER_MA_T_FLT *	pParam	**input, output**	Pointer to a filter structure with a filter accumulator and a smoothing factor.

Note: *The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.16.6.5.3 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_MA_T_FLT flttrMyMA = GDFLIB_FILTER_MA_DEFAULT_FLT;

void main(void)
{
  // filter window = 8 samples (using smoothing factor calculation
  // equivalent to the fixed-point implementation of GDFLIB_FilterMA)
  flttrMyMA.fltLambda = (tFloat)(1.0/8.0);

  GDFLIB_FilterMAInit_FLT(&flttrMyMA);

  GDFLIB_FilterMAInit(&flttrMyMA,FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  GDFLIB_FilterMAInit(&flttrMyMA);
}
```

2.16.7 Function GDFLIB_FilterMASetState

2.16.7.1 Description

This function initializes the internal accumulator of a moving average filter to achieve the required output value.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

Note: *This function should not be called periodically together with the GDFLIB_FilterMA function unless periodic setting of the filter accumulator is required.*

2.16.7.2 Re-entrancy

The function is re-entrant for a different `pParam`.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.16.7.3 Function GDFLIB_FilterMASetState_F32

2.16.7.3.1 Declaration

```
void GDFLIB_FilterMASetState_F32(tFrac32 f32FilterMAOut,
GDFLIB_FILTER_MA_T_F32 *pParam);
```

2.16.7.3.2 Arguments

Table 129. GDFLIB_FilterMASetState_F32 arguments

Type	Name	Direction	Description
tFrac32	f32FilterMAOut	**input**	Required output of the FilterMA.
GDFLIB_FILTER_MA_T_F32 *	pParam	**input, output**	Pointer to the filter structure with a filter accumulator and a smoothing factor.

2.16.7.3.3 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_MA_T_F32 f32trMyMA = GDFLIB_FILTER_MA_DEFAULT_F32;

void main(void)
{
  // filter window = 2^5 = 32 samples
  f32trMyMA.u16NSamples = 5;
  tFrac32 f32FilterMAOut;

  // Initialize the GDFLIB_FilterMA state variable to a predefined value
  // Warning: The u16NSamples parameter in f32trMyMA must be already
  // initialized.
  f32FilterMAOut = (tFrac32)123L; // required output value
  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GDFLIB_FilterMASetState_F32(f32FilterMAOut, &f32trMyMA);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GDFLIB_FilterMASetState(f32FilterMAOut, &f32trMyMA, F32);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GDFLIB_FilterMASetState(f32FilterMAOut, &f32trMyMA);
}
```

2.16.7.4 Function GDFLIB_FilterMASetState_F16

2.16.7.4.1 Declaration

```
void GDFLIB_FilterMASetState_F16(tFrac16 f16FilterMAOut,
GDFLIB_FILTER_MA_T_F16 *pParam);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.16.7.4.2 Arguments

Table 130. GDFLIB_FilterMASetState_F16 arguments

Type	Name	Direction	Description
tFrac16	f16FilterMAOut	**input**	Required output of the FilterMA.
GDFLIB_FILTER_MA_T_F16 *	pParam	**input, output**	Pointer to the filter structure with a filter accumulator and a smoothing factor.

2.16.7.4.3 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_MA_T_F16 f16trMyMA = GDFLIB_FILTER_MA_DEFAULT_F16;

void main(void)
{
  // filter window = 2^5 = 32 samples
  f16trMyMA.u16NSamples = 5;
  tFrac16 f16FilterMAOut;

  // Initialize the GDFLIB_FilterMA state variable to a predefined value
  // Warning: The u16NSamples parameter in f16trMyMA must be already
  // initialized.
  f16FilterMAOut = (tFrac16)123; // required output value
  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GDFLIB_FilterMASetState_F16(f16FilterMAOut, &f16trMyMA);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GDFLIB_FilterMASetState(f16FilterMAOut, &f16trMyMA, F16);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 16-bit fractional implementation is selected as default.
  GDFLIB_FilterMASetState(f16FilterMAOut, &f16trMyMA);
}
```

2.16.7.5 Function GDFLIB_FilterMASetState_FLT

2.16.7.5.1 Declaration

```
void GDFLIB_FilterMASetState_FLT(tFloat fltFilterMAOut,
GDFLIB_FILTER_MA_T_FLT *pParam);
```

2.16.7.5.2 Arguments

Table 131. GDFLIB_FilterMASetState_FLT arguments

Type	Name	Direction	Description
tFloat	fltFilterMAOut	**input**	Required output of the FilterMA.
GDFLIB_FILTER_MA_T_FLT *	pParam	**input, output**	Pointer to the filter structure with a filter accumulator and a smoothing factor.

2.16.7.5.3 Code Example

```
#include "gdflib.h"
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
GDFLIB_FILTER_MA_T_FLT flttrMyMA = GDFLIB_FILTER_MA_DEFAULT_FLT;

void main(void)
{
  flttrMyMA.fltLambda = 0.5f;
  tFloat fltFilterMAOut;

  // Initialize the GDFLIB_FilterMA state variable to a predefined value
  // Warning: The fltLambda parameter in flttrMyMA must be already
  // initialized.
  fltFilterMAOut = 123.0f; // required output value
  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GDFLIB_FilterMASetState_FLT(fltFilterMAOut, &flttrMyMA);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GDFLIB_FilterMASetState(fltFilterMAOut, &flttrMyMA, FLT);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if single precision floating point implementation is selected as default.
  GDFLIB_FilterMASetState(fltFilterMAOut, &flttrMyMA);
}
```

2.17 Function GFLIB_Acos

This function implements an approximation of arccosine function.

2.17.1 Description

The GFLIB_Acos function provides a computational method for calculation of the standard inverse trigonometric *arccosine* function arccos(x), using the piece-wise polynomial approximation. Function arccos(x) takes the ratio of the length of the adjacent side to the length of the hypotenuse and returns the angle.

Figure 85. Course of the function GFLIB_Acos

2.17.2 Re-entrancy

The function is re-entrant.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.17.3 Function GFLIB_Acos_F32

2.17.3.1 Declaration

```
tFrac32 GFLIB_Acos_F32(tFrac32 f32In, const GFLIB_ACOS_T_F32
*const pParam);
```

2.17.3.2 Arguments

Table 132. GFLIB_Acos_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument is a 32-bit number that contains a value between [-1,1).
const GFLIB_ACOS_T_F32 *const	pParam	**input**	Pointer to an array of Taylor coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ACOS_DEFAULT_F32 symbol.

2.17.3.3 Return

The function returns arccos(f32In)/ π as a fixed point 32-bit number, normalized between [0,1).

2.17.3.4 Implementation details

The computational algorithm uses the relation between arccosine and arcsine function. At first the arcsin(x) functional value is computed and then the result is corrected by following formula:

$$\cos^{-1}(x) = \frac{\pi}{2} - \sin^{-1}(x)$$

Equation GFLIB_Acos_F32_Eq1

The computation of arcsin(x) uses the symmetry of the function around the point (0, 0), which allows to for computing the function values just in the interval [0, 1) and to compute the function values in the interval [-1, 0) by the simple formula:

$$\sin^{-1}(-x) = -\sin^{-1}(x) \quad x \epsilon [0,\ 1]$$

Equation GFLIB_Acos_F32_Eq2

Additionally, because the arcsin(x) function is difficult for polynomial approximation for x approaching 1 (or -1 by symmetry), due to its derivatives approaching infinity, a special transformation is used to transform the range of x from [0.5, 1) to (0, 0.5]:

$$\sin^{-1}\left(\sqrt{1-x}\right) = \frac{\pi}{2} - \sin^{-1}\left(\sqrt{x}\right)$$

Equation GFLIB_Acos_F32_Eq3

In this way, the computation of the arcsin(x) function in the range [0.5, 1) can be replaced by the computation in the range (0, 0.5], in which approximation is easier.

Moreover for interval [0.997, 1), different approximation coeficients are used to eliminate the imprecision of the polynom in this range.

For the interval (0, 0.5], the algorithm uses polynomial approximation as follows:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$\frac{\sin^{-1}(fIn)}{\pi} = \left(a_0 + a_1 \cdot |fIn| + a_2 \cdot |fIn|^2 + a_3 \cdot |fIn|^3 + a_4 \cdot |fIn|^4\right) \cdot sign\left(fIn\right)$$

Equation GFLIB_Acos_F32_Eq4

The division of the [0,1) interval into three sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in Table 133 .

Table 133. Integer polynomial coefficients for each interval

Interval	a_0	a_1	a_2	a_3	a_4
<0 , 1/2)	12751	682829947	9729967	66340080	91918582
<1/2, 0.997)	1073630175	-964576326	-15136243	-36708911	-52453538
<0.997, 1)	1073739175	-966167437	-15136243	-36708911	-52453538

Until this point the implementation of the GFLIB_Acos_F32 is the same as in the function GFLIB_Asin_F32. As last step the output of the GFLIB_Acos_F32 is corrected as follows:

$$fOut = \frac{\cos^{-1}(fIn)}{\pi} = \frac{1}{2} - \frac{\sin^{-1}(fIn)}{\pi}$$

Equation GFLIB_Acos_F32_Eq5

The polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of 5th order was used for the fitting of each respective sub-interval. The functions *arcsine* and *arccosine* are similar, therefore the GFLIB_Acos_F32 function uses the same polynomial coefficients as he GFLIB_Asin_F32 function.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.17.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32Input;
tFrac32 f32Angle;

void main(void)
{
    // input f32Input = 0
    f32Input  = FRAC32(0);

    // output should be 0x400031CE = 0.5 => pi/2
    f32Angle = GFLIB_Acos_F32(f32Input, GFLIB_ACOS_DEFAULT_F32);

    // output should be 0x400031CE = 0.5 => pi/2
    f32Angle = GFLIB_Acos(f32Input, GFLIB_ACOS_DEFAULT_F32, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x400031CE = 0.5 => pi/2
    f32Angle = GFLIB_Acos(f32Input);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.17.4 Function GFLIB_Acos_F16

2.17.4.1 Declaration

```
tFrac16 GFLIB_Acos_F16(tFrac16 f16In, const GFLIB_ACOS_T_F16
*const pParam);
```

2.17.4.2 Arguments

Table 134. GFLIB_Acos_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument is a 16-bit number that contains a value between [-1,1).
const GFLIB_ACOS_T_F16 *const	pParam	**input**	Pointer to an array of Taylor coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ACOS_DEFAULT_F16 symbol.

2.17.4.3 Return

The function returns arccos(f16In)/ π as a fixed point 16-bit number, normalized between [0,1).

2.17.4.4 Implementation details

The computational algorithm uses the relation between arccosine and arcsine function. At first the arcsin(x) functional value is computed and then the result is corrected by following formula:

$$\cos^{-1}(x) = \frac{\pi}{2} - \sin^{-1}(x)$$

Equation GFLIB_Acos_F16_Eq1

The computation of arcsin(x) uses the symmetry of the function around the point (0, 0), which allows to for computing the function values just in the interval [0, 1) and to compute the function values in the interval [-1, 0) by the simple formula:

$$\sin^{-1}(-x) = -\sin^{-1}(x) \quad x \epsilon [0,\ 1]$$

Equation GFLIB_Acos_F16_Eq2

Additionally, because the arcsin(x) function is difficult for polynomial approximation for x approaching 1 (or -1 by symmetry), due to its derivatives approaching infinity, a special transformation is used to transform the range of x from [0.5, 1) to (0, 0.5]:

$$\sin^{-1}\left(\sqrt{1-x}\right) = \frac{\pi}{2} - \sin^{-1}\left(\sqrt{x}\right)$$

Equation GFLIB_Acos_F16_Eq3

In this way, the computation of the arcsin(x) function in the range [0.5, 1) can be replaced by the computation in the range (0, 0.5], in which approximation is easier.

For the interval (0, 0.5], the algorithm uses polynomial approximation as follows:

$$\frac{\sin^{-1}(fIn)}{\pi} = \left(a_0 + a_1 \cdot |fIn| + a_2 \cdot |fIn|^2 + a_3 \cdot |fIn|^3 + a_4 \cdot |fIn|^4\right) \cdot sign\left(fIn\right)$$

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Equation GFLIB_Acos_F16_Eq4

The division of the [0,1) interval into two sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in Table 135 .

Table 135. Integer polynomial coefficients for each interval

Interval	a_0	a_1	a_2	a_3	a_4
<0 , 1/2)	1	10419	148	1012	1403
<1/2, 1)	16384	-14718	-231	-560	-800

Until this point the implementation of the GFLIB_Acos_F16 is the same as in the function GFLIB_Asin_F16. As last step the output of the GFLIB_Acos_F16 is corrected as follows:

$$fOut = \frac{\cos^{-1}(fIn)}{\pi} = \frac{1}{2} - \frac{\sin^{-1}(fIn)}{\pi}$$

Equation GFLIB_Acos_F16_Eq5

The polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of 5th order was used for the fitting of each respective sub-interval. The functions *arcsine* and *arccosine* are similar, therefore the GFLIB_Acos_F16 function uses the same polynomial coefficients as the GFLIB_Asin_F16 function.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.17.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16Input;
tFrac16 f16Angle;

void main(void)
{
  // input f16Input = 0
  f16Input  = FRAC16(0);

  // output should be 0x4000 = 0.5 => pi/2
  f16Angle = GFLIB_Acos_F16(f16Input, GFLIB_ACOS_DEFAULT_F16);

  // output should be 0x4000 = 0.5 => pi/2
  f16Angle = GFLIB_Acos(f16Input, GFLIB_ACOS_DEFAULT_F16, F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be 0x4000 = 0.5 => pi/2
  f16Angle = GFLIB_Acos(f16Input);
}
```


© NXP B.V. 2023. All rights reserved.

2.17.5 Function GFLIB_Acos_FLT

2.17.5.1 Declaration

```
tFloat GFLIB_Acos_FLT(tFloat fltIn, const GFLIB_ACOS_T_FLT *const
pParam);
```

2.17.5.2 Arguments

Table 136. GFLIB_Acos_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument is a 32-bit number that contains a single precision floating point value.
const GFLIB_ACOS_T_FLT *const	pParam	**input**	Pointer to an array of approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ACOS_DEFAULT_FLT symbol.

2.17.5.3 Return

The function returns arccos(fltIn) as a single precision floating point number.

2.17.5.4 Implementation details

The computational algorithm uses the relation between arccosine and arcsine function. At first the arcsin(x) functional value is computed and then the result is corrected by following formula:

$$\cos^{-1}(x) = \frac{\pi}{2} - \sin^{-1}(x)$$

Equation GFLIB_Acos_FLT_Eq1

The computation algorithm for arcsin(x) uses the symmetry of the function around the point (0, 0), which allows the calculation of the function values just in interval [0, 1), and to calculate the function values in the interval [-1, 0) use the simple formula:

$$\sin^{-1}(-x) = -\sin^{-1}(x) \quad x \epsilon [0,\ 1]$$

Equation GFLIB_Acos_FLT_Eq2

For the interval [0, 1], the algorithm uses a polynomial approximation as follows:

$$\sin^{-1}(fIn) = \left[\frac{\pi}{2} - \sqrt{1 - fIn} \cdot \left(a_0 + a_1 \cdot |fIn| + a_2 \cdot |fIn|^2 + a_3 \cdot |fIn|^3\right)\right] \cdot sign(fIn)$$

Equation GFLIB_Acos_FLT_Eq3

Polynomial approximation coefficients used for GFLIB_Acos_FLT calculation are noted in Table 137 .

Table 137. Approximation polynomial coefficients

a_0	a_1	a_2	a_3	a_4
1.5707811e+00	-2.1389899e-01	8.3511069e-02	-3.3877850e-02	7.6983864e-03

In order to get functional value of arccosine function, output of GFLIB_Acos_FLT_Eq3 must be corrected by GFLIB_Acos_FLT_Eq1.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

The function may raise floating-point exceptions (invalid operation, division by zero, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.

2.17.5.5 Code Example

```
#include "gflib.h"

tFloat fltInput;
tFloat fltAngle;

void main(void)
{
  // input fltInput = 0
  fltInput  = 0;

  // output should be 1.570796
  fltAngle = GFLIB_Acos_FLT(fltInput, GFLIB_ACOS_DEFAULT_FLT);

  // output should be 1.570796
  fltAngle = GFLIB_Acos(fltInput, GFLIB_ACOS_DEFAULT_FLT, FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output should be 1.570796
  fltAngle = GFLIB_Acos(fltInput);
}
```

2.18 Function GFLIB_Asin

This function implements polynomial approximation of arcsine function.

2.18.1 Description

The GFLIB_Asin function provides a computational method for calculation of a standard inverse trigonometric *arcsine* function arcsin(x), using the piece-wise polynomial approximation. Function arcsin(x) takes the ratio of the length of the opposite side to the length of the hypotenuse and returns the angle.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 86. Course of the function GFLIB_Asin

2.18.2 Re-entrancy

The function is re-entrant.

2.18.3 Function GFLIB_Asin_F32

2.18.3.1 Declaration

```
tFrac32 GFLIB_Asin_F32(tFrac32 f32In, const GFLIB_ASIN_T_F32
*const pParam);
```

2.18.3.2 Arguments

Table 138. GFLIB_Asin_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument is a 32-bit number that contains a value between [-1,1).
const GFLIB_ASIN_T_F32 *const	pParam	**input**	Pointer to an array of Taylor coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ASIN_DEFAULT_F32 symbol.

2.18.3.3 Return

The function returns arcsin(f32In)/ π as a fixed point 32-bit number, normalized between [-0.5,0.5).

2.18.3.4 Implementation details

The computational algorithm uses the symmetry of the arcsin(x) function around the point (0, 0), which allows to for computing the function values just in the interval [0, 1) and to compute the function values in the interval [-1, 0) by the simple formula:

$$\sin^{-1}(-x) = -\sin^{-1}(x) \quad x \epsilon [0,\ 1]$$

Equation GFLIB_Asin_F32_Eq1

Additionally, because the arcsin(x) function is difficult for polynomial approximation for x approaching 1 (or -1 by symmetry), due to its derivatives approaching infinity, a special transformation is used to transform the range of x from [0.5, 1) to (0, 0.5]:

$$\sin^{-1}\left(\sqrt{1-x}\right) = \frac{\pi}{2} - \sin^{-1}\left(\sqrt{x}\right)$$

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Equation GFLIB_Asin_F32_Eq2

In this way, the computation of the arcsin(x) function in the range [0.5, 1) can be replaced by the computation in the range (0, 0.5], in which approximation is easier.

Moreover for interval [0.997, 1), different approximation coeficients are used to eliminate the imprecision of the polynom in this range.

For the interval (0, 0.5], the algorithm uses polynomial approximation as follows:

$$fOut = \frac{\sin^{-1}(fIn)}{\pi} = \left(a_0 + a_1 \cdot |fIn| + a_2 \cdot |fIn|^2 + a_3 \cdot |fIn|^3 + a_4 \cdot |fIn|^4\right) \cdot sign\left(fIn\right)$$

Equation GFLIB_Asin_F32_Eq3

The division of the [0,1) interval into two sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in Table 139 .

Table 139. Integer polynomial coefficients for each interval

Interval	a_0	a_1	a_2	a_3	a_4
<0 , 1/2)	12751	682829947	9729967	66340080	91918582
<1/2, 0.997)	1073630175	-964576326	-15136243	-36708911	-52453538
<0.997, 1)	1073739175	-966167437	-15136243	-36708911	-52453538

Polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of the 5th order was used for the fitting of each respective sub-interval. The Matlab was used as follows:

```
clear all
clc

number_of_range = 2;
i = 1;
range = 0;

    Range = 1 / number_of_range;
    x(i,:) = (((i-1)*Range):(1/(2^15)):((i)*Range))';
    y(i,:) = asin(x(i,:))/pi;
    p(i,:) = polyfit((x(i,:)),(y(i,:)),4);

    i=i+1;

    Range = 1 / number_of_range;
    x(i,:) = (((i-1)*Range):(1/(2^15)):((i)*Range))';
    y(i,:) = asin(x(i,:))/pi;
    x1(i,:) = ((x(i,:) - ((i-1)*Range)));
    x1(i,:) = 0.5 - x1(i,:);
    x2(i,:) = sqrt(x1(i,:));
    p(i,:) = polyfit((x2(i,:)),(y(i,:)),4);
    i=i+1;

f(2,:) = polyval(p(2,:),x2(2,:));
f(1,:) = polyval(p(1,:),x(1,:));
error_1 = abs(f(2,:) - y(2,:));
max(error_1 * (2^15))
error_2 = abs(f(1,:) - y(1,:));
max(error_2 * (2^15))
plot(x(2,:),y(2,:),'-',x(2,:),f(2,:),'-',x(1,:),y(1,:),'-',x(1,:),f(1,:),'-');
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
coef = round(p * (2^31))
```

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.18.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32Input;
tFrac32 f32Angle;

void main(void)
{
    // input f32Input = (1-(2^-31))
    f32Input  = (tFrac32)(0x7FFFFFFF);

    // output should be 0x3FFFF5A7 = 0.4999987665
    f32Angle = GFLIB_Asin_F32(f32Input, GFLIB_ASIN_DEFAULT_F32);

    // output should be 0x3FFFF5A7 = 0.4999987665
    f32Angle = GFLIB_Asin(f32Input, GFLIB_ASIN_DEFAULT_F32, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x3FFFF5A7 = 0.4999987665
    f32Angle = GFLIB_Asin(f32Input);
}
```

2.18.4 Function GFLIB_Asin_F16

2.18.4.1 Declaration

```
tFrac16 GFLIB_Asin_F16(tFrac16 f16In, const GFLIB_ASIN_T_F16
*const pParam);
```

2.18.4.2 Arguments

Table 140. GFLIB_Asin_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument is a 16-bit number that contains a value between [-1,1).
const GFLIB_ASIN_T_F16 *const	pParam	**input**	Pointer to an array of Taylor coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ASIN_DEFAULT_F16 symbol.

2.18.4.3 Return

The function returns arcsin(f16In)/ π as a fixed point 16-bit number, normalized between [-0.5,0.5).

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.18.4.4 Implementation details

The computational algorithm uses the symmetry of the arcsin(x) function around the point (0, 0), which allows to for computing the function values just in the interval [0, 1) and to compute the function values in the interval [-1, 0) by the simple formula:

$$\sin^{-1}(-x) = -\sin^{-1}(x) \quad x\epsilon[0,\ 1]$$

Equation GFLIB_Asin_F16_Eq1

Additionally, because the arcsin(x) function is difficult for polynomial approximation for x approaching 1 (or -1 by symmetry), due to its derivatives approaching infinity, a special transformation is used to transform the range of x from [0.5, 1) to (0, 0.5]:

$$\sin^{-1}\left(\sqrt{1-x}\right) = \frac{\pi}{2} - \sin^{-1}\left(\sqrt{x}\right)$$

Equation GFLIB_Asin_F16_Eq2

In this way, the computation of the arcsin(x) function in the range [0.5, 1) can be replaced by the computation in the range (0, 0.5], in which approximation is easier.

For the interval (0, 0.5], the algorithm uses polynomial approximation as follows:

$$fOut = \frac{\sin^{-1}(fIn)}{\pi} = \left(a_0 + a_1 \cdot |fIn| + a_2 \cdot |fIn|^2 + a_3 \cdot |fIn|^3 + a_4 \cdot |fIn|^4\right) \cdot sign\left(fIn\right)$$

Equation GFLIB_Asin_F16_Eq3

The division of the [0,1) interval into two sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in Table 141 .

Table 141. Integer polynomial coefficients for each interval

Interval	a_0	a_1	a_2	a_3	a_4
<0 , 1/2)	1	10419	148	1012	1403
<1/2, 1)	16384	-14718	-231	-560	-800

Polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of the 5th order was used for the fitting of each respective sub-interval. The Matlab was used as follows:

```
clear all
clc

number_of_range = 2;
i = 1;
range = 0;

    Range = 1 / number_of_range;
    x(i,:) = (((i-1)*Range):(1/(2^15)):((i)*Range))';
    y(i,:) = asin(x(i,:))/pi;
    p(i,:) = polyfit((x(i,:)),(y(i,:)),4);

    i=i+1;

    Range = 1 / number_of_range;
    x(i,:) = (((i-1)*Range):(1/(2^15)):((i)*Range))';
    y(i,:) = asin(x(i,:))/pi;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    x1(i,:) = ((x(i,:) - ((i-1)*Range)));
    x1(i,:) = 0.5 - x1(i,:);
    x2(i,:) = sqrt(x1(i,:));
    p(i,:) = polyfit((x2(i,:)),(y(i,:)),4);
    i=i+1;

f(2,:) = polyval(p(2,:),x2(2,:));
f(1,:) = polyval(p(1,:),x(1,:));
error_1 = abs(f(2,:) - y(2,:));
max(error_1 * (2^15))
error_2 = abs(f(1,:) - y(1,:));
max(error_2 * (2^15))
plot(x(2,:),y(2,:),'-',x(2,:),f(2,:),'-',x(1,:),y(1,:),'-',x(1,:),f(1,:),'-');
coef = round(p * (2^31))
```

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.18.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16Input;
tFrac16 f16Angle;

void main(void)
{
    // input f16Input = (1-(2^-15))
    f16Input  = (tFrac16)(0x7FFF);

    // output should be 0x3FAE = 0.4974975
    f16Angle = GFLIB_Asin_F16(f16Input, GFLIB_ASIN_DEFAULT_F16);

    // output should be 0x3FAE = 0.4974975
    f16Angle = GFLIB_Asin(f16Input, GFLIB_ASIN_DEFAULT_F16, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x3FAE = 0.4974975
    f16Angle = GFLIB_Asin(f16Input);
}
```

2.18.5 Function GFLIB_Asin_FLT

2.18.5.1 Declaration

```
tFloat GFLIB_Asin_FLT(tFloat fltIn, const GFLIB_ASIN_T_FLT *const
pParam);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.18.5.2 Arguments

Table 142. GFLIB_Asin_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument is a 32-bit number that contains a single precision floating point value.
const GFLIB_ASIN_T_FLT *const	pParam	**input**	Pointer to an array of approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ASIN_DEFAULT_FLT symbol.

2.18.5.3 Return

The function returns arcsin(fltIn) as a single precision floating point number.

2.18.5.4 Implementation details

The computation algorithm for arcsin(x) uses the symmetry of the arcsin(x) function around the point (0, 0), which allows the calculation of the function values just in interval [0, 1), and to calculation of the function values in the interval [-1, 0), use the simple formula:

$$\sin^{-1}(-x) = -\sin^{-1}(x) \quad x \epsilon [0,\ 1]$$

Equation GFLIB_Asin_FLT_Eq1

For the interval [0, 0.41), the algorithm uses a Minimax approximation as follows:

$$\sin^{-1}(fIn) = \left(a_0 \cdot |fIn| + a_1 \cdot |fIn|^3 + a_2 \cdot |fIn|^5\right) \cdot sign(fIn)$$

Equation GFLIB_Asin_FLT_Eq2

For the interval [0.41, 1), the algorithm uses a polynomial approximation as follows.

$$\sin^{-1}(fIn) = \left[\frac{\pi}{2} - \sqrt{1 - fIn} \cdot \left(a_0 + a_1 \cdot |fIn| + a_2 \cdot |fIn|^2 + a_3 \cdot |fIn|^3\right)\right] \cdot sign(fIn)$$

Equation GFLIB_Asin_FLT_Eq3

Minimax approximation coefficients used for GFLIB_Asin_FLT calculation in interval [0, 0.41), are noted in Table 143 .

Table 143. Approximation polynomial coefficients

a_0	a_1	a_2
1.0000083e+00	1.6580229e-01	8.8028289e-02

In Table 144 there are the approximation coefficients used for GFLIB_Asin_FLT calculation in interval [0.41, 1).

Table 144. Approximation polynomial coefficients

a_0	a_1	a_2	a_3
1.5693614e+00	-2.0464350e-01	6.1917335e-02	-1.2426286e-02

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Note: *The function may raise floating-point exceptions (invalid operation, division by zero, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).

2.18.5.5 Code Example

```
#include "gflib.h"

tFloat fltInput;
tFloat fltAngle;

void main(void)
{
    // input fltInput = 1
    fltInput  = 1;

    // output should be 1.570796
    fltAngle = GFLIB_Asin_FLT(fltInput, GFLIB_ASIN_DEFAULT_FLT);

    // output should be 1.570796
    fltAngle = GFLIB_Asin(fltInput, GFLIB_ASIN_DEFAULT_FLT, FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be 1.570796
    fltAngle = GFLIB_Asin(fltInput);
}
```

2.19 Function GFLIB_Atan

This function implements minimax polynomial approximation of arctangent function.

2.19.1 Description

The GFLIB_Atan function provides a computational method for calculation of a standard trigonometric *arctangent* function arctan(x). Function arctan(x) takes a ratio and returns the angle of two sides of a right-angled triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle. The graph of arctan(x) is shown in Figure 87 .

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 87. Course of the function GFLIB_Atan

2.19.2 Re-entrancy

The function is re-entrant.

2.19.3 Function GFLIB_Atan_F32

2.19.3.1 Declaration

```
tFrac32 GFLIB_Atan_F32(tFrac32 f32In, const GFLIB_ATAN_T_F32
*const pParam);
```

2.19.3.2 Arguments

Table 145. GFLIB_Atan_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument is a 32-bit number between [-1,1).
const GFLIB_ATAN_T_F32 *const	pParam	**input**	Pointer to an array of minimax approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ATAN_DEFAULT_F32 symbol.

2.19.3.3 Return

The function returns the atan(f32In)/ π as a fixed point 32-bit number, normalized between [-0.25, 0.25).

2.19.3.4 Implementation details

The computational algorithm uses the symmetry of the arctangent function around the point (0, 0), which allows to for computing the function values just in the interval [0, 1) and to compute the function values in the interval [-1, 0) by the simple formula:

$$\tan^{-1}(-x) = -\tan^{-1}(x)$$

Equation GFLIB_Atan_F32_Eq1

The GFLIB_Atan_F32 function approximates the arctangent function using a piece-wise minimax polynomial approximation. The input range [0, 1) is divided into eight equally

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

spaced sub intervals, each with a distinct set of minimax coefficients. Negative inputs are calculated according to the antisymmetry of the function.

The GFLIB_Atan_F32 function uses fixed point fractional arithmetic, so to cast the fractional value of the output angle [-0.25, 0.25) into the correct range [- π/4, π/4), the fixed point output angle can be multiplied by π for an angle in radians. Then, the fixed point fractional implementation of the minimax approximation polynomial, used for calculation of each sub sector, is defined as follows:

$$fOut = \frac{\tan^{-1}(fIn)}{\pi} = \left(a_0 + a_1 \cdot |fIn| + a_2 \cdot |fIn|^2\right) \cdot sign\left(fIn\right)$$

Equation GFLIB_Atan_F32_Eq2

The division of the [0, 1) interval into eight sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in Table 146 .

Table 146. Integer minimax polynomial coefficients for each interval

Interval	a_0	a_1	a_2
<0, 1/8)	42667172	42515925	-164794
<1/8, 2/8)	126697014	41238272	-465182
<2/8, 3/8)	207041074	38899574	-690034
<3/8, 4/8)	281909001	35848645	-820713
<4/8, 5/8)	350251355	32453241	-865105
<5/8, 6/8)	411702516	29016149	-845462
<6/8, 7/8)	466407809	25743137	-786689
<7/8, 1)	514828039	22748418	-708969

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.19.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32Input;
tFrac32 f32Angle;

void main(void)
{
    // input ratio = 0x7FFFFFFF
    f32Input  = (tFrac32)(0x7FFFFFFF);

    // output angle should be 0x1FFFF29F = 0.249999 => pi/4
    f32Angle = GFLIB_Atan_F32(f32Input, GFLIB_ATAN_DEFAULT_F32);

    // output angle should be 0x1FFFF29F = 0.249999 => pi/4
    f32Angle = GFLIB_Atan(f32Input, GFLIB_ATAN_DEFAULT_F32, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output angle should be 0x1FFFF29F = 0.249999 => pi/4
    f32Angle = GFLIB_Atan(f32Input);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
}
```

2.19.4 Function GFLIB_Atan_F16

2.19.4.1 Declaration

```
tFrac16 GFLIB_Atan_F16(tFrac16 f16In, const GFLIB_ATAN_T_F16
*const pParam);
```

2.19.4.2 Arguments

Table 147. GFLIB_Atan_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument is a 16-bit number between [-1, 1).
const GFLIB_ATAN_T_F16 *const	pParam	**input**	Pointer to an array of minimax approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ATAN_DEFAULT_F16 symbol.

2.19.4.3 Return

The function returns the atan(f32In)/ π as a fixed point 32-bit number, normalized between [-0.25, 0.25).

2.19.4.4 Implementation details

The computational algorithm uses the symmetry of the arctangent function around the point (0, 0), which allows to for computing the function values just in the interval [0, 1) and to compute the function values in the interval [-1, 0) by the simple formula:

$$\tan^{-1}(-x) = -\tan^{-1}(x)$$

Equation GFLIB_Atan_F16_Eq1

The GFLIB_Atan_F16 function approximates the arctangent function using a piece-wise minimax polynomial approximation. The input range [0, 1) is divided into eight equally spaced sub intervals, each with a distinct set of minimax coefficients. Negative inputs are calculated according to the antisymmetry of the function.

The GFLIB_Atan_F16 function uses fixed point fractional arithmetic, so to cast the fractional value of the output angle [-0.25, 0.25) into the correct range [- π/4, π/4), the fixed point output angle can be multiplied by π for an angle in radians. Then, the fixed point fractional implementation of the minimax approximation polynomial, used for calculation of each sub sector, is defined as follows:

$$fOut = \frac{\tan^{-1}(fIn)}{\pi} = \left(a_0 + a_1 \cdot |fIn| + a_2 \cdot |fIn|^2\right) \cdot sign\left(fIn\right)$$

Equation GFLIB_Atan_F16_Eq2

The division of the [0, 1) interval into eight sub-intervals, with minimax polynomial coefficients calculated for each sub-interval, is noted in Table 148 .

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 148. Integer polynomial coefficients for each interval

Interval	a_0	a_1	a_2
<0, 1/8)	652	649	-3
<1/8, 2/8)	1934	630	-7
<2/8, 3/8)	3160	594	-11
<3/8, 4/8)	4302	547	-13
<4/8, 5/8)	5345	495	-13
<5/8, 6/8)	6283	443	-13
<6/8, 7/8)	7117	393	-12
<7/8, 1)	7856	347	-11

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.19.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16Input;
tFrac16 f16Angle;

void main(void)
{
    // input ratio = 0x7FFF
    f16Input  = (tFrac16)(0x7FFF);

    // output angle should be 0x1FFF = 0.249999 => pi/4
    f16Angle = GFLIB_Atan_F16(f16Input, GFLIB_ATAN_DEFAULT_F16);

    // output angle should be 0x1FFF = 0.249999 => pi/4
    f16Angle = GFLIB_Atan(f16Input, GFLIB_ATAN_DEFAULT_F16, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output angle should be 0x1FFF = 0.249999 => pi/4
    f16Angle = GFLIB_Atan(f16Input);
}
```

2.19.5 Function GFLIB_Atan_FLT

2.19.5.1 Declaration

```
tFloat GFLIB_Atan_FLT(tFloat fltIn, const GFLIB_ATAN_T_FLT *const
pParam);
```

2.19.5.2 Arguments

Table 149. GFLIB_Atan_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument is a single precision floating point number between (- 2^{128}, 2^{128}).

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 149. GFLIB_Atan_FLT arguments...*continued*

Type	Name	Direction	Description
const GFLIB_ATAN_T_FLT *const	pParam	**input**	Pointer to an array of rational polynomial coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ATAN_DEFAULT_FLT symbol.

2.19.5.3 Return

The function returns the atan of the input argument as a single precision floating point number that contains the angle in radians between (- π/2, π/2).

2.19.5.4 Implementation details

The computational algorithm uses the symmetry of the arctangent function around the point (0, 0), which allows to for computing the function values just in the interval [0, 1) and to compute the function values in the interval [-1, 0) by the simple formula:

$$\tan^{-1}(-x) = -\tan^{-1}(x)$$

Equation GFLIB_Atan_FLT_Eq1

The GFLIB_Atan_FLT function approximates the arctangent function using a rational polynomial approximation. The base interval of input values [0, tan(π/12)] is calculated using the equation

$$fOut = \frac{fIn\left[a_0 + fIn^2\left(a_1 + a_2 fIn^2\right)\right]}{a_3 + fIn^2\left[a_4 + fIn^2\left(a_5 + fIn^2\right)\right]} \cdot sign\left(fIn\right)$$

Equation GFLIB_Atan_FLT_Eq2

The rational polynomial coefficients are noted in Table 150 .

Table 150. Rational polynomial approximation coefficients

a_0	a_1	a_2	a_3	a_4	a_5
48.701070044049	89.532632877225	54.060435423162	24.766327050892	6.5878701670202	0.27010700440499

Input values in the interval (tan(π/12), 1] are transformed into the base interval [0, tan(π/12)] as follows:

$$fIn = \frac{fIn - \tan\left(\frac{\pi}{6}\right)}{1 + fIn \cdot \tan\left(\frac{\pi}{6}\right)} \quad if \quad |fIn| > \tan\left(\frac{\pi}{12}\right)$$

Equation GFLIB_Atan_FLT_Eq3

For input values greater than one, a reciprocal value is used in the rational polynomial approximation and the result is subtracted from π/2.

Results for the negative inputs are calculated according to the antisymmetry of the arctangent function.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The function may raise floating-point exceptions (invalid operation, division by zero, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.

2.19.5.5 Code Example

```
#include "gflib.h"

tFloat fltInput;
tFloat fltAngle;

void main(void)
{
    // input ratio = 1
    fltInput  = 1;

    // output angle should be 0.7853981634 => pi/4
    fltAngle = GFLIB_Atan_FLT(fltInput, GFLIB_ATAN_DEFAULT_FLT);

    // output angle should be 0.7853981634 => pi/4
    fltAngle = GFLIB_Atan(fltInput, GFLIB_ATAN_DEFAULT_FLT, FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output angle should be 0.7853981634 => pi/4
    fltAngle = GFLIB_Atan(fltInput);

}
```

2.20 Function GFLIB_AtanYX

This function calculates the angle between the positive x-axis and the direction of a vector given by the (x, y) coordinates.

2.20.1 Description

The function returns the angle between the positive x-axis of a plane and the direction of the vector given by the x and y coordinates provided as parameters. The first parameter is the ordinate (the y coordinate) and the second one is the abscissa (the x coordinate).

2.20.2 Re-entrancy

The function is re-entrant.

2.20.3 Function GFLIB_AtanYX_F32

2.20.3.1 Declaration

```
tFrac32 GFLIB_AtanYX_F32(tFrac32 f32InY, tFrac32 f32InX);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.20.3.2 Arguments

Table 151. GFLIB_AtanYX_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InY	**input**	The ordinate of the input vector (y coordinate).
tFrac32	f32InX	**input**	The abscissa of the input vector (x coordinate).

2.20.3.3 Return

The function returns the angle between the positive x-axis of a plane and the direction of the vector given by the x and y coordinates provided as parameters.

2.20.3.4 Implementation details

Both the input parameters are assumed to be in the fractional range of [-1, 1). The computed angle is limited by the fractional range of [-1, 1), which corresponds to the real range of [- π, π). The counter-clockwise direction is assumed to be positive and thus a positive angle will be computed if the provided ordinate (f32InY) is positive. Similarly, a negative angle will be computed for the negative ordinate.

The calculations are performed in a few steps.

In the first step, the angle is positioned within the correct half-quarter of the circumference of a circle by dividing the angle into two parts: the integral multiple of 45 deg (half-quarter) and the remaining offset within the 45 deg range. Simple geometric properties of the Cartesian coordinate system are used to calculate the coordinates of the vector with the calculated angle offset.

In the second step, the vector ordinate is divided by the vector abscissa (y/x) to obtain the tangent value of the angle offset. The angle offset is computed by applying the ordinary arctangent function.

The sum of the integral multiple of half-quarters and the angle offset within a single half-quarter form the angle to be computed. The function will return 0 if both input arguments are 0.

In comparison to the GFLIB_Atan_F32 function, the GFLIB_AtanYX_F32 function correctly places the calculated angle within the whole fractional range of [-1, 1), which corresponds to the real angle range of [- π, π).

Note: *The function calls the GFLIB_Atan_F32 function. The computed value is within the range of [-1, 1).*

2.20.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32InY;
tFrac32 f32InX;
tFrac32 f32Ang;

void main(void)
{
    // Angle 45 deg = PI/4 rad
    f32InY = FRAC32(0.5);
    f32InX = FRAC32(0.5);

    // output should be close to 0x200034EA
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    f32Ang = GFLIB_AtanYX_F32(f32InY, f32InX);

    // output should be close to 0x200034EA
    f32Ang = GFLIB_AtanYX(f32InY, f32InX, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be close to 0x200034EA
    f32Ang = GFLIB_AtanYX(f32InY, f32InX);
}
```

2.20.4 Function GFLIB_AtanYX_F16

2.20.4.1 Declaration

```
tFrac16 GFLIB_AtanYX_F16(tFrac16 f16InY, tFrac16 f16InX);
```

2.20.4.2 Arguments

Table 152. GFLIB_AtanYX_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InY	**input**	The ordinate of the input vector (y coordinate).
tFrac16	f16InX	**input**	The abscissa of the input vector (x coordinate).

2.20.4.3 Return

The function returns the angle between the positive x-axis of a plane and the direction of the vector given by the x and y coordinates provided as parameters.

2.20.4.4 Implementation details

Both the input parameters are assumed to be in the fractional range of [-1, 1). The computed angle is limited by the fractional range of [-1, 1), which corresponds to the real range of [- π, π). The counter-clockwise direction is assumed to be positive and thus a positive angle will be computed if the provided ordinate (f16InY) is positive. Similarly, a negative angle will be computed for the negative ordinate.

The calculations are performed in a few steps.

In the first step, the angle is positioned within the correct half-quarter of the circumference of a circle by dividing the angle into two parts: the integral multiple of 45 deg (half-quarter) and the remaining offset within the 45 deg range. Simple geometric properties of the Cartesian coordinate system are used to calculate the coordinates of the vector with the calculated angle offset.

In the second step, the vector ordinate is divided by the vector abscissa (y/x) to obtain the tangent value of the angle offset. The angle offset is computed by applying the ordinary arctangent function.

The sum of the integral multiple of half-quarters and the angle offset within a single half-quarter form the angle to be computed. The function will return 0 if both input arguments are 0.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

In comparison to the GFLIB_Atan_F16 function, the GFLIB_AtanYX_F16 function correctly places the calculated angle within the whole fractional range of [-1, 1), which corresponds to the real angle range of [- π, π).

Note: *The function calls the GFLIB_Atan_F16 function. The computed value is within the range of [-1, 1).*

2.20.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16InY;
tFrac16 f16InX;
tFrac16 f16Ang;

void main(void)
{
    // Angle 45 deg = PI/4 rad
    f16InY = FRAC16(0.5);
    f16InX = FRAC16(0.5);

    // output should be close to 0x2000
    f16Ang = GFLIB_AtanYX_F16(f16InY, f16InX);

    // output should be close to 0x2000
    f16Ang = GFLIB_AtanYX(f16InY, f16InX, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be close to 0x2000
    f16Ang = GFLIB_AtanYX(f16InY, f16InX);
}
```

2.20.5 Function GFLIB_AtanYX_FLT

2.20.5.1 Declaration

```
tFloat GFLIB_AtanYX_FLT(tFloat fltInY, tFloat fltInX);
```

2.20.5.2 Arguments

Table 153. GFLIB_AtanYX_FLT arguments

Type	Name	Direction	Description
tFloat	fltInY	**input**	The ordinate of the input vector (y coordinate).
tFloat	fltInX	**input**	The abscissa of the input vector (x coordinate).

2.20.5.3 Return

The function returns the angle between the positive x-axis of a plane and the direction of the vector given by the x and y coordinates provided as parameters.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.20.5.4 Implementation details

Both the input parameters are assumed to be in the single precision floating point range of (- 2^{128}, 2^{128}). The computed angle is in the range of [- π, π]. The counter-clockwise direction is assumed to be positive, and thus a positive angle will be computed if the provided ordinate (fltInY) is positive. Similarly, a negative angle will be computed for the negative ordinate. The calculations are performed in a few steps.

In the first step, the angle is positioned within the correct half-quarter of the circumference of a circle, and if necessary the final angle addition is prepared or the sign of the input value is corrected.

In the second step, the vector ordinate is divided by the vector abscissa (y/x) to obtain the tangent value of the angle offset. The angle offset is computed by applying the ordinary arctangent function.

The sum of the angle addition and the angle offset within a single quarter form the angle to be computed. The function will return 0 if both input arguments are 0.

Note: *The function may raise floating-point exceptions (invalid operation, division by zero, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

The function calls the GFLIB_Atan_FLT function. The computed value is within the range of [- π, π].

2.20.5.5 Code Example

```
#include "gflib.h"

tFloat fltInY;
tFloat fltInX;
tFloat fltAng;

void main(void)
{
    // Angle 45 deg = PI/4 rad
    fltInY = (tFloat)(0.5);
    fltInX = (tFloat)(0.5);

    // Output angle should be 45 deg = PI/4 rad = 0.7853981634
    fltAng = GFLIB_AtanYX_FLT(fltInY, fltInX);

    // Output angle should be 45 deg = PI/4 rad = 0.7853981634
    fltAng = GFLIB_AtanYX(fltInY, fltInX, FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // Output angle should be 45 deg = PI/4 rad = 0.7853981634
    fltAng = GFLIB_AtanYX(fltInY, fltInX);
}
```

2.21 Function GFLIB_AtanYXShifted

This function calculates the angle of two sine waves shifted in phase to each other.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.21.1 Description

The function calculates the angle of two sinusoidal signals, one shifted in phase to the other. The phase shift between sinusoidal signals does not have to be π/2 and can be any value.

It is assumed that the arguments of the function are as follows:

$$y = \sin(\theta)$$
$$x = \sin(\theta + \Delta\theta)$$

Equation GFLIB_AtanYXShifted_Eq1

where:

- x, y are respectively, the InX and InY arguments
- θ is the angle to be computed by the function
- Δθ is the phase difference between the x, y signals

At the end of computations, an angle offset θ_{Offset} is added to the computed angle θ. The angle offset is an additional parameter, which can be used to set the zero of the θ axis. If θ_{Offset} is zero, then the angle computed by the function will be exactly θ.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.21.2 Re-entrancy

The function is re-entrant.

2.21.3 Function GFLIB_AtanYXShifted_F32

2.21.3.1 Declaration

```
tFrac32 GFLIB_AtanYXShifted_F32(tFrac32 f32InY, tFrac32 f32InX,
const GFLIB_ATANYXSHIFTED_T_F32 *pParam);
```

2.21.3.2 Arguments

Table 154. GFLIB_AtanYXShifted_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InY	**input**	The value of the first signal, assumed to be sin (θ).
tFrac32	f32InX	**input**	The value of the second signal, assumed to be sin(θ + Δθ).
const GFLIB_ATANYXSHIFTED_T_F32 *	pParam	**input**	The parameters for the function.

2.21.3.3 Return

The function returns the angle of two sine waves shifted in phase to each other.

2.21.3.4 Implementation details

The GFLIB_AtanYXShifted_F32 function does not directly use the angle offset θ_{Offset} and the phase difference θ. The function's parameters, contained in the function parameters

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

structure GFLIB_ATANYXSHIFTED_T_F32, need to be computed by means of the provided Matlab function (see below).

If Δθ= π/2 or Δθ=- π/2, then the function is similar to the GFLIB_AtanYX_F32 function, however, the GFLIB_AtanYX_F32 function in this case is more effective with regard to execution time and accuracy.

In order to use the function, the following necessary steps need to be completed:

- define Δθ and θ_{Offset}, the Δθ shall be known from the input sinusoidal signals, the θ_{Offset} needs to be set arbitrarily
- compute values for the function parameters structure by means of the provided Matlab function
- convert the computed values into integer format and insert them into the C code (see also the C code example)

The function uses the following algorithm for computing the angle:

$$b = \frac{S}{2 \cdot \cos\left(\frac{\Delta\theta}{2}\right)}(y + x)$$

$$a = \frac{S}{2 \cdot \sin\left(\frac{\Delta\theta}{2}\right)}(x - y)$$

$$\theta = \tan^{-1}\left(\frac{y}{x}\right) - \left(\frac{\Delta\theta}{2} - \theta_{offset}\right)$$

Equation GFLIB_AtanYXShifted_F32_Eq1

where:

- x, y are respectively, the f32InX, and f32InY
- θ is the angle to be computed by the function, see the previous equation
- Δθ is the phase difference between the x, y signals, see the previous equation
- S is a scaling coefficient, S is almost 1, (S < 1), see also the explanation below
- a, b intermediate variables
- θ_{Offset} is the additional phase shift, the computed angle will be θ + θ_{Offset}

The scale coefficient S is used to prevent overflow and to assure symmetry around 0 for the entire fractional range. S shall be less than 1.0, but as large as possible. The algorithm implemented in this function uses the value of 1 - 2^{-15}.

The algorithm can be easily justified by proving the trigonometric identity:

$$\tan(\theta + \Delta\theta) = \frac{(y+x) \cdot \cos\left(\frac{\Delta\theta}{2}\right)}{(x-y) \cdot \sin\left(\frac{\Delta\theta}{2}\right)}$$

Equation GFLIB_AtanYXShifted_F32_Eq2

For the purposes of fractional arithmetic, the algorithm is implemented such that additional values are used as shown in the equation below:

$$\frac{S}{2 \cdot \cos\left(\frac{\Delta\theta}{2}\right)} = C_y = K_y \cdot 2^{N_y}$$

$$\frac{S}{2 \cdot \sin\left(\frac{\Delta\theta}{2}\right)} = C_x = K_x \cdot 2^{N_x}$$

$$\theta_{adj} = \frac{\Delta\theta}{2} - \theta_{offset}$$

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Equation GFLIB_AtanYXShifted_F32_Eq3

where:

- C_y, C_x are the algorithm coefficients for y and x signals
- K_y is multiplication coefficient of the y signal, represented by the parameters structure member pParam->f32Ky
- K_x is multiplication coefficient of the x signal, represented by the parameters structure member pParam->f32Kx
- N_y is scaling coefficient of the y signal, represented the by parameters structure member pParam->s32Ny
- N_x is scaling coefficient of the x signal, represented by the parameters structure member pParam->s32Nx
- θ_{adj} is an adjusting angle, represented by the parameters structure member pParam->f32ThetaAdj

The multiplication and scaling coefficients, and the adjusting angle, shall be defined in a parameters structure provided as the function input parameter.

The function initialization parameters can be calculated as shown in the following Matlab code:

```
function [KY, KX, NY, NX, THETAADJ] = atanyxshiftedpar(dthdeg, thoffsetdeg)
// ATANYXSHIFTEDPAR calculation of parameters for atanyxshifted() function
//
// [KY, KX, NY, NX, THETAADJ] = atanyxshiftedpar(dthdeg, thoffsetdeg)
//
// dthdeg = phase shift (delta theta) between sine waves in degrees
// thoffsetdeg = angle offset (theta offset) in degrees
// NY - scaling coefficient of y signal
// NX - scaling coefficient of x signal
// KY - multiplication coefficient of y signal
// KX - multiplication coefficient of x signal
// THETAADJ - adjusting angle in radians, scaled from [-pi, pi) to [-1, 1)

    if (dthdeg < -180) || (dthdeg >= 180)
        error('atanyxshiftedpar: dthdeg out of range');
    end
    if (thoffsetdeg < -180) || (thoffsetdeg >= 180)
        error('atanyxshiftedpar: thoffsetdeg out of range');
    end

    dth2 = ((dthdeg/2)/180*pi);
    thoffset = (thoffsetdeg/180*pi);
    CY = (1 - 2^-15)/(2*cos(dth2));
    CX = (1 - 2^-15)/(2*sin(dth2));
    if(abs(CY) >= 1) NY = ceil(log2(abs(CY)));
    else NY = 0;
    end
    if(abs(CX) >= 1) NX = ceil(log2(abs(CX)));
    else NX = 0;
    end
    KY = CY/2^NY;
    KX = CX/2^NX;
    THETAADJ = dthdeg/2 - thoffsetdeg;

    if THETAADJ >= 180
        THETAADJ = THETAADJ - 360;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    elseif THETAADJ < -180
        THETAADJ = THETAADJ + 360;
    end

    THETAADJ = THETAADJ/180;

return;
```

While applying the function, some general guidelines should be considered as stated below.

At some values of the phase shift, and particularly at phase shift approaching -180, 0 or 180 degrees, the algorithm may become numerically unstable, causing any error, contributed by input signal imperfections or through finite precision arithmetic, to be magnified significantly. Therefore, some care should be taken to avoid error where possible. The detailed error analysis of the algorithm is beyond the scope of this documentation, however, general guidelines are provided.

There are several sources of error in the function:

- error of the supplied signal values due to the finite resolution of the AD conversion
- error contributed by higher order harmonics appearing in the input signals
- computational error of the multiplication due to the finite length of registers
- error of the phase shift Δθ representation in the finite precision arithmetic and in the values
- error due to differences in signal amplitudes

It should be noted that the function requires both signals to have the same amplitude. To minimize the output error, the amplitude of both signals should be as close to 1.0 as much as possible.

The function has been tested to be reliable at a phase shift in the range of [-165, -15] and [15, 165] degrees for perfectly sinusoidal input signals. Beyond this range, the function operates correctly, however, the output error can be beyond the guaranteed value. In a real application, an error, contributed by an AD conversion and by higher order harmonics of the input signals, should be also taken into account.

Note: *The function calls the GFLIB_AtanYX_F32 function. The function may become numerically unstable for a phase shift approaching -180, 0 or 180 degrees. The function accuracy is guaranteed for a phase shift in the range of [-165, -15] and [15, 165] degrees at perfect input signals.*

CAUTION: *Due to the cyclic character of the GFLIB_AtanYX_F16, in case the difference between the adjusting angle θ_{adj} and the input vector angle is approaching to 1- 2^{-15} or -1, the GFLIB_AtanYX_F16 function operates correctly, however the output error might exceed the guaranteed limits.*

2.21.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32InY;
tFrac32 f32InX;
tFrac32 f32Ang;
GFLIB_ATANYXSHIFTED_T_F32 Param;

void main(void)
{
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // dtheta = 69.33deg, thetaoffset = 10deg
    // CY = (1 - 2^-15)/(2*cos((69.33/2)/180*pi))= 0.60789036201452440
    // CX = (1 - 2^-15)/(2*sin((69.33/2)/180*pi))= 0.87905201358520957
    // NY = 0 (abs(CY) < 1)
    // NX = 0 (abs(CX) < 1)
    // KY = 0.60789/2^0 = 0.60789036201452440
    // KX = 0.87905/2^0 = 0.87905201358520957
    // THETAADJ = 10/180 = 0.05555555555

    Param.f32Ky = FRAC32(0.60789036201452440);
    Param.f32Kx = FRAC32(0.87905201358520957);
    Param.s32Ny = 0;
    Param.s32Nx = 0;
    Param.f32ThetaAdj = FRAC32(0.05555555555);

    // theta = 15 deg
    // Y = sin(theta) = 0.2588190
    // X = sin(theta + dtheta) = 0.9951074
    f32InY = FRAC32(0.2588190);
    f32InX = FRAC32(0.9951074);

    // f32Ang output should be close to 0x1C34824A
    f32Ang = GFLIB_AtanYXShifted_F32(f32InY, f32InX, &Param);

    // f32Ang output should be close to 0x1C34824A
    f32Ang = GFLIB_AtanYXShifted(f32InY, f32InX, &Param, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // f32Ang output should be close to 0x1C34824A
    f32Ang = GFLIB_AtanYXShifted(f32InY, f32InX, &Param);
}
```

2.21.4 Function GFLIB_AtanYXShifted_F16

2.21.4.1 Declaration

```
tFrac16 GFLIB_AtanYXShifted_F16(tFrac16 f16InY, tFrac16 f16InX,
const GFLIB_ATANYXSHIFTED_T_F16 *pParam);
```

2.21.4.2 Arguments

Table 155. GFLIB_AtanYXShifted_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InY	**input**	The value of the first signal, assumed to be sin (θ).
tFrac16	f16InX	**input**	The value of the second signal, assumed to be sin(θ + Δθ).
const GFLIB_ATANYXSHIFTED_T_F16 *	pParam	**input**	The parameters for the function.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.21.4.3 Return

The function returns the angle of two sine waves shifted in phase to each other.

2.21.4.4 Implementation details

The GFLIB_AtanYXShifted_F16 function does not directly use the angle offset θ_{Offset} and the phase difference θ. The function's parameters, contained in the function parameters structure GFLIB_ATANYXSHIFTED_T_F16, need to be computed by means of the provided Matlab function (see below).

If Δθ= π/2 or Δθ=- π/2, then the function is similar to the GFLIB_AtanYX_F16 function, however, the GFLIB_AtanYX_F16 function in this case is more effective with regard to execution time and accuracy.

In order to use the function, the following necessary steps need to be completed:

- define Δθ and θ_{Offset}, the Δθ shall be known from the input sinusoidal signals, the θ_{Offset} needs to be set arbitrarily
- compute values for the function parameters structure by means of the provided Matlab function
- convert the computed values into integer format and insert them into the C code (see also the C code example)

The function uses the following algorithm for computing the angle:

$$b = \frac{S}{2 \cdot \cos\left(\frac{\Delta\theta}{2}\right)}(y + x)$$

$$a = \frac{S}{2 \cdot \sin\left(\frac{\Delta\theta}{2}\right)}(x - y)$$

$$\theta = \tan^{-1}\left(\frac{y}{x}\right) - \left(\frac{\Delta\theta}{2} - \theta_{offset}\right)$$

Equation GFLIB_AtanYXShifted_F16_Eq1

where:

- x, y are respectively, the f16InX, and f16InY
- θ is the angle to be computed by the function, see the previous equation
- Δθ is the phase difference between the x, y signals, see the previous equation
- S is a scaling coefficient, S is almost 1, (S < 1), see also the explanation below
- a, b intermediate variables
- θ_{Offset} is the additional phase shift, the computed angle will be θ + θ_{Offset}

The scale coefficient S is used to prevent overflow and to assure symmetry around 0 for the entire fractional range. S shall be less than 1.0, but as large as possible. The algorithm implemented in this function uses the value of $1 - 2^{-15}$.

The algorithm can be easily justified by proving the trigonometric identity:

$$\tan(\theta + \Delta\theta) = \frac{(y+x) \cdot \cos\left(\frac{\Delta\theta}{2}\right)}{(x-y) \cdot \sin\left(\frac{\Delta\theta}{2}\right)}$$

Equation GFLIB_AtanYXShifted_F16_Eq2

For the purposes of fractional arithmetic, the algorithm is implemented such that additional values are used as shown in the equation below:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$\frac{S}{2\cdot\cos\left(\frac{\Delta\theta}{2}\right)} = C_y = K_y \cdot 2^{N_y}$$

$$\frac{S}{2\cdot\sin\left(\frac{\Delta\theta}{2}\right)} = C_x = K_x \cdot 2^{N_x}$$

$$\theta_{adj} = \frac{\Delta\theta}{2} - \theta_{offset}$$

Equation GFLIB_AtanYXShifted_F16_Eq3

where:

- C_y, C_x are the algorithm coefficients for y and x signals
- K_y is multiplication coefficient of the y signal, represented by the parameters structure member pParam->f16Ky
- K_x is multiplication coefficient of the x signal, represented by the parameters structure member pParam->f16Kx
- N_y is scaling coefficient of the y signal, represented the by parameters structure member pParam->s16Ny
- N_x is scaling coefficient of the x signal, represented by the parameters structure member pParam->s16Nx
- θ_{adj} is an adjusting angle, represented by the parameters structure member pParam->f16ThetaAdj

The multiplication and scaling coefficients, and the adjusting angle, shall be defined in a parameters structure provided as the function input parameter.

The function initialization parameters can be calculated as shown in the following Matlab code:

```
function [KY, KX, NY, NX, THETAADJ] = atanyxshiftedpar(dthdeg, thoffsetdeg)
// ATANYXSHIFTEDPAR calculation of parameters for atanyxshifted() function
//
// [KY, KX, NY, NX, THETAADJ] = atanyxshiftedpar(dthdeg, thoffsetdeg)
//
// dthdeg = phase shift (delta theta) between sine waves in degrees
// thoffsetdeg = angle offset (theta offset) in degrees
// NY - scaling coefficient of y signal
// NX - scaling coefficient of x signal
// KY - multiplication coefficient of y signal
// KX - multiplication coefficient of x signal
// THETAADJ - adjusting angle in radians, scaled from [-pi, pi) to [-1, 1)

    if (dthdeg < -180) || (dthdeg >= 180)
        error('atanyxshiftedpar: dthdeg out of range');
    end
    if (thoffsetdeg < -180) || (thoffsetdeg >= 180)
        error('atanyxshiftedpar: thoffsetdeg out of range');
    end

    dth2 = ((dthdeg/2)/180*pi);
    thoffset = (thoffsetdeg/180*pi);
    CY = (1 - 2^-15)/(2*cos(dth2));
    CX = (1 - 2^-15)/(2*sin(dth2));
    if(abs(CY) >= 1) NY = ceil(log2(abs(CY)));
    else NY = 0;
    end
    if(abs(CX) >= 1) NX = ceil(log2(abs(CX)));
    else NX = 0;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    end
    KY = CY/2^NY;
    KX = CX/2^NX;
    THETAADJ = dthdeg/2 - thoffsetdeg;

    if THETAADJ >= 180
        THETAADJ = THETAADJ - 360;
    elseif THETAADJ < -180
        THETAADJ = THETAADJ + 360;
    end

    THETAADJ = THETAADJ/180;

return;
```

While applying the function, some general guidelines should be considered as stated below.

At some values of the phase shift, and particularly at phase shift approaching -180, 0 or 180 degrees, the algorithm may become numerically unstable, causing any error, contributed by input signal imperfections or through finite precision arithmetic, to be magnified significantly. Therefore, some care should be taken to avoid error where possible. The detailed error analysis of the algorithm is beyond the scope of this documentation, however, general guidelines are provided.

There are several sources of error in the function:

- error of the supplied signal values due to the finite resolution of the AD conversion
- error contributed by higher order harmonics appearing in the input signals
- computational error of the multiplication due to the finite length of registers
- error of the phase shift Δθ representation in the finite precision arithmetic and in the values
- error due to differences in signal amplitudes

It should be noted that the function requires both signals to have the same amplitude. To minimize the output error, the amplitude of both signals should be as close to 1.0 as much as possible.

The function has been tested to be reliable at a phase shift in the range of [-165, -15] and [15, 165] degrees for perfectly sinusoidal input signals. Beyond this range, the function operates correctly, however, the output error can be beyond the guaranteed value. In a real application, an error, contributed by an AD conversion and by higher order harmonics of the input signals, should be also taken into account.

Note: *The function calls the GFLIB_AtanYX_F16 function. The function may become numerically unstable for a phase shift approaching -180, 0 or 180 degrees. The function accuracy is guaranteed for a phase shift in the range of [-175, -5] and [5, 175] degrees at perfect input signals. To eliminate the calculation error the function uses the 32-bit internal accumulators.*

CAUTION: *Due to the cyclic character of the GFLIB_AtanYX_F16, in case the difference between the adjusting angle θ_{adj} and the input vector angle is approaching to 1- 2^{-15} or -1, the GFLIB_AtanYX_F16 function operates correctly, however the output error might exceed the guaranteed limits.*

2.21.4.5 Code Example

```
#include "gflib.h"
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
tFrac16 f16InY;
tFrac16 f16InX;
tFrac16 f16Ang;
GFLIB_ATANYXSHIFTED_T_F16 Param;

void main(void)
{
    // dtheta = 69.33deg, thetaoffset = 10deg
    // CY = (1 - 2^-15)/(2*cos((69.33/2)/180*pi))= 0.60789036201452440
    // CX = (1 - 2^-15)/(2*sin((69.33/2)/180*pi))= 0.87905201358520957
    // NY = 0 (abs(CY) < 1)
    // NX = 0 (abs(CX) < 1)
    // KY = 0.60789/2^0 = 0.60789036201452440
    // KX = 0.87905/2^0 = 0.87905201358520957
    // THETAADJ = 10/180 = 0.05555555555

    Param.f16Ky = FRAC16(0.60789036201452440);
    Param.f16Kx = FRAC16(0.87905201358520957);
    Param.s16Ny = 0;
    Param.s16Nx = 0;
    Param.f16ThetaAdj = FRAC16(0.05555555555);

    // theta = 15 deg
    // Y = sin(theta) = 0.2588190
    // X = sin(theta + dtheta) = 0.9951074
    f16InY = FRAC16(0.2588190);
    f16InX = FRAC16(0.9951074);

    // f16Ang output should be close to 0x1C34
    f16Ang = GFLIB_AtanYXShifted_F16(f16InY, f16InX, &Param);

    // f16Ang output should be close to 0x1C34
    f16Ang = GFLIB_AtanYXShifted(f16InY, f16InX, &Param, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // f16Ang output should be close to 0x1C34
    f16Ang = GFLIB_AtanYXShifted(f16InY, f16InX, &Param);
}
```

2.21.5 Function GFLIB_AtanYXShifted_FLT

2.21.5.1 Declaration

```
tFloat GFLIB_AtanYXShifted_FLT(tFloat fltInY, tFloat fltInX,
const GFLIB_ATANYXSHIFTED_T_FLT *pParam);
```

2.21.5.2 Arguments

Table 156. GFLIB_AtanYXShifted_FLT arguments

Type	Name	Direction	Description
tFloat	fltInY	**input**	The value of the first signal, assumed to be sin (θ).

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 156. GFLIB_AtanYXShifted_FLT arguments...*continued*

Type	Name	Direction	Description
tFloat	fltInX	**input**	The value of the second signal, assumed to be sin(θ + Δθ).
const GFLIB_ATANYXSHIFTED_T_FLT *	pParam	**input**	The parameters for the function.

2.21.5.3 Return

The function returns the angle of two sine waves shifted in phase to each other.

2.21.5.4 Implementation details

At the end of computations, an angle offset θ_{Offset} is added to the computed angle θ. The angle offset is an additional parameter which can be used to set the zero of the θ axis. If θ_{Offset} is zero, then the angle computed by the function will be exactly θ.

The GFLIB_AtanYXShifted_FLT function does not directly use the angle offset θ_{Offset} and the phase difference θ. The function's parameters, contained in the function parameters structure GFLIB_ATANYXSHIFTED_T_FLT, need to be computed by means of the provided Matlab function (see below).

If Δθ= π/2 or Δθ=- π/2, then the function is similar to the GFLIB_AtanYX_FLT function, however, the GFLIB_AtanYX_FLT function in this case is more effective with regards to execution time and accuracy.

In order to use the function, the following necessary steps need to be completed:

- define Δθ and θ_{Offset}, the Δθ shall be known from the input sinusoidal signals, the θ_{Offset} needs to be set arbitrarily
- compute values for the function parameters structure by means of the provided Matlab function

The function uses the following algorithm for computing the angle:

$$b = \frac{S}{2 \cdot \cos\left(\frac{\Delta\theta}{2}\right)}(y + x)$$

$$a = \frac{S}{2 \cdot \sin\left(\frac{\Delta\theta}{2}\right)}(x - y)$$

$$\theta = \tan^{-1}\left(\frac{y}{x}\right) - \left(\frac{\Delta\theta}{2} - \theta_{offset}\right)$$

Equation GFLIB_AtanYXShifted_FLT_Eq1

where:

- x, y are respectively, the fltInX, and fltInY
- θ is the angle to be computed by the function, see the previous equation
- Δθ is the phase difference between the x, y signals, see the previous equation
- a, b intermediate variables
- θ_{Offset} is the additional phase shift in radians, the computed angle will be θ + θ_{Offset}

The algorithm can be easily justified by proving the trigonometric identity:

$$\tan(\theta + \Delta\theta) = \frac{(y+x) \cdot \cos\left(\frac{\Delta\theta}{2}\right)}{(x-y) \cdot \sin\left(\frac{\Delta\theta}{2}\right)}$$

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Equation GFLIB_AtanYXShifted_FLT_Eq2

For the purpose of the caluclation, the multiplication coeficients K_y and K_x are representing the fractions in the previouse equation and are defined as follows:

$$\frac{S}{2 \cdot \cos\left(\frac{\Delta\theta}{2}\right)} = C_y = K_y$$

$$\frac{S}{2 \cdot \sin\left(\frac{\Delta\theta}{2}\right)} = C_x = K_x$$

$$\theta_{adj} = \frac{\Delta\theta}{2} - \theta_{offset}$$

Equation GFLIB_AtanYXShifted_FLT_Eq3

where:

- K_y is multiplication coefficient of the y signal, represented by the parameters structure member pParam->fltKy
- K_x is multiplication coefficient of the x signal, represented by the parameters structure member pParam->fltKx
- θ_{adj} is an adjusting angle, represented by the parameters structure member pParam->fltThetaAdj

The function initialization parameters can be calculated as shown in the following Matlab code:

```
function [KY, KX, THETAADJ] = atanyxshiftedpar(dthdeg, thoffsetdeg)
// ATANYXSHIFTEDPAR calculation of parameters for atanyxshifted() function
//
// [KY, KX, NY, NX, THETAADJ] = atanyxshiftedpar(dthdeg, thoffsetdeg)
//
// dthdeg = phase shift (delta theta) between sine waves in degrees
// thoffsetdeg = angle offset (theta offset) in degrees
// KY - multiplication coefficient of y signal
// KX - multiplication coefficient of x signal
// THETAADJ - adjusting angle in radians, scaled from [-pi, pi) to [-1, 1)

    if (dthdeg < -180) || (dthdeg >= 180)
        error('atanyxshiftedpar: dthdeg out of range');
    end
    if (thoffsetdeg < -180) || (thoffsetdeg >= 180)
        error('atanyxshiftedpar: thoffsetdeg out of range');
    end

    dth2 = ((dthdeg/2)/180*pi);
    thoffset = (thoffsetdeg/180*pi);
    KY = 1/(2*cos(dth2));
    KX = 1/(2*sin(dth2));
    THETAADJ = dthdeg/2 - thoffsetdeg;

    if THETAADJ >= 180
        THETAADJ = THETAADJ - 360;
    elseif THETAADJ < -180
        THETAADJ = THETAADJ + 360;
    end

    THETAADJ = THETAADJ/180;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
return;
```

While applying the function, some general guidelines should be considered as stated below.

At some values of the phase shift, and particularly at a phase shift approaching -180, 0, or 180 degrees, the algorithm may become numerically unstable, causing any error, contributed by input signal imperfections or through finite precision arithmetic, to be magnified significantly.

There are several sources of error in the function:

- error of the supplied signal values due to the finite resolution of the AD conversion
- error contributed by higher order harmonics appearing in the input signals
- computational error of arithmetic operations due to the finite length of registers
- error of the phase shift Δθ representation in the finite precision arithmetic and in the values
- error due to differences in signal amplitudes

To minimize the output error, both signals should have the same amplitude.

Note: *The function may raise floating-point exceptions (invalid operation, division by zero, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.21.5.5 Code Example

```
#include "gflib.h"

tFloat fltInY;
tFloat fltInX;
tFloat fltAng;
GFLIB_ATANYXSHIFTED_T_FLT Param;

void main(void)
{
    // dtheta = 69.33deg, thetaoffset = 10deg
    // KY = 1/(2*cos((69.33/2)/180*pi))= 0.6079089139
    // KX = 1/(2*sin((69.33/2)/180*pi))= 0.8790788409

    // THETAADJ = 10*pi/180 = 0.1745329252

    Param.fltKy = (tFloat)(0.6079089139);
    Param.fltKx = (tFloat)(0.8790788409);
    Param.fltThetaAdj = (tFloat)(0.1745329252);

    // theta = 15 deg
    // Y = sin(theta) = 0.2588190
    // X = sin(theta + dtheta) = 0.9951074
    fltInY = (tFloat)(0.2588190);
    fltInX = (tFloat)(0.9951074);

    // Output angle should be 0.69228911 rad
    fltAng = GFLIB_AtanYXShifted_FLT(fltInY, fltInX, &Param);

    // Output angle should be 0.69228911 rad
    fltAng = GFLIB_AtanYXShifted(fltInY, fltInX, &Param, FLT);

    // ##########################################################
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Available only if single precision floating point
    // implementation selected as default
    // ##########################################################

    // Output angle should be 0.69228911 rad
    fltAng = GFLIB_AtanYXShifted(fltInY, fltInX, &Param);
}
```

2.22 Function GFLIB_ControllerPIDpAW

The function calculates the parallel form of the Proportional-Integral-Derivative (PID) controller with implemented integral anti-windup functionality.

2.22.1 Description

A PID controller attempts to correct the error between a measured process variable and a desired set-point by calculating and then outputting a corrective action that can adjust the process accordingly. The GFLIB_ControllerPIDpAW function calculates the discrete-time approximation of the Proportional-Integral-Derivative (PID) algorithm according to the following equation:

$$u(t) = K_P \cdot e(t) + K_I \int_0^t e(t)\,dt + K_D \frac{de(t)}{dt}$$

Equation GFLIB_ControllerPIDpAW_Eq1

where

- e(t) - input error in the continuous time domain
- u(t) - controller output in the continuous time domain
- K_P - proportional gain
- K_I - integral gain
- K_D - derivative gain

The PID algorithm is implemented in the parallel (non-interacting) form, allowing the user to define the P, I and D parameters independently without interaction. The controller output is limited and the limit values (UpperLimit and LowerLimit) are defined by the user. The PID controller algorithm also returns a limitation flag. This flag (u16LimitFlag) is a member of the structure of the PID controller parameters. If the PID controller output reaches the upper or lower limit then u16LimitFlag = 1, otherwise u16LimitFlag = 0 (integer values). An anti-windup strategy is implemented by limiting the integral portion. The integral state is limited by the controller limits, in the same way as the controller output. In the GFLIB_ControllerPIDpAW function is implemented filtration of derivative term. Structure of continuous-time equivalent of the implemented PID controller is shown in Figure 88 .

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 88. PID controller structure

This system can be described using the Laplace transform as follows:

$$H(s) = \frac{U(s)}{E(s)} = K_P + K_I \cdot \frac{1}{s} + K_D \cdot \frac{N}{1 + N \cdot \frac{1}{s}}$$

Equation GFLIB_ControllerPIDpAW_Eq2

The proportional part of equation GFLIB_ControllerPIDpAW_Eq2 is transformed into the discrete-time domain simply as:

$$u_P(k) = K_P \cdot e(k)$$

Equation GFLIB_ControllerPIDpAW_Eq3

Transforming the integral part of equation GFLIB_ControllerPIDpAW_Eq2 into a discrete-time domain using the bilinear method (also known as trapezoidal approximation) leads to the following equation:

$$u_I(k) = \frac{K_I T_S}{2} \cdot [e(k) + e(k-1)] + u_I(k-1)$$

Equation GFLIB_ControllerPIDpAW_Eq4

where T_s [sec] is the sampling period.

The derivative part of equation GFLIB_ControllerPIDpAW_Eq2 is transformed into discrete-time domain using impulse invariance method as:

$$u_D(k) = \frac{K_D}{T_S} \cdot (1 - e^{-T_S N}) \cdot [e(k) - e(k-1)] + e^{-T_S N} \cdot u_D(k-1)$$

Equation GFLIB_ControllerPIDpAW_Eq5

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.22.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.22.3 Function GFLIB_ControllerPIDpAW_F32

2.22.3.1 Declaration

```
tFrac32 GFLIB_ControllerPIDpAW_F32(tFrac32 f32InErr,
GFLIB_CONTROLLER_PID_P_AW_T_F32 *const pParam);
```

2.22.3.2 Arguments

Table 157. GFLIB_ControllerPIDpAW_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InErr	**input**	Input error signal to the controller is a 32-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PID_P_AW_T_F32 *const	pParam	**input, output**	Pointer to the controller parameters structure.

2.22.3.3 Return

The function returns a 32-bit value in format Q1.31, representing the signal to be applied to the controlled system so that the input error is forced to zero.

2.22.3.4 Implementation details

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. The input and output of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

The fractional representation of both input and output signals, normalized between [-1, 1), is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation GFLIB_ControllerPIDpAW_F32_Eq1

Applying such scaling (normalization) on the proportional term of equation GFLIB_ControllerPIDpAW_eq3 results in:

$$u_{Pf}(k) = K_{P_{SC}} \cdot e_f(k) \quad where \quad K_{P_{SC}} = K_P \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIDpAW_F32_Eq2

where K_{P_sc} is the proportional gain parameter considering input/output scaling.

Similarly, scaling the integral term of equation GFLIB_ControllerPIDpAW_eq4 results in:

$$u_{If}(k) = K_{I_SC} \cdot [e_f(k) + e_f(k-1)] + u_{If}(k-1) \quad where \quad K_{I_SC} = \frac{K_I T_S}{2} \cdot \frac{E^{MAX}}{U^{MAX}}$$

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Equation GFLIB_ControllerPIDpAW_F32_Eq3

where K_{I_sc} is the integral gain parameter considering input/output scaling.

And finally, scaling the derivative term, equation GFLIB_ControllerPIDpAW_eq5 results in:

$$u_{If}(k) = K_{D_SC} \cdot [e_f(k) - e_f(k-1)] + e^{-\frac{T_s N}{T_D}} \cdot u_{Df}(k-1) \quad where \quad K_{D_SC} = \frac{K_D}{T_s} \cdot \left(1 - e^{-\frac{T_s N}{T_D}}\right) \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIDpAW_F32_Eq4

where K_{D_sc} is the derivative gain parameter considering input/output scaling.

In case the gain parameters K_{P_sc}, K_{I_sc} or K_{D_sc} exceed the fractional range [-1, 1) or they are extremely small (which could lead to loss of precision), their value must be bit-shifted by *s16PropGainShift*, *s16IntegGainShift*, and *s16DerivGainShift*, respectively. The shift amount can be calculated as

$$s16PropGainShift = \lceil \log_2(|K_{P_SC}|) \rceil$$
$$s16IntegGainShift = \lceil \log_2(|K_{I_SC}|) \rceil$$
$$s16DerivGainShift = \lceil \log_2(|K_{D_SC}|) \rceil$$

Equation GFLIB_ControllerPIDpAW_F32_Eq5

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 31st power of two (corresponds to the Q1.31 encoding):

$$f32PropGain = \left(K_{P_SC} \cdot 2^{-s16PropGainShift}\right) \cdot 2^{31}$$
$$f32IntegGain = \left(K_{I_SC} \cdot 2^{-s16IntegGainShift}\right) \cdot 2^{31}$$
$$f32DerivGain = \left(K_{D_SC} \cdot 2^{-s16DerivGainShift}\right) \cdot 2^{31}$$

Equation GFLIB_ControllerPIDpAW_F32_Eq6

where

- f32PropGain - is the scaled value of proportional gain
- s16PropGainShift - is the scaling shift for proportional gain [-31, 31]
- f32IntegGain - is the scaled value of integral gain
- s16IntegGainShift - is the scaling shift for integral gain [-31, 31]
- f32DerivGain - is the scaled value of derivative gain
- s16DerivGainShift - is the scaling shift for derivative gain [-31, 31]

Filtration coefficient for the derivative term given by GFLIB_ControllerPIDpAW_F32_Eq7 does not need scaling since T_s and N are positive numbers.

$$f32FiltCoef = e^{-\frac{T_s N}{T_D}}$$

Equation GFLIB_ControllerPIDpAW_F32_Eq7

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The sum of the scaled proportional, integral and derivative terms gives a complete equation of the controller:

$$u_f(k) = K_{P_SC} \cdot e_f(k) + K_{I_SC} \cdot [e_f(k) + e_f(k-1)] + u_{If}(k-1) + K_{D_SC} \cdot [e_f(k) - e_f(k-1)] + K_F \cdot u_{Df}(k-1)$$

Equation GFLIB_ControllerPIDpAW_F32_Eq8

The output signal limitation is implemented in this controller. The actual output u(k) is bounded not to exceed the given limit values f32UpperLimit, f32LowerLimit. This is due to either the bounded power of the actuator or to the physical constraints of the plant.

$$u_f(k) = \begin{cases} f32UpperLimit & if \quad u_f(k) \geq f32UpperLimit \\ u_f(k) & if \quad f32LoweLimit < u_f(k) < f32UpperLimit \\ f32LowerLimit & if \quad u_f(k) \leq f32LoweLimit \end{cases}$$

Equation GFLIB_ControllerPIDpAW_F32_Eq9

When the bounds are exceeded the non-linear saturation characteristic will take effect and influence the dynamic behavior. The described limitation is implemented on the integral part accumulator (limitation during the calculation) and on the overall controller output. Therefore, if the limitation occurs, the controller output is clipped to its bounds and the wind-up occurrence of the accumulator portion is avoided by saturating the actual sum.

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PID_P_AW_DEFAULT_F32 macro.

2.22.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PID_P_AW_T_F32 trMyPID = GFLIB_CONTROLLER_PID_P_AW_DEFAULT_F32;

void main(void)
{
    tFrac32 f32ControllerPIDpAWOut;

    // Set the input error
    f32InErr = FRAC32(0.25);

    // Initialize the controller parameters
    trMyPID.f32PropGain        = FRAC32(0.1);
    trMyPID.f32IntegGain       = FRAC32(0.2);
    trMyPID.f32DerivGain       = FRAC32(0.3);
    trMyPID.f32FiltCoef        = FRAC32(0.4);
    trMyPID.s16PropGainShift   = 1;
    trMyPID.s16IntegGainShift  = 1;
    trMyPID.s16DerivGainShift  = 1;
    trMyPID.f32UpperLimit      = FRAC32(1.0);
    trMyPID.f32LowerLimit      = FRAC32(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    GFLIB_ControllerPIDpAWInit_F32(&trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWInit(&trMyPID, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIDpAWInit(&trMyPID);


    // Initialize the state variables to predefined values
    f32ControllerPIDpAWOut = FRAC32(0.5);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWSetState_F32(f32ControllerPIDpAWOut, &trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWSetState(f32ControllerPIDpAWOut, &trMyPID, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIDpAWSetState(f32ControllerPIDpAWOut, &trMyPID);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIDpAW_F32(f32InErr, &trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIDpAW(f32InErr, &trMyPID, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    f32Output = GFLIB_ControllerPIDpAW(f32InErr, &trMyPID);
}
```

2.22.4 Function GFLIB_ControllerPIDpAW_F16

2.22.4.1 Declaration

```
tFrac16 GFLIB_ControllerPIDpAW_F16(tFrac16 f16InErr,
GFLIB_CONTROLLER_PID_P_AW_T_F16 *const pParam);
```

2.22.4.2 Arguments

Table 158. GFLIB_ControllerPIDpAW_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InErr	**input**	Input error signal to the controller is a 16-bit number normalized between [-1, 1).

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 158. GFLIB_ControllerPIDpAW_F16 arguments...*continued*

Type	Name	Direction	Description
GFLIB_CONTROLLER_PID_P_AW_T_F16 *const	pParam	**input, output**	Pointer to the controller parameters structure.

2.22.4.3 Return

The function returns a 16-bit value in format Q1.15, representing the signal to be applied to the controlled system so that the input error is forced to zero.

2.22.4.4 Implementation details

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. The input and output of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

The fractional representation of both input and output signals, normalized between [-1, 1), is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation GFLIB_ControllerPIDpAW_F16_Eq1

Applying such scaling (normalization) on the proportional term of equation GFLIB_ControllerPIDpAW_eq3 results in:

$$u_{Pf}(k) = K_{P_{SC}} \cdot e_f(k) \quad where \quad K_{P_{SC}} = K_P \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIDpAW_F16_Eq2

where K_{P_sc} is the proportional gain parameter considering input/output scaling.

Similarly, scaling the integral term of equation GFLIB_ControllerPIDpAW_eq4 results in:

$$u_{If}(k) = K_{I_SC} \cdot [e_f(k) + e_f(k-1)] + u_{If}(k-1) \quad where \quad K_{I_SC} = \frac{K_I T_S}{2} \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIDpAW_F16_Eq3

where K_{I_sc} is the integral gain parameter considering input/output scaling.

And finally, scaling the derivative term, equation GFLIB_ControllerPIDpAW_eq5 results in:

$$u_{If}(k) = K_{D_SC} \cdot [e_f(k) - e_f(k-1)] + e^{-\frac{T_S N}{T_D}} \cdot u_{Df}(k-1) \quad where \quad K_{D_SC} = \frac{K_D}{T_S} \cdot \left(1 - e^{-\frac{T_S N}{T_D}}\right) \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIDpAW_F16_Eq4

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

where K_{D_sc} is the derivative gain parameter considering input/output scaling.

In case the gain parameters K_{P_sc}, K_{I_sc} or K_{D_sc} exceed the fractional range [-1, 1) or they are extremely small (which could lead to loss of precision), their value must be bit-shifted by *s16PropGainShift*, *s16IntegGainShift*, and *s16DerivGainShift*, respectively. The shift amount can be calculated as

$$s16PropGainShift = \lceil \log_2(|K_{P_SC}|) \rceil$$
$$s16IntegGainShift = \lceil \log_2(|K_{I_SC}|) \rceil$$
$$s16DerivGainShift = \lceil \log_2(|K_{D_SC}|) \rceil$$

Equation GFLIB_ControllerPIDpAW_F16_Eq5

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 15th power of two (corresponds to the Q1.15 encoding):

$$f16PropGain = \left(K_{P_SC} \cdot 2^{-s16PropGainShift}\right) \cdot 2^{15}$$
$$f16IntegGain = \left(K_{I_SC} \cdot 2^{-s16IntegGainShift}\right) \cdot 2^{15}$$
$$f16DerivGain = \left(K_{D_SC} \cdot 2^{-s16DerivGainShift}\right) \cdot 2^{15}$$

Equation GFLIB_ControllerPIDpAW_F16_Eq6

where

- f16PropGain - is the scaled value of proportional gain
- s16PropGainShift - is the scaling shift for proportional gain [-31, 31]
- f16IntegGain - is the scaled value of integral gain
- s16IntegGainShift - is the scaling shift for integral gain [-31, 31]
- f16DerivGain - is the scaled value of derivative gain
- s16DerivGainShift - is the scaling shift for derivative gain [-31, 31]

Filtration coefficient for the derivative term given by GFLIB_ControllerPIDpAW_F16_Eq7 does not need scaling since T_s and N are positive numbers.

$$f16FiltCoef = e^{-\frac{T_s N}{T_D}}$$

Equation GFLIB_ControllerPIDpAW_F16_Eq7

The sum of the scaled proportional, integral and derivative terms gives a complete equation of the controller:

$$u_f(k) = K_{P_SC} \cdot e_f(k) + K_{I_SC} \cdot [e_f(k) + e_f(k-1)] + u_{If}(k-1) + K_{D_SC} \cdot [e_f(k) - e_f(k-1)] + K_F \cdot u_{Df}(k-1)$$

Equation GFLIB_ControllerPIDpAW_F16_Eq8

The output signal limitation is implemented in this controller. The actual output u(k) is bounded not to exceed the given limit values f16UpperLimit, f16LowerLimit. This is due to either the bounded power of the actuator or to the physical constraints of the plant.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$u_f(k) = \begin{cases} f16UpperLimit & if \quad u_f(k) \geq f16UpperLimit \\ u_f(k) & if \quad f16LoweLimit < u_f(k) < f16UpperLimit \\ f16LowerLimit & if \quad u_f(k) \leq f16LoweLimit \end{cases}$$

Equation GFLIB_ControllerPIDpAW_F16_Eq9

When the bounds are exceeded the non-linear saturation characteristic will take effect and influence the dynamic behavior. The described limitation is implemented on the integral part accumulator (limitation during the calculation) and on the overall controller output. Therefore, if the limitation occurs, the controller output is limited to its bounds and the wind-up occurrence of the accumulator portion is avoided by saturating the actual sum.

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PID_P_AW_DEFAULT_F16 macro.

2.22.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PID_P_AW_T_F16 trMyPID = GFLIB_CONTROLLER_PID_P_AW_DEFAULT_F16;

void main(void)
{
    tFrac16 f16ControllerPIDpAWOut;

    // Set the input error
    f16InErr = FRAC16(0.25);

    // Initialize the controller parameters
    trMyPID.f16PropGain        = FRAC16(0.1);
    trMyPID.f16IntegGain       = FRAC16(0.2);
    trMyPID.f16DerivGain       = FRAC16(0.3);
    trMyPID.f16FiltCoef        = FRAC16(0.4);
    trMyPID.s16PropGainShift   = 1;
    trMyPID.s16IntegGainShift  = 1;
    trMyPID.s16DerivGainShift  = 1;
    trMyPID.f16UpperLimit      = FRAC16(1.0);
    trMyPID.f16LowerLimit      = FRAC16(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWInit_F16(&trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWInit(&trMyPID, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIDpAWInit(&trMyPID);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Initialize the state variables to predefined values
    f16ControllerPIDpAWOut = FRAC16(0.5);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWSetState_F16(f16ControllerPIDpAWOut, &trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWSetState(f16ControllerPIDpAWOut, &trMyPID, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIDpAWSetState(f16ControllerPIDpAWOut, &trMyPID);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIDpAW_F16(f16InErr, &trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIDpAW(f16InErr, &trMyPID, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    f16Output = GFLIB_ControllerPIDpAW(f16InErr, &trMyPID);
}
```

2.22.5 Function GFLIB_ControllerPIDpAW_FLT

2.22.5.1 Declaration

```
tFloat GFLIB_ControllerPIDpAW_FLT(tFloat fltInErr,
GFLIB_CONTROLLER_PID_P_AW_T_FLT *const pParam);
```

2.22.5.2 Arguments

Table 159. GFLIB_ControllerPIDpAW_FLT arguments

Type	Name	Direction	Description
tFloat	fltInErr	**input**	Input error signal to the controller in single precision floating format.
GFLIB_CONTROLLER_PID_P_AW_T_FLT *const	pParam	**input, output**	Pointer to the controller parameters structure.

2.22.5.3 Return

The function returns a single precision floating point value, representing the signal to be applied to the controlled system so that the input error is forced to zero.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.22.5.4 Implementation details

Output of the proportional, integral and derivative terms are computed according to equations GFLIB_ControllerPIDpAW_eq3, GFLIB_ControllerPIDpAW_eq4 and GFLIB_ControllerPIDpAW_eq5 respectively. To ensure that behavior of GFLIB_ControllerPIDpAW_FLT will be consistent with continuous-time controller shown in GFLIB_ControllerPIDpAW_fig1, controller parameters must be calculated by following equations:

$$fltPropGain = K_P$$

$$fltIntegGain = \frac{K_I T_S}{2}$$

$$fltDerivGain = \frac{K_D}{T_S} \cdot \left(1 - e^{-T_S N}\right)$$

$$fltFiltCoef = e^{-T_S N}$$

Equation GFLIB_ControllerPIDpAW_FLT_Eq1

The sum of the proportional, integral and derivative terms gives a complete equation of the controller:

$$u(k) = K_P \cdot e(k) + \frac{K_I T_S}{2} \cdot [e(k) + e(k-1)] + u_I(k-1) + \frac{K_D}{T_S} \cdot \left(1 - e^{-T_S N}\right) \cdot [e(k) - e(k-1)] + e^{-T_S N} \cdot u_D(k-1)$$

Equation GFLIB_ControllerPIDpAW_FLT_Eq2

The output signal limitation is implemented in this controller. The actual output u(k) is bounded not to exceed the given limit values fltUpperLimit, fltLowerLimit. This is due to either the bounded power of the actuator or to the physical constraints of the plant.

$$u(k) = \begin{cases} fltUpperLimit & if \quad u(k) \geq fltUpperLimit \\ u(k) & if \quad fltLoweLimit < u(k) < fltUpperLimit \\ fltLowerLimit & if \quad u(k) \leq fltLoweLimit \end{cases}$$

Equation GFLIB_ControllerPIDpAW_FLT_Eq3

When the bounds are exceeded, the non-linear saturation characteristic will take effect and influence the dynamic behavior. The described limitation is implemented on the overall controller output. Therefore, if the limitation occurs, the controller output is limited to its bounds and the wind-up occurrence of the accumulator portion is avoided by saturating the actual sum.

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PID_P_AW_DEFAULT_FLT macro.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.22.5.5 Code Example

```
#include "gflib.h"

tFloat fltInErr;
tFloat fltOutput;

GFLIB_CONTROLLER_PID_P_AW_T_FLT trMyPID = GFLIB_CONTROLLER_PID_P_AW_DEFAULT_FLT;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
void main(void)
{
    tFloat fltControllerPIDpAWOut;

    // Set the input error
    fltInErr = 0.25F;

    // Initialize the controller parameters
    trMyPID.fltPropGain        = 0.1F;
    trMyPID.fltIntegGain       = 0.2F;
    trMyPID.fltDerivGain       = 0.3F;
    trMyPID.fltFiltCoef        = 0.4F;
    trMyPID.fltUpperLimit      = 1.0F;
    trMyPID.fltLowerLimit      = -1.0F;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWInit_FLT(&trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWInit(&trMyPID, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIDpAWInit(&trMyPID);


    // Initialize the state variables to predefined values
    fltControllerPIDpAWOut = 0.5F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWSetState_FLT(fltControllerPIDpAWOut, &trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWSetState(fltControllerPIDpAWOut, &trMyPID, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIDpAWSetState(fltControllerPIDpAWOut, &trMyPID);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIDpAW_FLT(fltInErr, &trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIDpAW(fltInErr, &trMyPID, FLT);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    fltOutput = GFLIB_ControllerPIDpAW(fltInErr, &trMyPID);
}
```

2.22.6 Function GFLIB_ControllerPIDpAWInit

2.22.6.1 Description

This function clears the GFLIB_ControllerPIDpAW state variables.

Note: *The input/output pointer must contain a valid address, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.22.6.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.22.6.3 Function GFLIB_ControllerPIDpAWInit_F32

2.22.6.3.1 Declaration

```
void
GFLIB_ControllerPIDpAWInit_F32(GFLIB_CONTROLLER_PID_P_AW_T_F32
*const pParam);
```

2.22.6.3.2 Arguments

Table 160. GFLIB_ControllerPIDpAWInit_F32 arguments

Type	Name	Direction	Description
GFLIB_CONTROLLER_PID_P_AW_T_F32 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIDpAW state.

2.22.6.3.3 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PID_P_AW_T_F32 trMyPID = GFLIB_CONTROLLER_PID_P_AW_DEFAULT_F32;

void main(void)
{
    tFrac32 f32ControllerPIDpAWOut;

    // Set the input error
    f32InErr = FRAC32(0.25);

    // Initialize the controller parameters
    trMyPID.f32PropGain        = FRAC32(0.1);
    trMyPID.f32IntegGain       = FRAC32(0.2);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    trMyPID.f32DerivGain        = FRAC32(0.3);
    trMyPID.f32FiltCoef         = FRAC32(0.4);
    trMyPID.s16PropGainShift    = 1;
    trMyPID.s16IntegGainShift   = 1;
    trMyPID.s16DerivGainShift   = 1;
    trMyPID.f32UpperLimit       = FRAC32(1.0);
    trMyPID.f32LowerLimit       = FRAC32(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWInit_F32(&trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWInit(&trMyPID, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIDpAWInit(&trMyPID);


    // Initialize the state variables to predefined values
    f32ControllerPIDpAWOut = FRAC32(0.5);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWSetState_F32(f32ControllerPIDpAWOut, &trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWSetState(f32ControllerPIDpAWOut, &trMyPID, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIDpAWSetState(f32ControllerPIDpAWOut, &trMyPID);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIDpAW_F32(f32InErr, &trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIDpAW(f32InErr, &trMyPID, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    f32Output = GFLIB_ControllerPIDpAW(f32InErr, &trMyPID);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.22.6.4 Function GFLIB_ControllerPIDpAWInit_F16

2.22.6.4.1 Declaration

```
void
GFLIB_ControllerPIDpAWInit_F16(GFLIB_CONTROLLER_PID_P_AW_T_F16
*const pParam);
```

2.22.6.4.2 Arguments

Table 161. GFLIB_ControllerPIDpAWInit_F16 arguments

Type	Name	Direction	Description
GFLIB_CONTROLLER_PID_P_AW_T_F16 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIDpAW state.

2.22.6.4.3 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PID_P_AW_T_F16 trMyPID = GFLIB_CONTROLLER_PID_P_AW_DEFAULT_F16;

void main(void)
{
    tFrac16 f16ControllerPIDpAWOut;

    // Set the input error
    f16InErr = FRAC16(0.25);

    // Initialize the controller parameters
    trMyPID.f16PropGain        = FRAC16(0.1);
    trMyPID.f16IntegGain       = FRAC16(0.2);
    trMyPID.f16DerivGain       = FRAC16(0.3);
    trMyPID.f16FiltCoef        = FRAC16(0.4);
    trMyPID.s16PropGainShift   = 1;
    trMyPID.s16IntegGainShift  = 1;
    trMyPID.s16DerivGainShift  = 1;
    trMyPID.f16UpperLimit      = FRAC16(1.0);
    trMyPID.f16LowerLimit      = FRAC16(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWInit_F16(&trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWInit(&trMyPID, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIDpAWInit(&trMyPID);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Initialize the state variables to predefined values
    f16ControllerPIDpAWOut = FRAC16(0.5);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWSetState_F16(f16ControllerPIDpAWOut, &trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWSetState(f16ControllerPIDpAWOut, &trMyPID, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIDpAWSetState(f16ControllerPIDpAWOut, &trMyPID);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIDpAW_F16(f16InErr, &trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIDpAW(f16InErr, &trMyPID, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    f16Output = GFLIB_ControllerPIDpAW(f16InErr, &trMyPID);
}
```

2.22.6.5 Function GFLIB_ControllerPIDpAWInit_FLT

2.22.6.5.1 Declaration

```
void
GFLIB_ControllerPIDpAWInit_FLT(GFLIB_CONTROLLER_PID_P_AW_T_FLT
*const pParam);
```

2.22.6.5.2 Arguments

Table 162. GFLIB_ControllerPIDpAWInit_FLT arguments

Type	Name	Direction	Description
GFLIB_CONTROLLER_PID_P_AW_T_FLT *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIDpAW state.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.22.6.5.3 Code Example

```
#include "gflib.h"

tFloat fltInErr;
tFloat fltOutput;

GFLIB_CONTROLLER_PID_P_AW_T_FLT trMyPID = GFLIB_CONTROLLER_PID_P_AW_DEFAULT_FLT;

void main(void)
{
    tFloat fltControllerPIDpAWOut;

    // Set the input error
    fltInErr = 0.25F;

    // Initialize the controller parameters
    trMyPID.fltPropGain        = 0.1F;
    trMyPID.fltIntegGain       = 0.2F;
    trMyPID.fltDerivGain       = 0.3F;
    trMyPID.fltFiltCoef        = 0.4F;
    trMyPID.fltUpperLimit      = 1.0F;
    trMyPID.fltLowerLimit      = -1.0F;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWInit_FLT(&trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWInit(&trMyPID, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIDpAWInit(&trMyPID);


    // Initialize the state variables to predefined values
    fltControllerPIDpAWOut = 0.5F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWSetState_FLT(fltControllerPIDpAWOut, &trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWSetState(fltControllerPIDpAWOut, &trMyPID, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIDpAWSetState(fltControllerPIDpAWOut, &trMyPID);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIDpAW_FLT(fltInErr, &trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIDpAW(fltInErr, &trMyPID, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    fltOutput = GFLIB_ControllerPIDpAW(fltInErr, &trMyPID);
}
```

2.22.7 Function GFLIB_ControllerPIDpAWSetState

2.22.7.1 Description

This function initializes the GFLIB_ControllerPIDpAW state variables to achieve the required output values.

Note: *The input/output pointer must contain a valid address, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.22.7.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.22.7.3 Function GFLIB_ControllerPIDpAWSetState_F32

2.22.7.3.1 Declaration

```
void GFLIB_ControllerPIDpAWSetState_F32(tFrac32
f32ControllerPIDpAWOut, GFLIB_CONTROLLER_PID_P_AW_T_F32 *const
pParam);
```

2.22.7.3.2 Arguments

Table 163. GFLIB_ControllerPIDpAWSetState_F32 arguments

Type	Name	Direction	Description
tFrac32	f32ControllerPIDpAWOut	**input**	Required output of the GFLIB_ ControllerPIDpAW.
GFLIB_ CONTROLLER_ PID_P_AW_T_ F32 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIDpAW state.

2.22.7.3.3 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PID_P_AW_T_F32 trMyPID = GFLIB_CONTROLLER_PID_P_AW_DEFAULT_F32;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
void main(void)
{
    tFrac32 f32ControllerPIDpAWOut;

    // Set the input error
    f32InErr = FRAC32(0.25);

    // Initialize the controller parameters
    trMyPID.f32PropGain        = FRAC32(0.1);
    trMyPID.f32IntegGain       = FRAC32(0.2);
    trMyPID.f32DerivGain       = FRAC32(0.3);
    trMyPID.f32FiltCoef        = FRAC32(0.4);
    trMyPID.s16PropGainShift   = 1;
    trMyPID.s16IntegGainShift  = 1;
    trMyPID.s16DerivGainShift  = 1;
    trMyPID.f32UpperLimit      = FRAC32(1.0);
    trMyPID.f32LowerLimit      = FRAC32(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWInit_F32(&trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWInit(&trMyPID, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIDpAWInit(&trMyPID);


    // Initialize the state variables to predefined values
    f32ControllerPIDpAWOut = FRAC32(0.5);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWSetState_F32(f32ControllerPIDpAWOut, &trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWSetState(f32ControllerPIDpAWOut, &trMyPID, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIDpAWSetState(f32ControllerPIDpAWOut, &trMyPID);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIDpAW_F32(f32InErr, &trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIDpAW(f32InErr, &trMyPID, F32);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    f32Output = GFLIB_ControllerPIDpAW(f32InErr, &trMyPID);
}
```

2.22.7.4 Function GFLIB_ControllerPIDpAWSetState_F16

2.22.7.4.1 Declaration

```
void GFLIB_ControllerPIDpAWSetState_F16(tFrac16
f16ControllerPIDpAWOut, GFLIB_CONTROLLER_PID_P_AW_T_F16 *const
pParam);
```

2.22.7.4.2 Arguments

Table 164. GFLIB_ControllerPIDpAWSetState_F16 arguments

Type	Name	Direction	Description
tFrac16	f16ControllerPIDpAWOut	**input**	Required output of the GFLIB_ ControllerPIDpAW.
GFLIB_ CONTROLLER_ PID_P_AW_T_ F16 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIDpAW state.

2.22.7.4.3 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PID_P_AW_T_F16 trMyPID = GFLIB_CONTROLLER_PID_P_AW_DEFAULT_F16;

void main(void)
{
    tFrac16 f16ControllerPIDpAWOut;

    // Set the input error
    f16InErr = FRAC16(0.25);

    // Initialize the controller parameters
    trMyPID.f16PropGain         = FRAC16(0.1);
    trMyPID.f16IntegGain        = FRAC16(0.2);
    trMyPID.f16DerivGain        = FRAC16(0.3);
    trMyPID.f16FiltCoef         = FRAC16(0.4);
    trMyPID.s16PropGainShift    = 1;
    trMyPID.s16IntegGainShift   = 1;
    trMyPID.s16DerivGainShift   = 1;
    trMyPID.f16UpperLimit       = FRAC16(1.0);
    trMyPID.f16LowerLimit       = FRAC16(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWInit_F16(&trMyPID);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWInit(&trMyPID, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIDpAWInit(&trMyPID);


    // Initialize the state variables to predefined values
    f16ControllerPIDpAWOut = FRAC16(0.5);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWSetState_F16(f16ControllerPIDpAWOut, &trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWSetState(f16ControllerPIDpAWOut, &trMyPID, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIDpAWSetState(f16ControllerPIDpAWOut, &trMyPID);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIDpAW_F16(f16InErr, &trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIDpAW(f16InErr, &trMyPID, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    f16Output = GFLIB_ControllerPIDpAW(f16InErr, &trMyPID);
}
```

2.22.7.5 Function GFLIB_ControllerPIDpAWSetState_FLT

2.22.7.5.1 Declaration

```
void GFLIB_ControllerPIDpAWSetState_FLT(tFloat
fltControllerPIDpAWOut, GFLIB_CONTROLLER_PID_P_AW_T_FLT *const
pParam);
```

2.22.7.5.2 Arguments

Table 165. GFLIB_ControllerPIDpAWSetState_FLT arguments

Type	Name	Direction	Description
tFloat	fltControllerPIDpAWOut	Input	Required output of the GFLIB_ ControllerPIDpAW.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 165. GFLIB_ControllerPIDpAWSetState_FLT arguments...*continued*

Type	Name	Direction	Description
GFLIB_CONTROLLER_PID_P_AW_T_FLT *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIDpAW state.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.22.7.5.3 Code Example

```
#include "gflib.h"

tFloat fltInErr;
tFloat fltOutput;

GFLIB_CONTROLLER_PID_P_AW_T_FLT trMyPID = GFLIB_CONTROLLER_PID_P_AW_DEFAULT_FLT;

void main(void)
{
    tFloat fltControllerPIDpAWOut;

    // Set the input error
    fltInErr = 0.25F;

    // Initialize the controller parameters
    trMyPID.fltPropGain        = 0.1F;
    trMyPID.fltIntegGain       = 0.2F;
    trMyPID.fltDerivGain       = 0.3F;
    trMyPID.fltFiltCoef        = 0.4F;
    trMyPID.fltUpperLimit      = 1.0F;
    trMyPID.fltLowerLimit      = -1.0F;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWInit_FLT(&trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWInit(&trMyPID, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIDpAWInit(&trMyPID);


    // Initialize the state variables to predefined values
    fltControllerPIDpAWOut = 0.5F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIDpAWSetState_FLT(fltControllerPIDpAWOut, &trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    GFLIB_ControllerPIDpAWSetState(fltControllerPIDpAWOut, &trMyPID, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIDpAWSetState(fltControllerPIDpAWOut, &trMyPID);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIDpAW_FLT(fltInErr, &trMyPID);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIDpAW(fltInErr, &trMyPID, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    fltOutput = GFLIB_ControllerPIDpAW(fltInErr, &trMyPID);
}
```

2.23 Function GFLIB_ControllerPIp

This function calculates a parallel form of the Proportional-Integral controller, without integral anti-windup.

2.23.1 Description

A PI controller attempts to correct the error between a measured process variable and a desired set-point by calculating and then outputting a corrective action that can adjust the process accordingly. The GFLIB_ControllerPIp function calculates the Proportional-Integral (PI) algorithm according to the equations below. The PI algorithm is implemented in the parallel (non-interacting) form, allowing the user to define the P and I parameters independently without interaction. An anti-windup strategy is not implemented in this function.

The PI algorithm in the continuous time domain can be described as:

$$u(t) = e(t) \cdot K_P + K_I \int_0^t e(t)\,dt$$

Equation GFLIB_ControllerPIp_Eq1

where

- e(t) - input error in the continuous time domain
- u(t) - controller output in the continuous time domain
- K_P - proportional gain
- K_I - integral gain

Equation GFLIB_ControllerPIp_Eq1 can be described using the Laplace transform as follows:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$H(s) = \frac{U(s)}{E(s)} = K_P + K_I \cdot \frac{1}{s}$$

Equation GFLIB_ControllerPIp_Eq2

The proportional part of equation GFLIB_ControllerPIp_Eq2 is transformed into the discrete time domain simply as:

$$u_P(k) = K_P \cdot e(k)$$

Equation GFLIB_ControllerPIp_Eq3

Transforming the integral part of equation GFLIB_ControllerPIp_Eq2 into a discrete time domain using the bilinear method (also known as trapezoidal approximation) leads to the following equation:

$$u_I(k) = u_I(k-1) + e(k) \cdot \frac{K_I T_S}{2} + e(k-1) \cdot \frac{K_I T_S}{2}$$

Equation GFLIB_ControllerPIp_Eq4

where T_s [sec] is the sampling period.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.23.2 Re-entrancy

The function is re-entrant for a different `pParam`.

2.23.3 Function GFLIB_ControllerPIp_F32

2.23.3.1 Declaration

```
tFrac32 GFLIB_ControllerPIp_F32(tFrac32 f32InErr,
GFLIB_CONTROLLER_PI_P_T_F32 *const pParam);
```

2.23.3.2 Arguments

Table 166. GFLIB_ControllerPIp_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InErr	**input**	Input error signal to the controller is a 32-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PI_P_T_F32 *const	pParam	**input, output**	Pointer to the controller parameters structure.

2.23.3.3 Return

The function returns a 32-bit value in format Q1.31, representing the signal to be applied to the controlled system so that the input error is forced to zero.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.23.3.4 Implementation details

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. The input and output of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

The fractional representation of both input and output signals, normalized between [-1, 1), is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$
$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation GFLIB_ControllerPIp_F32_Eq1

Applying such scaling (normalization) on the proportional term of equation GFLIB_ControllerPIp_eq3 results in:

$$u_{Pf}(k) = e_f(k) \cdot K_{P_SC} \quad where \quad K_{P_SC} = K_P \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIp_F32_Eq2

where K_{P_sc} is the proportional gain parameter considering input/output scaling.

Similarly, scaling the integral term of equation GFLIB_ControllerPIp_eq4 results in:

$$u_{If}(k) = u_{If}(k-1) + K_{I_SC} \cdot e_f(k) + K_{I_SC} \cdot e_f(k-1) \quad where \quad K_{I_SC} = \frac{K_I T_S}{2} \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIp_F32_Eq3

where K_{I_sc} is the integral gain parameter considering input/output scaling.

The sum of the scaled proportional and integral terms gives a complete equation of the controller:

$$u_f(k) = e_f(k) \cdot K_{P_SC} + u_{If}(k-1) + K_{I_SC} \cdot e_f(k) + K_{I_SC} \cdot e_f(k-1)$$

Equation GFLIB_ControllerPIp_F32_Eq4

In case the gain parameters K_{P_sc} or K_{I_sc} exceed the fractional range [-1, 1) or they are extremely small (which could lead to loss of precision), their value must be bit-shifted by *s16PropGainShift* and *s16IntegGainShift*, respectively. The shift amount can be calculated as

$$s16PropGainShift = \lceil \log_2(|K_{P_SC}|) \rceil$$
$$s16IntegGainShift = \lceil \log_2(|K_{I_SC}|) \rceil$$

Equation GFLIB_ControllerPIp_F32_Eq5

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 31st power of two (corresponds to the Q1.31 encoding):

$$f32PropGain = \left(K_{P_SC} \cdot 2^{-s16PropGainShift}\right) \cdot 2^{31}$$
$$f32IntegGain = \left(K_{I_SC} \cdot 2^{-s16IntegGainShift}\right) \cdot 2^{31}$$

Equation GFLIB_ControllerPIp_F32_Eq6

where

- f32PropGain - is the scaled value of proportional gain
- s16PropGainShift - is the scaling shift for proportional gain [-31,31]
- f32IntegGain - is the scaled value of integral gain
- s16IntegGainShift - is the scaling shift for integral gain [-31,31]

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PI_P_DEFAULT_F32 macro.

2.23.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PI_P_T_F32 trMyPI = GFLIB_CONTROLLER_PI_P_DEFAULT_F32;

void main(void)
{
    tFrac32 f32ControllerPIpOut;

    // Set the input error
    f32InErr = FRAC32(0.25);

    // Initialize the controller parameters
    trMyPI.f32PropGain        = FRAC32(0.01);
    trMyPI.f32IntegGain       = FRAC32(0.02);
    trMyPI.s16PropGainShift   = 1;
    trMyPI.s16IntegGainShift  = 1;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpInit_F32(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpInit(&trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpInit(&trMyPI);


    // Initialize the state variables to predefined values
    f32ControllerPIpOut = FRAC32(0.03);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpSetState_F32(f32ControllerPIpOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpSetState(f32ControllerPIpOut, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpSetState(f32ControllerPIpOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIp_F32(f32InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIp(f32InErr, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    f32Output = GFLIB_ControllerPIp(f32InErr, &trMyPI);
}
```

2.23.4 Function GFLIB_ControllerPIp_F16

2.23.4.1 Declaration

```
tFrac16 GFLIB_ControllerPIp_F16(tFrac16 f16InErr,
GFLIB_CONTROLLER_PI_P_T_F16 *const pParam);
```

2.23.4.2 Arguments

Table 167. GFLIB_ControllerPIp_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InErr	**input**	Input error signal to the controller is a 16-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PI_P_T_F16 *const	pParam	**input, output**	Pointer to the controller parameters structure.

2.23.4.3 Return

The function returns a 16-bit value in format Q1.15, representing the signal to be applied to the controlled system so that the input error is forced to zero.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.23.4.4 Implementation details

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. The input and output of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

The fractional representation of both input and output signals, normalized between [-1, 1), is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$
$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation GFLIB_ControllerPIp_F16_Eq1

Applying such scaling (normalization) on the proportional term of equation GFLIB_ControllerPIp_eq3 results in:

$$u_{Pf}(k) = e_f(k) \cdot K_{P_SC} \quad where \quad K_{P_SC} = K_P \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIp_F16_Eq2

where K_{P_sc} is the proportional gain parameter considering input/output scaling.

Similarly, scaling the integral term of equation GFLIB_ControllerPIp_eq4 results in:

$$u_{If}(k) = u_{If}(k-1) + K_{I_SC} \cdot e_f(k) + K_{I_SC} \cdot e_f(k-1) \quad where \quad K_{I_SC} = \frac{K_I T_S}{2} \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIp_F16_Eq3

where K_{I_sc} is the integral gain parameter considering input/output scaling.

The sum of the scaled proportional and integral terms gives a complete equation of the controller:

$$u_f(k) = e_f(k) \cdot K_{P_SC} + u_{If}(k-1) + K_{I_SC} \cdot e_f(k) + K_{I_SC} \cdot e_f(k-1)$$

Equation GFLIB_ControllerPIp_F16_Eq4

In case the gain parameters K_{P_sc} or K_{I_sc} exceed the fractional range [-1, 1) or they are extremely small (which could lead to loss of precision), their value must be bit-shifted by *s16PropGainShift* and *s16IntegGainShift*, respectively. The shift amount can be calculated as

$$s16PropGainShift = \lceil \log_2(|K_{P_SC}|) \rceil$$
$$s16IntegGainShift = \lceil \log_2(|K_{I_SC}|) \rceil$$

Equation GFLIB_ControllerPIp_F16_Eq5

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 15th power of two (corresponds to the Q1.15 encoding):

$$f16PropGain = \left(K_{P_SC} \cdot 2^{-s16PropGainShift}\right) \cdot 2^{15}$$
$$f16IntegGain = \left(K_{I_SC} \cdot 2^{-s16IntegGainShift}\right) \cdot 2^{15}$$

Equation GFLIB_ControllerPIp_F16_Eq6

where

- f16PropGain - is the scaled value of proportional gain
- s16PropGainShift - is the scaling shift for proportional gain [-15, 15)
- f16IntegGain - is the scaled value of integral gain
- s16IntegGainShift - is the scaling shift for integral gain [-15, 15)

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PI_P_DEFAULT_F16 macro.

2.23.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PI_P_T_F16 trMyPI = GFLIB_CONTROLLER_PI_P_DEFAULT_F16;

void main(void)
{
    tFrac16 f16ControllerPIpOut;

    // Set the input error
    f16InErr = FRAC16(0.25);

    // Initialize the controller parameters
    trMyPI.f16PropGain        = FRAC16(0.01);
    trMyPI.f16IntegGain       = FRAC16(0.02);
    trMyPI.s16PropGainShift   = 1;
    trMyPI.s16IntegGainShift  = 1;
    trMyPI.f16UpperLimit      = FRAC16(1.0);
    trMyPI.f16LowerLimit      = FRAC16(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpInit_F16(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpInit(&trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpInit(&trMyPI);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Initialize the state variables to predefined values
    f16ControllerPIpOut = FRAC16(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpSetState_F16(f16ControllerPIpOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpSetState(f16ControllerPIpOut, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpSetState(f16ControllerPIpOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIp_F16(f16InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIp(f16InErr, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    f16Output = GFLIB_ControllerPIp(f16InErr, &trMyPI);
}
```

2.23.5 Function GFLIB_ControllerPIp_FLT

2.23.5.1 Declaration

```
tFloat GFLIB_ControllerPIp_FLT(tFloat fltInErr,
GFLIB_CONTROLLER_PI_P_T_FLT *const pParam);
```

2.23.5.2 Arguments

Table 168. GFLIB_ControllerPIp_FLT arguments

Type	Name	Direction	Description
tFloat	fltInErr	**input**	Input error signal to the controller in single precision floating format.
GFLIB_CONTROLLER_PI_P_T_FLT *const	pParam	**input, output**	Pointer to the controller parameters structure.

2.23.5.3 Return

The function returns a single precision floating point value, representing the signal to be applied to the controlled system so that the input error is forced to zero.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.23.5.4 Implementation details

The sum of the scaled proportional and integral terms gives a complete equation of the controller:

$$u_f(k) = e_f(k) \cdot K_{P_SC} + u_{If}(k-1) + K_{I_SC} \cdot e_f(k) + K_{I_SC} \cdot e_f(k-1)$$

Equation GFLIB_ControllerPIp_FLT_Eq1

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PI_P_DEFAULT_FLT macro.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.23.6 Function GFLIB_ControllerPIpInit

2.23.6.1 Description

This function clears the GFLIB_ControllerPIp state variables.

Note: *The input/output pointer must contain a valid address, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.23.6.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.23.6.3 Function GFLIB_ControllerPIpInit_F32

2.23.6.3.1 Declaration

```
void GFLIB_ControllerPIpInit_F32(GFLIB_CONTROLLER_PI_P_T_F32
*const pParam);
```

2.23.6.3.2 Arguments

Table 169. GFLIB_ControllerPIpInit_F32 arguments

Type	Name	Direction	Description
GFLIB_CONTROLLER_PI_P_T_F32 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIp state.

2.23.6.3.3 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PI_P_T_F32 trMyPI = GFLIB_CONTROLLER_PI_P_DEFAULT_F32;

void main(void)
{
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    tFrac32 f32ControllerPIpOut;

    // Set the input error
    f32InErr = FRAC32(0.25);

    // Initialize the controller parameters
    trMyPI.f32PropGain        = FRAC32(0.01);
    trMyPI.f32IntegGain       = FRAC32(0.02);
    trMyPI.s16PropGainShift   = 1;
    trMyPI.s16IntegGainShift  = 1;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpInit_F32(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpInit(&trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpInit(&trMyPI);


    // Initialize the state variables to predefined values
    f32ControllerPIpOut = FRAC32(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpSetState_F32(f32ControllerPIpOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpSetState(f32ControllerPIpOut, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpSetState(f32ControllerPIpOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIp_F32(f32InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIp(f32InErr, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    f32Output = GFLIB_ControllerPIp(f32InErr, &trMyPI);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.23.6.4 Function GFLIB_ControllerPIpInit_F16

2.23.6.4.1 Declaration

```
void GFLIB_ControllerPIpInit_F16(GFLIB_CONTROLLER_PI_P_T_F16
*const pParam);
```

2.23.6.4.2 Arguments

Table 170. GFLIB_ControllerPIpInit_F16 arguments

Type	Name	Direction	Description
GFLIB_CONTROLLER_PI_P_T_F16 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIp state.

2.23.6.4.3 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PI_P_T_F16 trMyPI = GFLIB_CONTROLLER_PI_P_DEFAULT_F16;

void main(void)
{
    tFrac16 f16ControllerPIpOut;

    // Set the input error
    f16InErr = FRAC16(0.25);

    // Initialize the controller parameters
    trMyPI.f16PropGain        = FRAC16(0.01);
    trMyPI.f16IntegGain       = FRAC16(0.02);
    trMyPI.s16PropGainShift   = 1;
    trMyPI.s16IntegGainShift  = 1;
    trMyPI.f16UpperLimit      = FRAC16(1.0);
    trMyPI.f16LowerLimit      = FRAC16(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpInit_F16(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpInit(&trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpInit(&trMyPI);


    // Initialize the state variables to predefined values
    f16ControllerPIpOut = FRAC16(0.03);
    // Alternative 1: API call with postfix
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // (only one alternative shall be used).
    GFLIB_ControllerPIpSetState_F16(f16ControllerPIpOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpSetState(f16ControllerPIpOut, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpSetState(f16ControllerPIpOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIp_F16(f16InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIp(f16InErr, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    f16Output = GFLIB_ControllerPIp(f16InErr, &trMyPI);
}
```

2.23.6.5 Function GFLIB_ControllerPIpInit_FLT

2.23.6.5.1 Declaration

```
void GFLIB_ControllerPIpInit_FLT(GFLIB_CONTROLLER_PI_P_T_FLT
*const pParam);
```

2.23.6.5.2 Arguments

Table 171. GFLIB_ControllerPIpInit_FLT arguments

Type	Name	Direction	Description
GFLIB_CONTROLLER_PI_P_T_FLT *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIp state.

2.23.6.5.3 Code Example

```
#include "gflib.h"

tFloat fltInErr;
tFloat fltOutput;

GFLIB_CONTROLLER_PI_P_T_FLT trMyPI = GFLIB_CONTROLLER_PI_P_DEFAULT_FLT;

void main(void)
{
    tFloat fltControllerPIpOut;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Set the input error
    fltInErr = 0.25F;

    // Initialize the controller parameters
    trMyPI.fltPropGain   =  0.04F;
    trMyPI.fltIntegGain  =  0.02F;
    trMyPI.fltUpperLimit =  1.0F;
    trMyPI.fltLowerLimit = -1.0F;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpInit_FLT(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpInit(&trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIpInit(&trMyPI);


    // Initialize the state variables to predefined values
    fltControllerPIpOut = 0.03F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpSetState_FLT(fltControllerPIpOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpSetState(fltControllerPIpOut, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIpSetState(fltControllerPIpOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIp_FLT(fltInErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIp(fltInErr, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    fltOutput = GFLIB_ControllerPIp(fltInErr, &trMyPI);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.23.7 Function GFLIB_ControllerPIpSetState

2.23.7.1 Description

This function initializes the GFLIB_ControllerPIp state variables to achieve the required output values.

Note: *The input/output pointer must contain a valid address, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.23.7.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.23.7.3 Function GFLIB_ControllerPIpSetState_F32

2.23.7.3.1 Declaration

```
void GFLIB_ControllerPIpSetState_F32(tFrac32 f32ControllerPIpOut,
GFLIB_CONTROLLER_PI_P_T_F32 *const pParam);
```

2.23.7.3.2 Arguments

Table 172. GFLIB_ControllerPIpSetState_F32 arguments

Type	Name	Direction	Description
tFrac32	f32ControllerPIpOut	**input**	Required output of the GFLIB_ControllerPIp.
GFLIB_CONTROLLER_PI_P_T_F32 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIp state.

2.23.7.3.3 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PI_P_T_F32 trMyPI = GFLIB_CONTROLLER_PI_P_DEFAULT_F32;

void main(void)
{
    tFrac32 f32ControllerPIpOut;

    // Set the input error
    f32InErr = FRAC32(0.25);

    // Initialize the controller parameters
    trMyPI.f32PropGain        = FRAC32(0.01);
    trMyPI.f32IntegGain       = FRAC32(0.02);
    trMyPI.s16PropGainShift   = 1;
    trMyPI.s16IntegGainShift  = 1;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpInit_F32(&trMyPI);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpInit(&trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpInit(&trMyPI);


    // Initialize the state variables to predefined values
    f32ControllerPIpOut = FRAC32(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpSetState_F32(f32ControllerPIpOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpSetState(f32ControllerPIpOut, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpSetState(f32ControllerPIpOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIp_F32(f32InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIp(f32InErr, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    f32Output = GFLIB_ControllerPIp(f32InErr, &trMyPI);
}
```

2.23.7.4 Function GFLIB_ControllerPIpSetState_F16

2.23.7.4.1 Declaration

```
void GFLIB_ControllerPIpSetState_F16(tFrac16 f16ControllerPIpOut,
GFLIB_CONTROLLER_PI_P_T_F16 *const pParam);
```

2.23.7.4.2 Arguments

Table 173. GFLIB_ControllerPIpSetState_F16 arguments

Type	Name	Direction	Description
tFrac16	f16ControllerPIpOut	**input**	Required output of the GFLIB_ControllerPIp.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 173. GFLIB_ControllerPIpSetState_F16 arguments...*continued*

Type	Name	Direction	Description
GFLIB_CONTROLLER_PI_P_T_F16 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIp state.

2.23.7.4.3 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PI_P_T_F16 trMyPI = GFLIB_CONTROLLER_PI_P_DEFAULT_F16;

void main(void)
{
    tFrac16 f16ControllerPIpOut;

    // Set the input error
    f16InErr = FRAC16(0.25);

    // Initialize the controller parameters
    trMyPI.f16PropGain        = FRAC16(0.01);
    trMyPI.f16IntegGain       = FRAC16(0.02);
    trMyPI.s16PropGainShift   = 1;
    trMyPI.s16IntegGainShift  = 1;
    trMyPI.f16UpperLimit      = FRAC16(1.0);
    trMyPI.f16LowerLimit      = FRAC16(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpInit_F16(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpInit(&trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpInit(&trMyPI);


    // Initialize the state variables to predefined values
    f16ControllerPIpOut = FRAC16(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpSetState_F16(f16ControllerPIpOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpSetState(f16ControllerPIpOut, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    GFLIB_ControllerPIpSetState(f16ControllerPIpOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIp_F16(f16InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIp(f16InErr, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    f16Output = GFLIB_ControllerPIp(f16InErr, &trMyPI);
}
```

2.23.7.5 Function GFLIB_ControllerPIpSetState_FLT

2.23.7.5.1 Declaration

```
void GFLIB_ControllerPIpSetState_FLT(tFloat fltControllerPIpOut,
GFLIB_CONTROLLER_PI_P_T_FLT *const pParam);
```

2.23.7.5.2 Arguments

Table 174. GFLIB_ControllerPIpSetState_FLT arguments

Type	Name	Direction	Description
tFloat	fltControllerPIpOut	**input**	Required output of the GFLIB_ControllerPIp.
GFLIB_CONTROLLER_PI_P_T_FLT *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIp state.

2.23.7.5.3 Code Example

```
#include "gflib.h"

tFloat fltInErr;
tFloat fltOutput;

GFLIB_CONTROLLER_PI_P_T_FLT trMyPI = GFLIB_CONTROLLER_PI_P_DEFAULT_FLT;

void main(void)
{
    tFloat fltControllerPIpOut;

    // Set the input error
    fltInErr = 0.25F;

    // Initialize the controller parameters
    trMyPI.fltPropGain    =  0.04F;
    trMyPI.fltIntegGain   =  0.02F;
    trMyPI.fltUpperLimit  =  1.0F;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    trMyPI.fltLowerLimit = -1.0F;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpInit_FLT(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpInit(&trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIpInit(&trMyPI);


    // Initialize the state variables to predefined values
    fltControllerPIpOut = 0.03F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpSetState_FLT(fltControllerPIpOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpSetState(fltControllerPIpOut, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIpSetState(fltControllerPIpOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIp_FLT(fltInErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIp(fltInErr, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    fltOutput = GFLIB_ControllerPIp(fltInErr, &trMyPI);
}
```

2.24 Function GFLIB_ControllerPIpAW

The function calculates the parallel form of the Proportional-Integral (PI) controller with implemented integral anti-windup functionality.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.24.1 Description

A PI controller attempts to correct the error between a measured process variable and a desired set-point by calculating and then outputting a corrective action that can adjust the process accordingly. The GFLIB_ControllerPIpAW function calculates the Proportional-Integral (PI) algorithm according to the equations below. The PI algorithm is implemented in the parallel (non-interacting) form, allowing the user to define the P and I parameters independently without interaction. The controller output is limited and the limit values (UpperLimit and LowerLimit) are defined by the user. The PI controller algorithm also returns a limitation flag. This flag (u16LimitFlag) is a member of the structure of the PI controller parameters. If the PI controller output reaches the upper or lower limit then u16LimitFlag = 1, otherwise u16LimitFlag = 0 (integer values). An anti-windup strategy is implemented by limiting the integral portion. The integral state is limited by the controller limits, in the same way as the controller output.

The PI algorithm in the continuous time domain can be described as:

$$u(t) = e(t) \cdot K_P + K_I \int_0^t e(t)\,dt$$

Equation GFLIB_ControllerPIpAW_Eq1

where

- e(t) - input error in the continuous time domain
- u(t) - controller output in the continuous time domain
- K_P - proportional gain
- K_I - integral gain

Equation GFLIB_ControllerPIpAW_Eq1 can be described using the Laplace transform as follows:

$$H(s) = \frac{U(s)}{E(s)} = K_P + K_I \cdot \frac{1}{s}$$

Equation GFLIB_ControllerPIpAW_Eq2

The proportional part of equation GFLIB_ControllerPIpAW_Eq2 is transformed into the discrete time domain simply as:

$$u_P(k) = K_P \cdot e(k)$$

Equation GFLIB_ControllerPIpAW_Eq3

Transforming the integral part of equation GFLIB_ControllerPIpAW_Eq2 into a discrete time domain using the bilinear method (also known as trapezoidal approximation) leads to the following equation:

$$u_I(k) = u_I(k-1) + e(k) \cdot \frac{K_I T_S}{2} + e(k-1) \cdot \frac{K_I T_S}{2}$$

Equation GFLIB_ControllerPIpAW_Eq4

where T_s [sec] is the sampling period.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.24.2 Re-entrancy

The function is re-entrant for a different `pParam`.

2.24.3 Function GFLIB_ControllerPIpAW_F32

2.24.3.1 Declaration

```
tFrac32 GFLIB_ControllerPIpAW_F32(tFrac32 f32InErr,
GFLIB_CONTROLLER_PIAW_P_T_F32 *const pParam);
```

2.24.3.2 Arguments

Table 175. GFLIB_ControllerPIpAW_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InErr	**input**	Input error signal to the controller is a 32-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PIAW_P_T_F32 *const	pParam	**input, output**	Pointer to the controller parameters structure.

2.24.3.3 Return

The function returns a 32-bit value in format Q1.31, representing the signal to be applied to the controlled system so that the input error is forced to zero.

2.24.3.4 Implementation details

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. The input and output of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

The fractional representation of both input and output signals, normalized between [-1, 1), is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$
$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation GFLIB_ControllerPIpAW_F32_Eq1

Applying such scaling (normalization) on the proportional term of equation GFLIB_ControllerPIpAW_eq3 results in:

$$u_{Pf}(k) = e_f(k) \cdot K_{P_SC} \quad where \quad K_{P_SC} = K_P \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIpAW_F32_Eq2

where K_{P_sc} is the proportional gain parameter considering input/output scaling.

Similarly, scaling the integral term of equation GFLIB_ControllerPIpAW_eq4 results in:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$u_{If}(k) = u_{If}(k-1) + K_{I_SC} \cdot e_f(k) + K_{I_SC} \cdot e_f(k-1) \quad where \quad K_{I_SC} = \frac{K_I T_S}{2} \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIpAW_F32_Eq3

where K_{I_sc} is the integral gain parameter considering input/output scaling. The sum of the scaled proportional and integral terms gives a complete equation of the controller.

In case the gain parameters K_{P_sc} or K_{I_sc} exceed the fractional range [-1, 1) or they are extremely small (which could lead to loss of precision), their value must be bit-shifted by *s16PropGainShift* and *s16IntegGainShift*, respectively. The shift amount can be calculated as

$$s16PropGainShift = \lceil \log_2(|K_{P_SC}|) \rceil$$
$$s16IntegGainShift = \lceil \log_2(|K_{I_SC}|) \rceil$$

Equation GFLIB_ControllerPIpAW_F32_Eq4

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 31st power of two (corresponds to the Q1.31 encoding):

$$f32PropGain = \left(K_{P_SC} \cdot 2^{-s16PropGainShift}\right) \cdot 2^{31}$$
$$f32IntegGain = \left(K_{I_SC} \cdot 2^{-s16IntegGainShift}\right) \cdot 2^{31}$$

Equation GFLIB_ControllerPIpAW_F32_Eq5

where

- f16PropGain - is the scaled value of proportional gain
- s16PropGainShift - is the scaling shift for proportional gain [-31, 31)
- f16IntegGain - is the scaled value of integral gain
- s16IntegGainShift - is the scaling shift for integral gain [-31, 31)

The sum of the scaled proportional and integral terms gives a complete equation of the controller:

$$u_f(k) = e_f(k) \cdot K_{P_SC} + u_{If}(k-1) + K_{I_SC} \cdot e_f(k) + K_{I_SC} \cdot e_f(k-1)$$

Equation GFLIB_ControllerPIpAW_F32_Eq6

The output signal limitation is implemented in this controller. The actual output u(k) is bounded not to exceed the given limit values f32UpperLimit, f32LowerLimit. This is due to either the bounded power of the actuator or to the physical constraints of the plant.

$$u(k) = \begin{cases} f32UpperLimit & if \quad u_f(k) \geq f32UpperLimit \\ u_f(k) & if \quad f32LoweLimit < u_f(k) < f32UpperLimit \\ f32LowerLimit & if \quad u_f(k) \leq f32LoweLimit \end{cases}$$

Equation GFLIB_ControllerPIpAW_F32_Eq7

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The bounds are described by a limitation element equation GFLIB_ControllerPIpAW_F32_Eq7. When the bounds are exceeded the non-linear saturation characteristic will take effect and influence the dynamic behavior. The described limitation is implemented on the integral part accumulator (limitation during the calculation) and on the overall controller output. Therefore, if the limitation occurs, the controller output is limited to its bounds and the wind-up occurrence of the accumulator portion is avoided by saturating the actual sum.

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32 macro.

2.24.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PIAW_P_T_F32 trMyPI = GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32;

void main(void)
{
    tFrac32 f32ControllerPIpAWOut;

    // Set the input error
    f32InErr = FRAC32(0.25);

    // Initialize the controller parameters
    trMyPI.f32PropGain        = FRAC32(0.01);
    trMyPI.f32IntegGain       = FRAC32(0.02);
    trMyPI.s16PropGainShift   = 1;
    trMyPI.s16IntegGainShift  = 1;
    trMyPI.f32UpperLimit      = FRAC32(1.0);
    trMyPI.f32LowerLimit      = FRAC32(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWInit_F32(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWInit(&trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpAWInit(&trMyPI);


    // Initialize the state variables to predefined values
    f32ControllerPIpAWOut = FRAC32(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWSetState_F32(f32ControllerPIpAWOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWSetState(f32ControllerPIpAWOut, &trMyPI, F32);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpAWSetState(f32ControllerPIpAWOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIpAW_F32(f32InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIpAW(f32InErr, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    f32Output = GFLIB_ControllerPIpAW(f32InErr, &trMyPI);
}
```

2.24.4 Function GFLIB_ControllerPIpAW_F16

2.24.4.1 Declaration

```
tFrac16 GFLIB_ControllerPIpAW_F16(tFrac16 f16InErr,
GFLIB_CONTROLLER_PIAW_P_T_F16 *const pParam);
```

2.24.4.2 Arguments

Table 176. GFLIB_ControllerPIpAW_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InErr	**input**	Input error signal to the controller is a 16-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PIAW_P_T_F16 *const	pParam	**input, output**	Pointer to the controller parameters structure.

2.24.4.3 Return

The function returns a 16-bit value in format Q1.15, representing the signal to be applied to the controlled system so that the input error is forced to zero.

2.24.4.4 Implementation details

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. The input and output of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The fractional representation of both input and output signals, normalized between [-1, 1), is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$
$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation GFLIB_ControllerPIpAW_F16_Eq1

Applying such scaling (normalization) on the proportional term of equation GFLIB_ControllerPIpAW_eq3 results in:

$$u_{Pf}(k) = e_f(k) \cdot K_{P_SC} \quad where \quad K_{P_SC} = K_P \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIpAW_F16_Eq2

where K_{P_sc} is the proportional gain parameter considering input/output scaling.

Similarly, scaling the integral term of equation GFLIB_ControllerPIpAW_eq4 results in:

$$u_{If}(k) = u_{If}(k-1) + K_{I_SC} \cdot e_f(k) + K_{I_SC} \cdot e_f(k-1) \quad where \quad K_{I_SC} = \frac{K_I T_s}{2} \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIpAW_F16_Eq3

where K_{I_sc} is the integral gain parameter considering input/output scaling. The sum of the scaled proportional and integral terms gives a complete equation of the controller.

In case the gain parameters K_{P_sc} or K_{I_sc} exceed the fractional range [-1, 1) or they are extremely small (which could lead to loss of precision), their value must be bit-shifted by *s16PropGainShift* and *s16IntegGainShift*, respectively. The shift amount can be calculated as

$$s16PropGainShift = \lceil \log_2(|K_{P_SC}|) \rceil$$
$$s16IntegGainShift = \lceil \log_2(|K_{I_SC}|) \rceil$$

Equation GFLIB_ControllerPIpAW_F16_Eq4

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 15th power of two (corresponds to the Q1.15 encoding):

$$f16PropGain = \left(K_{P_{SC}} \cdot 2^{-s16PropGainShift}\right) \cdot 2^{15}$$
$$f16IntegGain = \left(K_{I_{SC}} \cdot 2^{-s16IntegGainShift}\right) \cdot 2^{15}$$

Equation GFLIB_ControllerPIpAW_F16_Eq5

where

- f16PropGain - is the scaled value of proportional gain
- s16PropGainShift - is the scaling shift for proportional gain [-31, 31)
- f16IntegGain - is the scaled value of integral gain

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

- s16IntegGainShift - is the scaling shift for integral gain [-31, 31)

The sum of the scaled proportional and integral terms gives a complete equation of the controller:

$$u_f(k) = e_f(k) \cdot K_{P_SC} + u_{If}(k-1) + K_{I_SC} \cdot e_f(k) + K_{I_SC} \cdot e_f(k-1)$$

Equation GFLIB_ControllerPIpAW_F16_Eq6

The output signal limitation is implemented in this controller. The actual output u(k) is bounded not to exceed the given limit values f16UpperLimit, f16LowerLimit. This is due to either the bounded power of the actuator or to the physical constraints of the plant.

$$u(k) = \begin{cases} f16UpperLimit & if \quad u_f(k) \geq f16UpperLimit \\ u_f(k) & if \quad f16LoweLimit < u_f(k) < f16UpperLimit \\ f16LowerLimit & if \quad u_f(k) \leq f16LoweLimit \end{cases}$$

Equation GFLIB_ControllerPIpAW_F16_Eq7

The bounds are described by a limitation element equation GFLIB_ControllerPIpAW_F16_Eq7. When the bounds are exceeded the non-linear saturation characteristic will take effect and influence the dynamic behavior. The described limitation is implemented on the integral part accumulator (limitation during the calculation) and on the overall controller output. Therefore, if the limitation occurs, the controller output is limited to its bounds and the wind-up occurrence of the accumulator portion is avoided by saturating the actual sum.

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16 macro.

2.24.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PIAW_P_T_F16 trMyPI = GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16;

void main(void)
{
    tFrac16 f16ControllerPIpAWOut;

    // Set the input error
    f16InErr = FRAC16(0.25);

    // Initialize the controller parameters
    trMyPI.f16PropGain         = FRAC16(0.01);
    trMyPI.f16IntegGain        = FRAC16(0.02);
    trMyPI.s16PropGainShift    = 1;
    trMyPI.s16IntegGainShift   = 1;
    trMyPI.f16UpperLimit       = FRAC16(1.0);
    trMyPI.f16LowerLimit       = FRAC16(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    GFLIB_ControllerPIpAWInit_F16(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWInit(&trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpAWInit(&trMyPI);


    // Initialize the state variables to predefined values
    f16ControllerPIpAWOut = FRAC16(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWSetState_F16(f16ControllerPIpAWOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWSetState(f16ControllerPIpAWOut, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpAWSetState(f16ControllerPIpAWOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIpAW_F16(f16InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIpAW(f16InErr, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    f16Output = GFLIB_ControllerPIpAW(f16InErr, &trMyPI);
}
```

2.24.5 Function GFLIB_ControllerPIpAW_FLT

2.24.5.1 Declaration

```
tFloat GFLIB_ControllerPIpAW_FLT(tFloat fltInErr,
GFLIB_CONTROLLER_PIAW_P_T_FLT *const pParam);
```

2.24.5.2 Arguments

Table 177. GFLIB_ControllerPIpAW_FLT arguments

Type	Name	Direction	Description
tFloat	fltInErr	**input**	Input error signal to the controller is a single precision floating point data type.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 177. GFLIB_ControllerPIpAW_FLT arguments...*continued*

Type	Name	Direction	Description
GFLIB_CONTROLLER_PIAW_P_T_FLT *const	pParam	**input, output**	Pointer to the controller parameters structure.

2.24.5.3 Return

The function returns a single precision floating point value, representing the signal to be applied to the controlled system so that the input error is forced to zero.

2.24.5.4 Implementation details

The output signal limitation is implemented in this controller. The actual output u(k) is bounded not to exceed the given limit values fltUpperLimit, fltLowerLimit. This is due to either the bounded power of the actuator or to the physical constraints of the plant.

$$u(k) = \begin{cases} fltUpperLimit & if \quad u_f(k) \geq fltUpperLimit \\ u_f(k) & if \quad fltLoweLimit < u_f(k) < fltUpperLimit \\ fltLowerLimit & if \quad u_f(k) \leq fltLoweLimit \end{cases}$$

Equation GFLIB_ControllerPIpAW_FLT_Eq1

The sum of the scaled proportional and integral terms gives a complete equation of the controller:

$$u(k) = e_f(k) \cdot K_P + u_I(k-1) + e(k) \cdot \frac{K_I T_S}{2} + e(k-1) \cdot \frac{K_I T_S}{2}$$

Equation GFLIB_ControllerPIpAW_FLT_Eq2

The bounds are described by a limitation element equation GFLIB_ControllerPIpAW_FLT_Eq2. When the bounds are exceeded, the non-linear saturation characteristic will take effect and influence the dynamic behavior. The described limitation is implemented on the overall controller output. Therefore, if the limitation occurs, the controller output is limited to its bounds and the wind-up occurrence of the accumulator portion is avoided by saturating the actual sum.

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT macro.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.24.5.5 Code Example

```
#include "gflib.h"

tFloat fltInErr;
tFloat fltOutput;

GFLIB_CONTROLLER_PIAW_P_T_FLT trMyPI = GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT;

void main(void)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
{
    tFloat fltControllerPIpAWOut;

    // Set the input error
    fltInErr = 0.25F;

    // Initialize the controller parameters
    trMyPI.fltPropGain   =  0.04F;
    trMyPI.fltIntegGain  =  0.02F;
    trMyPI.fltUpperLimit =  1.0F;
    trMyPI.fltLowerLimit = -1.0F;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWInit_FLT(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWInit(&trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIpAWInit(&trMyPI);


    // Initialize the state variables to predefined values
    fltControllerPIpAWOut = 0.03F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWSetState_FLT(fltControllerPIpAWOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWSetState(fltControllerPIpAWOut, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIpAWSetState(fltControllerPIpAWOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIpAW_FLT(fltInErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIpAW(fltInErr, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    fltOutput = GFLIB_ControllerPIpAW(fltInErr, &trMyPI);
}
```

2.24.6 Function GFLIB_ControllerPIpAWInit

2.24.6.1 Description

This function clears the GFLIB_ControllerPIpAW state variables.

Note: *The input/output pointer must contain a valid address, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.24.6.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.24.6.3 Function GFLIB_ControllerPIpAWInit_F32

2.24.6.3.1 Declaration

```
void GFLIB_ControllerPIpAWInit_F32(GFLIB_CONTROLLER_PIAW_P_T_F32
*const pParam);
```

2.24.6.3.2 Arguments

Table 178. GFLIB_ControllerPIpAWInit_F32 arguments

Type	Name	Direction	Description
GFLIB_CONTROLLER_PIAW_P_T_F32 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIpAW state.

2.24.6.3.3 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PIAW_P_T_F32 trMyPI = GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32;

void main(void)
{
    tFrac32 f32ControllerPIpAWOut;

    // Set the input error
    f32InErr = FRAC32(0.25);

    // Initialize the controller parameters
    trMyPI.f32PropGain        = FRAC32(0.01);
    trMyPI.f32IntegGain       = FRAC32(0.02);
    trMyPI.s16PropGainShift   = 1;
    trMyPI.s16IntegGainShift  = 1;
    trMyPI.f32UpperLimit      = FRAC32(1.0);
    trMyPI.f32LowerLimit      = FRAC32(-1.0);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWInit_F32(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWInit(&trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpAWInit(&trMyPI);


    // Initialize the state variables to predefined values
    f32ControllerPIpAWOut = FRAC32(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWSetState_F32(f32ControllerPIpAWOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWSetState(f32ControllerPIpAWOut, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpAWSetState(f32ControllerPIpAWOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIpAW_F32(f32InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIpAW(f32InErr, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    f32Output = GFLIB_ControllerPIpAW(f32InErr, &trMyPI);
}
```

2.24.6.4 Function GFLIB_ControllerPIpAWInit_F16

2.24.6.4.1 Declaration

```
void GFLIB_ControllerPIpAWInit_F16(GFLIB_CONTROLLER_PIAW_P_T_F16
*const pParam);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.24.6.4.2 Arguments

Table 179. GFLIB_ControllerPIpAWInit_F16 arguments

Type	Name	Direction	Description
GFLIB_CONTROLLER_PIAW_P_T_F16 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIpAW state.

2.24.6.4.3 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PIAW_P_T_F16 trMyPI = GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16;

void main(void)
{
    tFrac16 f16ControllerPIpAWOut;

    // Set the input error
    f16InErr = FRAC16(0.25);

    // Initialize the controller parameters
    trMyPI.f16PropGain        = FRAC16(0.01);
    trMyPI.f16IntegGain       = FRAC16(0.02);
    trMyPI.s16PropGainShift   = 1;
    trMyPI.s16IntegGainShift  = 1;
    trMyPI.f16UpperLimit      = FRAC16(1.0);
    trMyPI.f16LowerLimit      = FRAC16(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWInit_F16(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWInit(&trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpAWInit(&trMyPI);


    // Initialize the state variables to predefined values
    f16ControllerPIpAWOut = FRAC16(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWSetState_F16(f16ControllerPIpAWOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWSetState(f16ControllerPIpAWOut, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpAWSetState(f16ControllerPIpAWOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIpAW_F16(f16InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIpAW(f16InErr, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    f16Output = GFLIB_ControllerPIpAW(f16InErr, &trMyPI);
}
```

2.24.6.5 Function GFLIB_ControllerPIpAWInit_FLT

2.24.6.5.1 Declaration

```
void GFLIB_ControllerPIpAWInit_FLT(GFLIB_CONTROLLER_PIAW_P_T_FLT
*const pParam);
```

2.24.6.5.2 Arguments

Table 180. GFLIB_ControllerPIpAWInit_FLT arguments

Type	Name	Direction	Description
GFLIB_CONTROLLER_PIAW_P_T_FLT *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIpAW state.

2.24.6.5.3 Code Example

```
#include "gflib.h"

tFloat fltInErr;
tFloat fltOutput;

GFLIB_CONTROLLER_PIAW_P_T_FLT trMyPI = GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT;

void main(void)
{
    tFloat fltControllerPIpAWOut;

    // Set the input error
    fltInErr = 0.25F;

    // Initialize the controller parameters
    trMyPI.fltPropGain    =  0.04F;
    trMyPI.fltIntegGain   =  0.02F;
    trMyPI.fltUpperLimit  =  1.0F;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    trMyPI.fltLowerLimit = -1.0F;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWInit_FLT(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWInit(&trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIpAWInit(&trMyPI);


    // Initialize the state variables to predefined values
    fltControllerPIpAWOut = 0.03F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWSetState_FLT(fltControllerPIpAWOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWSetState(fltControllerPIpAWOut, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIpAWSetState(fltControllerPIpAWOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIpAW_FLT(fltInErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIpAW(fltInErr, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    fltOutput = GFLIB_ControllerPIpAW(fltInErr, &trMyPI);
}
```

2.24.7 Function GFLIB_ControllerPIpAWSetState

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.24.7.1 Description

This function initializes the GFLIB_ControllerPIpAW state variables to achieve the required output values.

Note: *The input/output pointer must contain a valid address, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.24.7.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.24.7.3 Function GFLIB_ControllerPIpAWSetState_F32

2.24.7.3.1 Declaration

```
void GFLIB_ControllerPIpAWSetState_F32(tFrac32
f32ControllerPIpAWOut, GFLIB_CONTROLLER_PIAW_P_T_F32 *const
pParam);
```

2.24.7.3.2 Arguments

Table 181. GFLIB_ControllerPIpAWSetState_F32 arguments

Type	Name	Direction	Description
tFrac32	f32ControllerPIpAWOut	**Output**	Required output of the GFLIB_ControllerPIpAW.
GFLIB_CONTROLLER_PIAW_P_T_F32 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIpAW state.

2.24.7.3.3 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PIAW_P_T_F32 trMyPI = GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32;

void main(void)
{
    tFrac32 f32ControllerPIpAWOut;

    // Set the input error
    f32InErr = FRAC32(0.25);

    // Initialize the controller parameters
    trMyPI.f32PropGain         = FRAC32(0.01);
    trMyPI.f32IntegGain        = FRAC32(0.02);
    trMyPI.s16PropGainShift    = 1;
    trMyPI.s16IntegGainShift   = 1;
    trMyPI.f32UpperLimit       = FRAC32(1.0);
    trMyPI.f32LowerLimit       = FRAC32(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWInit_F32(&trMyPI);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWInit(&trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpAWInit(&trMyPI);


    // Initialize the state variables to predefined values
    f32ControllerPIpAWOut = FRAC32(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWSetState_F32(f32ControllerPIpAWOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWSetState(f32ControllerPIpAWOut, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpAWSetState(f32ControllerPIpAWOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIpAW_F32(f32InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIpAW(f32InErr, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    f32Output = GFLIB_ControllerPIpAW(f32InErr, &trMyPI);
}
```

2.24.7.4 Function GFLIB_ControllerPIpAWSetState_F16

2.24.7.4.1 Declaration

```
void GFLIB_ControllerPIpAWSetState_F16(tFrac16
f16ControllerPIpAWOut, GFLIB_CONTROLLER_PIAW_P_T_F16 *const
pParam);
```

2.24.7.4.2 Arguments

Table 182. GFLIB_ControllerPIpAWSetState_F16 arguments

Type	Name	Direction	Description
tFrac16	f16ControllerPIpAWOut	Input	Required output of the GFLIB_ControllerPIpAW.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 182. GFLIB_ControllerPIpAWSetState_F16 arguments...*continued*

Type	Name	Direction	Description
GFLIB_CONTROLLER_PIAW_P_T_F16 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIpAW state.

2.24.7.4.3 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PIAW_P_T_F16 trMyPI = GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16;

void main(void)
{
    tFrac16 f16ControllerPIpAWOut;

    // Set the input error
    f16InErr = FRAC16(0.25);

    // Initialize the controller parameters
    trMyPI.f16PropGain        = FRAC16(0.01);
    trMyPI.f16IntegGain       = FRAC16(0.02);
    trMyPI.s16PropGainShift   = 1;
    trMyPI.s16IntegGainShift  = 1;
    trMyPI.f16UpperLimit      = FRAC16(1.0);
    trMyPI.f16LowerLimit      = FRAC16(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWInit_F16(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWInit(&trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIpAWInit(&trMyPI);


    // Initialize the state variables to predefined values
    f16ControllerPIpAWOut = FRAC16(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWSetState_F16(f16ControllerPIpAWOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWSetState(f16ControllerPIpAWOut, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    GFLIB_ControllerPIpAWSetState(f16ControllerPIpAWOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIpAW_F16(f16InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIpAW(f16InErr, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    f16Output = GFLIB_ControllerPIpAW(f16InErr, &trMyPI);
}
```

2.24.7.5 Function GFLIB_ControllerPIpAWSetState_FLT

2.24.7.5.1 Declaration

```
void GFLIB_ControllerPIpAWSetState_FLT(tFloat
fltControllerPIpAWOut, GFLIB_CONTROLLER_PIAW_P_T_FLT *const
pParam);
```

2.24.7.5.2 Arguments

Table 183. GFLIB_ControllerPIpAWSetState_FLT arguments

Type	Name	Direction	Description
tFloat	fltControllerPIpAWOut	**input**	Required output of the GFLIB_ControllerPIpAW.
GFLIB_CONTROLLER_PIAW_P_T_FLT *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIpAW state.

2.24.7.5.3 Code Example

```
#include "gflib.h"

tFloat fltInErr;
tFloat fltOutput;

GFLIB_CONTROLLER_PIAW_P_T_FLT trMyPI = GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT;

void main(void)
{
    tFloat fltControllerPIpAWOut;

    // Set the input error
    fltInErr = 0.25F;

    // Initialize the controller parameters
    trMyPI.fltPropGain    =  0.04F;
    trMyPI.fltIntegGain   =  0.02F;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    trMyPI.fltUpperLimit =  1.0F;
    trMyPI.fltLowerLimit = -1.0F;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWInit_FLT(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWInit(&trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIpAWInit(&trMyPI);


    // Initialize the state variables to predefined values
    fltControllerPIpAWOut = 0.03F;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWSetState_FLT(fltControllerPIpAWOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIpAWSetState(fltControllerPIpAWOut, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIpAWSetState(fltControllerPIpAWOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIpAW_FLT(fltInErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIpAW(fltInErr, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    fltOutput = GFLIB_ControllerPIpAW(fltInErr, &trMyPI);
}
```

2.25 Function GFLIB_ControllerPIr

This function calculates a standard recurrent form of the Proportional-Integral controller, without integral anti-windup.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.25.1 Description

The function GFLIB_ControllerPIr calculates a standard recurrent form of the Proportional-Integral controller, without integral anti-windup.

The continuous time domain representation of the PI controller is defined as:

$$u(t) = e(t) \cdot K_P + K_I \int_0^t e(t)\,dt$$

Equation GFLIB_ControllerPIr_Eq1

The transfer function for this kind of PI controller, in a continuous time domain, is described using the Laplace transform as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{s \cdot K_P + K_I}{s}$$

Equation GFLIB_ControllerPIr_Eq2

Transforming equation GFLIB_ControllerPIr_Eq2 into a discrete time domain leads to the following equation:

$$u(k) = u(k-1) + e(k) \cdot CC1 + e(k-1) \cdot CC2$$

Equation GFLIB_ControllerPIr_Eq3

where Kp is proportional gain, Ki is integral gain, Ts is the sampling period, u(k) is the controller output, e(k) is the controller input error signal, CC1 and CC2 are controller coefficients calculated depending on the discretization method used, as shown in Table 184 .

Table 184. Calculation of coefficients CC1 and CC2 using various discretization methods

	Trapezoidal (bilinear)	Backward rectangular	Forward rectangular
CC1=	$K_p+K_IT_S/2$	$K_p+K_IT_S$	K_p
CC2=	$-K_p+K_IT_S/2$	$-K_p$	$-K_p+K_IT_S$

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.25.2 Re-entrancy

The function is re-entrant for a different `pParam`.

2.25.3 Function GFLIB_ControllerPIr_F32

2.25.3.1 Declaration

```
tFrac32 GFLIB_ControllerPIr_F32(tFrac32 f32InErr,
GFLIB_CONTROLLER_PI_R_T_F32 *const pParam);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.25.3.2 Arguments

Table 185. GFLIB_ControllerPIr_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InErr	**input**	Input error signal to the controller is a 32-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PI_R_T_F32 *const	pParam	**input, output**	Pointer to the controller parameters structure.

2.25.3.3 Return

The function returns a 32-bit value in the fractional format Q1.31, representing the signal to be applied to the controlled system so that the input error is forced to zero.

2.25.3.4 Implementation details

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. The input and output of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

Then the fractional representation [-1, 1) of both input and output signals is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation GFLIB_ControllerPIr_F32_Eq1

The resulting controller discrete time domain equation in fixed point fractional representation is therefore given as:

$$u_f(k) \cdot U^{MAX} = u_f(k-1) \cdot U^{MAX} + e_f(k) \cdot E^{MAX} \cdot CC1 + e_f(k-1) \cdot E^{MAX} \cdot CC2$$

Equation GFLIB_ControllerPIr_F32_Eq2

which can be rearranged into the following form:

$$u_f(k) = u_f(k-1) + e_f(k) \cdot CC1_{SC} + e_f(k-1) \cdot CC2_{SC}$$

Equation GFLIB_ControllerPIr_F32_Eq3

where

$$CC1_{SC} = CC1 \cdot \frac{E^{MAX}}{U^{MAX}}$$

$$CC2_{SC} = CC2 \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIr_F32_Eq4

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

are the scaled controller coefficients. In case the coefficients $CC1_{SC}$ or $CC2_{SC}$ exceed the fractional range [-1, 1), their value must be right-shifted by *u16NShift*. The binary shift amount corresponds to

$$u16Nshift = \max\left(0, \left\lceil \log_2\left(\max\left(\left|CC1_{SC}\right|, \left|CC2_{SC}\right|\right)\right)\right\rceil\right)$$

Equation GFLIB_ControllerPIr_F32_Eq5

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 31st power of two (corresponds to the Q1.31 encoding):

$$f32CC1sc = \left(CC1_{SC} \cdot 2^{-u16NShift}\right) \cdot 2^{31}$$
$$f32CC2sc = \left(CC2_{SC} \cdot 2^{-u16NShift}\right) \cdot 2^{31}$$

Equation GFLIB_ControllerPIr_F32_Eq6

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PI_R_DEFAULT_F32 macro.

2.25.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PI_R_T_F32 trMyPI = GFLIB_CONTROLLER_PI_R_DEFAULT_F32;

void main(void)
{
    tFrac32 f32ControllerPIrOut;

    // Set the input error
    f32InErr = FRAC32(0.25);

    // Initialize the controller parameters
    trMyPI.f32CC1sc  = FRAC32(0.01);
    trMyPI.f32CC2sc  = FRAC32(0.02);
    trMyPI.u16NShift = 1;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrInit_F32(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrInit(&trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrInit(&trMyPI);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Initialize the state variables to predefined values
    f32ControllerPIrOut = FRAC32(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrSetState_F32(f32ControllerPIrOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrSetState(f32ControllerPIrOut, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrSetState(f32ControllerPIrOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIr_F32(f32InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIr(f32InErr, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    f32Output = GFLIB_ControllerPIr(f32InErr, &trMyPI);
}
```

2.25.4 Function GFLIB_ControllerPIr_F16

2.25.4.1 Declaration

```
tFrac16 GFLIB_ControllerPIr_F16(tFrac16 f16InErr,
GFLIB_CONTROLLER_PI_R_T_F16 *const pParam);
```

2.25.4.2 Arguments

Table 186. GFLIB_ControllerPIr_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InErr	**input**	Input error signal to the controller is a 16-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PI_R_T_F16 *const	pParam	**input, output**	Pointer to the controller parameters structure.

2.25.4.3 Return

The function returns a 16-bit value in the fractional format Q1.15, representing the signal to be applied to the controlled system so that the input error is forced to zero.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.25.4.4 Implementation details

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. The input and output of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

Then the fractional representation [-1, 1) of both input and output signals is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$
$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation GFLIB_ControllerPIr_F16_Eq1

The resulting controller discrete time domain equation in fixed point fractional representation is therefore given as:

$$u_f(k) \cdot U^{MAX} = u_f(k-1) \cdot U^{MAX} + e_f(k) \cdot E^{MAX} \cdot CC1 + e_f(k-1) \cdot E^{MAX} \cdot CC2$$

Equation GFLIB_ControllerPIr_F16_Eq2

which can be rearranged into the following form:

$$u_f(k) = u_f(k-1) + e_f(k) \cdot CC1_{SC} + e_f(k-1) \cdot CC2_{SC}$$

Equation GFLIB_ControllerPIr_F16_Eq3

where

$$CC1_{SC} = CC1 \cdot \frac{E^{MAX}}{U^{MAX}}$$
$$CC2_{SC} = CC2 \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIr_F16_Eq4

are the scaled controller coefficients. In case the coefficients $CC1_{SC}$ or $CC2_{SC}$ exceed the fractional range [-1, 1), their value must be right-shifted by *u16NShift*. The binary shift amount corresponds to

$$u16Nshift = \max\left(0, \left\lceil \log_2\left(\max\left(\left|CC1_{SC}\right|, \left|CC2_{SC}\right|\right)\right)\right\rceil\right)$$

Equation GFLIB_ControllerPIr_F16_Eq5

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 15th power of two (corresponds to the Q1.15 encoding):

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$f16CC1sc = \left(CC1_{SC} \cdot 2^{-u16NShift}\right) \cdot 2^{15}$$
$$f16CC2sc = \left(CC2_{SC} \cdot 2^{-u16NShift}\right) \cdot 2^{15}$$

Equation GFLIB_ControllerPIr_F16_Eq6

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PI_R_DEFAULT_F16 macro.

2.25.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PI_R_T_F16 trMyPI = GFLIB_CONTROLLER_PI_R_DEFAULT_F16;

void main(void)
{
    tFrac16 f16ControllerPIrOut;

    // Set the input error
    f16InErr = FRAC16(0.25);

    // Initialize the controller parameters
    trMyPI.f16CC1sc  = FRAC16(0.01);
    trMyPI.f16CC2sc  = FRAC16(0.02);
    trMyPI.u16NShift = 1;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrInit_F16(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrInit(&trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrInit(&trMyPI);


    // Initialize the state variables to predefined values
    f16ControllerPIrOut = FRAC16(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrSetState_F16(f16ControllerPIrOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrSetState(f16ControllerPIrOut, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrSetState(f16ControllerPIrOut, &trMyPI);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIr_F16(f16InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIr(f16InErr, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    f16Output = GFLIB_ControllerPIr(f16InErr, &trMyPI);
}
```

2.25.5 Function GFLIB_ControllerPIr_FLT

2.25.5.1 Declaration

```
tFloat GFLIB_ControllerPIr_FLT(tFloat fltInErr,
GFLIB_CONTROLLER_PI_R_T_FLT *const pParam);
```

2.25.5.2 Arguments

Table 187. GFLIB_ControllerPIr_FLT arguments

Type	Name	Direction	Description
tFloat	fltInErr	**input**	Input error signal to the controller as a single precision floating point value.
GFLIB_CONTROLLER_PI_R_T_FLT *const	pParam	**input, output**	Pointer to the controller parameters structure.

2.25.5.3 Return

The function returns a single precision floating point value, representing the signal to be applied to the controlled system so that the input error is forced to zero.

2.25.5.4 Implementation details

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PI_R_DEFAULT_FLT macro.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.25.5.5 Code Example

```
#include "gflib.h"

tFloat fltInErr;
tFloat fltOutput;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
GFLIB_CONTROLLER_PI_R_T_FLT trMyPI = GFLIB_CONTROLLER_PI_R_DEFAULT_FLT;

void main(void)
{
    tFloat fltControllerPIrOut;

    // Set the input error
    fltInErr = 0.25f;

    // Initialize the controller parameters
    trMyPI.fltCC1sc  = 0.01f;
    trMyPI.fltCC2sc  = 0.02f;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrInit_FLT(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrInit(&trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIrInit(&trMyPI);


    // Initialize the state variables to predefined values
    fltControllerPIrOut = 0.03f;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrSetState_FLT(fltControllerPIrOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrSetState(fltControllerPIrOut, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIrSetState(fltControllerPIrOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIr_FLT(fltInErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIr(fltInErr, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // only if single precision floating-point implementation is selected
    // as default.
    fltOutput = GFLIB_ControllerPIr(fltInErr, &trMyPI);
}
```

2.25.6 Function GFLIB_ControllerPIrInit

2.25.6.1 Description

This function clears the GFLIB_ControllerPIr state variables.

Note: *The input/output pointer must contain a valid address, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.25.6.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.25.6.3 Function GFLIB_ControllerPIrInit_F32

2.25.6.3.1 Declaration

```
void GFLIB_ControllerPIrInit_F32(GFLIB_CONTROLLER_PI_R_T_F32
*const pParam);
```

2.25.6.3.2 Arguments

Table 188. GFLIB_ControllerPIrInit_F32 arguments

Type	Name	Direction	Description
GFLIB_CONTROLLER_PI_R_T_F32 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIr state.

2.25.6.3.3 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PI_R_T_F32 trMyPI = GFLIB_CONTROLLER_PI_R_DEFAULT_F32;

void main(void)
{
    tFrac32 f32ControllerPIrOut;

    // Set the input error
    f32InErr = FRAC32(0.25);

    // Initialize the controller parameters
    trMyPI.f32CC1sc  = FRAC32(0.01);
    trMyPI.f32CC2sc  = FRAC32(0.02);
    trMyPI.u16NShift = 1;

    // Clear the state variables
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrInit_F32(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrInit(&trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrInit(&trMyPI);


    // Initialize the state variables to predefined values
    f32ControllerPIrOut = FRAC32(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrSetState_F32(f32ControllerPIrOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrSetState(f32ControllerPIrOut, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrSetState(f32ControllerPIrOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIr_F32(f32InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIr(f32InErr, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    f32Output = GFLIB_ControllerPIr(f32InErr, &trMyPI);
}
```

2.25.6.4 Function GFLIB_ControllerPIrInit_F16

2.25.6.4.1 Declaration

```
void GFLIB_ControllerPIrInit_F16(GFLIB_CONTROLLER_PI_R_T_F16
*const pParam);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.25.6.4.2 Arguments

Table 189. GFLIB_ControllerPIrInit_F16 arguments

Type	Name	Direction	Description
GFLIB_CONTROLLER_PI_R_T_F16 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIr state.

2.25.6.4.3 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PI_R_T_F16 trMyPI = GFLIB_CONTROLLER_PI_R_DEFAULT_F16;

void main(void)
{
    tFrac16 f16ControllerPIrOut;

    // Set the input error
    f16InErr = FRAC16(0.25);

    // Initialize the controller parameters
    trMyPI.f16CC1sc  = FRAC16(0.01);
    trMyPI.f16CC2sc  = FRAC16(0.02);
    trMyPI.u16NShift = 1;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrInit_F16(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrInit(&trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrInit(&trMyPI);


    // Initialize the state variables to predefined values
    f16ControllerPIrOut = FRAC16(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrSetState_F16(f16ControllerPIrOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrSetState(f16ControllerPIrOut, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrSetState(f16ControllerPIrOut, &trMyPI);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIr_F16(f16InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIr(f16InErr, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    f16Output = GFLIB_ControllerPIr(f16InErr, &trMyPI);
}
```

2.25.6.5 Function GFLIB_ControllerPIrInit_FLT

2.25.6.5.1 Declaration

```
void GFLIB_ControllerPIrInit_FLT(GFLIB_CONTROLLER_PI_R_T_FLT
*const pParam);
```

2.25.6.5.2 Arguments

Table 190. GFLIB_ControllerPIrInit_FLT arguments

Type	Name	Direction	Description
GFLIB_CONTROLLER_PI_R_T_FLT *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIr state.

2.25.6.5.3 Code Example

```
#include "gflib.h"

tFloat fltInErr;
tFloat fltOutput;

GFLIB_CONTROLLER_PI_R_T_FLT trMyPI = GFLIB_CONTROLLER_PI_R_DEFAULT_FLT;

void main(void)
{
    tFloat fltControllerPIrOut;

    // Set the input error
    fltInErr = 0.25f;

    // Initialize the controller parameters
    trMyPI.fltCC1sc  = 0.01f;
    trMyPI.fltCC2sc  = 0.02f;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    GFLIB_ControllerPIrInit_FLT(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrInit(&trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIrInit(&trMyPI);


    // Initialize the state variables to predefined values
    fltControllerPIrOut = 0.03f;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrSetState_FLT(fltControllerPIrOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrSetState(fltControllerPIrOut, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIrSetState(fltControllerPIrOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIr_FLT(fltInErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIr(fltInErr, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    fltOutput = GFLIB_ControllerPIr(fltInErr, &trMyPI);
}
```

2.25.7 Function GFLIB_ControllerPIrSetState

2.25.7.1 Description

This function initializes the GFLIB_ControllerPIr state variables to achieve the required output values.

Note: *The input/output pointer must contain a valid address, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.25.7.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.25.7.3 Function GFLIB_ControllerPIrSetState_F32

2.25.7.3.1 Declaration

```
void GFLIB_ControllerPIrSetState_F32(tFrac32 f32ControllerPIrOut,
GFLIB_CONTROLLER_PI_R_T_F32 *const pParam);
```

2.25.7.3.2 Arguments

Table 191. GFLIB_ControllerPIrSetState_F32 arguments

Type	Name	Direction	Description
tFrac32	f32ControllerPIrOut	**input**	Required output of the GFLIB_ControllerPIr.
GFLIB_CONTROLLER_PI_R_T_F32 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIr state.

2.25.7.3.3 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PI_R_T_F32 trMyPI = GFLIB_CONTROLLER_PI_R_DEFAULT_F32;

void main(void)
{
    tFrac32 f32ControllerPIrOut;

    // Set the input error
    f32InErr = FRAC32(0.25);

    // Initialize the controller parameters
    trMyPI.f32CC1sc  = FRAC32(0.01);
    trMyPI.f32CC2sc  = FRAC32(0.02);
    trMyPI.u16NShift = 1;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrInit_F32(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrInit(&trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrInit(&trMyPI);


    // Initialize the state variables to predefined values
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    f32ControllerPIrOut = FRAC32(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrSetState_F32(f32ControllerPIrOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrSetState(f32ControllerPIrOut, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrSetState(f32ControllerPIrOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIr_F32(f32InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIr(f32InErr, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    f32Output = GFLIB_ControllerPIr(f32InErr, &trMyPI);
}
```

2.25.7.4 Function GFLIB_ControllerPIrSetState_F16

2.25.7.4.1 Declaration

```
void GFLIB_ControllerPIrSetState_F16(tFrac16 f16ControllerPIrOut,
GFLIB_CONTROLLER_PI_R_T_F16 *const pParam);
```

2.25.7.4.2 Arguments

Table 192. GFLIB_ControllerPIrSetState_F16 arguments

Type	Name	Direction	Description
tFrac16	f16ControllerPIrOut	**input**	Required output of the GFLIB_ControllerPIr.
GFLIB_CONTROLLER_PI_R_T_F16 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIr state.

2.25.7.4.3 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PI_R_T_F16 trMyPI = GFLIB_CONTROLLER_PI_R_DEFAULT_F16;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
void main(void)
{
    tFrac16 f16ControllerPIrOut;

    // Set the input error
    f16InErr = FRAC16(0.25);

    // Initialize the controller parameters
    trMyPI.f16CC1sc  = FRAC16(0.01);
    trMyPI.f16CC2sc  = FRAC16(0.02);
    trMyPI.u16NShift = 1;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrInit_F16(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrInit(&trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrInit(&trMyPI);


    // Initialize the state variables to predefined values
    f16ControllerPIrOut = FRAC16(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrSetState_F16(f16ControllerPIrOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrSetState(f16ControllerPIrOut, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrSetState(f16ControllerPIrOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIr_F16(f16InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIr(f16InErr, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    f16Output = GFLIB_ControllerPIr(f16InErr, &trMyPI);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.25.7.5 Function GFLIB_ControllerPIrSetState_FLT

2.25.7.5.1 Declaration

```
void GFLIB_ControllerPIrSetState_FLT(tFloat fltControllerPIrOut,
GFLIB_CONTROLLER_PI_R_T_FLT *const pParam);
```

2.25.7.5.2 Arguments

Table 193. GFLIB_ControllerPIrSetState_FLT arguments

Type	Name	Direction	Description
tFloat	fltControllerPIrOut	**input**	Required output of the GFLIB_ControllerPIr.
GFLIB_CONTROLLER_PI_R_T_FLT *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIr state.

2.25.7.5.3 Code Example

```
#include "gflib.h"

tFloat fltInErr;
tFloat fltOutput;

GFLIB_CONTROLLER_PI_R_T_FLT trMyPI = GFLIB_CONTROLLER_PI_R_DEFAULT_FLT;

void main(void)
{
    tFloat fltControllerPIrOut;

    // Set the input error
    fltInErr = 0.25f;

    // Initialize the controller parameters
    trMyPI.fltCC1sc  = 0.01f;
    trMyPI.fltCC2sc  = 0.02f;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrInit_FLT(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrInit(&trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIrInit(&trMyPI);


    // Initialize the state variables to predefined values
    fltControllerPIrOut = 0.03f;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrSetState_FLT(fltControllerPIrOut, &trMyPI);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrSetState(fltControllerPIrOut, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIrSetState(fltControllerPIrOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIr_FLT(fltInErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIr(fltInErr, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    fltOutput = GFLIB_ControllerPIr(fltInErr, &trMyPI);
}
```

2.26 Function GFLIB_ControllerPIrAW

This function calculates a standard recurrent form of the Proportional-Integral controller, with integral anti-windup.

2.26.1 Description

The function GFLIB_ControllerPIrAW calculates a standard recurrent form of the Proportional-Integral controller, with integral anti-windup.

The continuous time domain representation of the PI controller is defined as:

$$u(t) = e(t) \cdot K_P + K_I \int_0^t e(t)\,dt$$

Equation GFLIB_ControllerPIrAW_Eq1

The transfer function for this kind of PI controller, in a continuous time domain is described using the Laplace transform as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{s \cdot K_P + K_I}{s}$$

Equation GFLIB_ControllerPIrAW_Eq2

Transforming equation GFLIB_ControllerPIrAW_Eq2 into a discrete time domain leads to the following equation:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$u(k) = u(k-1) + e(k) \cdot CC1 + e(k-1) \cdot CC2$$

Equation GFLIB_ControllerPIrAW_Eq3

where Kp is proportional gain, Ki is integral gain, Ts is the sampling period, u(k) is the controller output, e(k) is the controller input error signal, CC1 and CC2 are the controller coefficients calculated depending on the discretization method used, as shown in Table 194 .

Table 194. Calculation of coefficients CC1 and CC2 using various discretization methods

	Trapezoidal (bilinear)	Backward rectangular	Forward rectangular
CC1=	$K_p+K_iT_s/2$	$K_p+K_iT_s$	K_p
CC2=	$-K_p+K_iT_s/2$	$-K_p$	$-K_p+K_iT_s$

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.26.2 Re-entrancy

The function is re-entrant for a different `pParam`.

2.26.3 Function GFLIB_ControllerPIrAW_F32

2.26.3.1 Declaration

```
tFrac32 GFLIB_ControllerPIrAW_F32(tFrac32 f32InErr,
GFLIB_CONTROLLER_PIAW_R_T_F32 *const pParam);
```

2.26.3.2 Arguments

Table 195. GFLIB_ControllerPIrAW_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InErr	**input**	Input error signal to the controller is a 32-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PIAW_R_T_F32 *const	pParam	**input, output**	Pointer to the controller parameters structure.

2.26.3.3 Return

The function returns a 32-bit value in the fractional format Q1.31, representing the signal to be applied to the controlled system so that the input error is forced to zero.

2.26.3.4 Implementation details

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. The input and output of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Then the fractional representation [-1, 1) of both input and output signals is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$
$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation GFLIB_ControllerPIrAW_F32_Eq1

The resulting controller discrete time domain equation in fixed point fractional representation is therefore given as:

$$u_f(k) \cdot U^{MAX} = u_f(k-1) \cdot U^{MAX} + e_f(k) \cdot E^{MAX} \cdot CC1 + e_f(k-1) \cdot E^{MAX} \cdot CC2$$

Equation GFLIB_ControllerPIrAW_F32_Eq2

which can be rearranged into the following form:

$$u_f(k) = u_f(k-1) + e_f(k) \cdot CC1_{SC} + e_f(k-1) \cdot CC2_{SC}$$

Equation GFLIB_ControllerPIrAW_F32_Eq3

where

$$CC1_{SC} = CC1 \cdot \frac{E^{MAX}}{U^{MAX}}$$
$$CC2_{SC} = CC2 \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIrAW_F32_Eq4

are the scaled controller coefficients. In case the coefficients $CC1_{SC}$ or $CC2_{SC}$ exceed the fractional range [-1, 1), their value must be right-shifted by *u16NShift*. The binary shift amount corresponds to

$$u16Nshift = \max\left(0, \left\lceil \log_2\left(\max\left(\left|CC1_{SC}\right|, \left|CC2_{SC}\right|\right)\right)\right\rceil\right)$$

Equation GFLIB_ControllerPIrAW_F32_Eq5

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 31st power of two (corresponds to the Q1.31 encoding):

$$f32CC1sc = \left(CC1_{SC} \cdot 2^{-u16NShift}\right) \cdot 2^{31}$$
$$f32CC2sc = \left(CC2_{SC} \cdot 2^{-u16NShift}\right) \cdot 2^{31}$$

Equation GFLIB_ControllerPIrAW_F32_Eq6

The output signal limitation is implemented in this controller. The actual output u(k) is bounded not to exceed the given limit values UpperLimit, LowerLimit. This is due to either the bounded power of the actuator or to the physical constraints of the plant.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$u(k) = \begin{cases} f32UpperLimit & if \quad u_f(k) \geq f32UpperLimit \\ u_f(k) & if \quad f32LoweLimit < u_f(k) < f32UpperLimit \\ f32LowerLimit & if \quad u_f(k) \leq f32LoweLimit \end{cases}$$

Equation GFLIB_ControllerPIrAW_F32_Eq7

The bounds are described by a limitation element equation eq13_GFLIB_ControllerPIrAW_F32. When the bounds are exceeded, the non-linear saturation characteristic will take effect and influence the dynamic behavior. The described limitation is implemented on the output sum. Therefore, if the limitation occurs, the controller output is limited to its bounds and the wind-up occurrence of the accumulator is avoided by saturating the output sum.

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32 macro.

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

2.26.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PIAW_R_T_F32 trMyPI = GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32;

void main(void)
{
    tFrac32 f32ControllerPIrAWOut;

    // Set the input error
    f32InErr = FRAC32(0.25);

    // Initialize the controller parameters
    trMyPI.f32CC1sc      = FRAC32(0.01);
    trMyPI.f32CC2sc      = FRAC32(0.02);
    trMyPI.u16NShift     = 1;
    trMyPI.f32UpperLimit = FRAC32(1.0);
    trMyPI.f32LowerLimit = FRAC32(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWInit_F32(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWInit(&trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrAWInit(&trMyPI);


    // Initialize the state variables to predefined values
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    f32ControllerPIrAWOut = FRAC32(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWSetState_F32(f32ControllerPIrAWOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWSetState(f32ControllerPIrAWOut, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrAWSetState(f32ControllerPIrAWOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIrAW_F32(f32InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIrAW(f32InErr, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    f32Output = GFLIB_ControllerPIrAW(f32InErr, &trMyPI);
}
```

2.26.4 Function GFLIB_ControllerPIrAW_F16

2.26.4.1 Declaration

```
tFrac16 GFLIB_ControllerPIrAW_F16(tFrac16 f16InErr,
GFLIB_CONTROLLER_PIAW_R_T_F16 *const pParam);
```

2.26.4.2 Arguments

Table 196. GFLIB_ControllerPIrAW_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InErr	**input**	Input error signal to the controller is a 16-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PIAW_R_T_F16 *const	pParam	**input, output**	Pointer to the controller parameters structure.

2.26.4.3 Return

The function returns a 16-bit value in fractional format 1.16, representing the signal to be applied to the controlled system so that the input error is forced to zero.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.26.4.4 Implementation details

Prior to calculating the controller coefficients, it is necessary to set the scaling constants. The input and output of the algorithm are limited to the fractional range [-1, 1). Incorrect setting of the scaling constants may lead to undesirable overflow or saturation during the calculation.

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

Then the fractional representation [-1, 1) of both input and output signals is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$
$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation GFLIB_ControllerPIrAW_F16_Eq1

The resulting controller discrete time domain equation in fixed point fractional representation is therefore given as:

$$u_f(k) \cdot U^{MAX} = u_f(k-1) \cdot U^{MAX} + e_f(k) \cdot E^{MAX} \cdot CC1 + e_f(k-1) \cdot E^{MAX} \cdot CC2$$

Equation GFLIB_ControllerPIrAW_F16_Eq2

which can be rearranged into the following form:

$$u_f(k) = u_f(k-1) + e_f(k) \cdot CC1_{SC} + e_f(k-1) \cdot CC2_{SC}$$

Equation GFLIB_ControllerPIrAW_F16_Eq3

where

$$CC1_{SC} = CC1 \cdot \frac{E^{MAX}}{U^{MAX}}$$
$$CC2_{SC} = CC2 \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_ControllerPIrAW_F16_Eq4

are the scaled controller coefficients. In case the coefficients $CC1_{SC}$ or $CC2_{SC}$ exceed the fractional range [-1, 1), their value must be right-shifted by *u16NShift*. The binary shift amount corresponds to

$$u16Nshift = \max\left(0, \left\lceil \log_2\left(\max\left(\left|CC1_{SC}\right|, \left|CC2_{SC}\right|\right)\right)\right\rceil\right)$$

Equation GFLIB_ControllerPIrAW_F16_Eq5

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficients fit in the fractional range and can be recorded into the parameters structure. To represent the coefficients in integer format expected by the C language, simply shift the radix point by multiplying the coefficient values by the 15th power of two (corresponds to the Q1.15 encoding):

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$f16CC1sc = \left(CC1_{SC} \cdot 2^{-u16NShift}\right) \cdot 2^{15}$$
$$f16CC2sc = \left(CC2_{SC} \cdot 2^{-u16NShift}\right) \cdot 2^{15}$$

Equation GFLIB_ControllerPIrAW_F16_Eq6

The output signal limitation is implemented in this controller. The actual output u(k) is bounded not to exceed the given limit values UpperLimit, LowerLimit. This is due to either the bounded power of the actuator or to the physical constraints of the plant.

$$u(k) = \begin{cases} f16UpperLimit & if \quad u_f(k) \geq f16UpperLimit \\ u_f(k) & if \quad f16LoweLimit < u_f(k) < f16UpperLimit \\ f16LowerLimit & if \quad u_f(k) \leq f16LoweLimit \end{cases}$$

Equation GFLIB_ControllerPIrAW_F16_Eq7

The bounds are described by a limitation element equation eq13_GFLIB_ControllerPIrAW_F16. When the bounds are exceeded, the non-linear saturation characteristic will take effect and influence the dynamic behavior. The described limitation is implemented on the output sum. Therefore, if the limitation occurs, the controller output is limited to its bounds and the wind-up occurrence of the accumulator is avoided by saturating the output sum.

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16 macro.

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

2.26.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PIAW_R_T_F16 trMyPI = GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16;

void main(void)
{
    tFrac16 f16ControllerPIrAWOut;

    // Set the input error
    f16InErr = FRAC16(0.25);

    // Initialize the controller parameters
    trMyPI.f16CC1sc      = FRAC16(0.01);
    trMyPI.f16CC2sc      = FRAC16(0.02);
    trMyPI.u16NShift     = 1;
    trMyPI.f16UpperLimit = FRAC16(1.0);
    trMyPI.f16LowerLimit = FRAC16(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWInit_F16(&trMyPI);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWInit(&trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrAWInit(&trMyPI);


    // Initialize the state variables to predefined values
    f16ControllerPIrAWOut = FRAC16(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWSetState_F16(f16ControllerPIrAWOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWSetState(f16ControllerPIrAWOut, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrAWSetState(f16ControllerPIrAWOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIrAW_F16(f16InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIrAW(f16InErr, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    f16Output = GFLIB_ControllerPIrAW(f16InErr, &trMyPI);
}
```

2.26.5 Function GFLIB_ControllerPIrAW_FLT

2.26.5.1 Declaration

```
tFloat GFLIB_ControllerPIrAW_FLT(tFloat fltInErr,
GFLIB_CONTROLLER_PIAW_R_T_FLT *const pParam);
```

2.26.5.2 Arguments

Table 197. GFLIB_ControllerPIrAW_FLT arguments

Type	Name	Direction	Description
tFloat	fltInErr	**input**	Input error signal to the controller in single precision floating point data format.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 197. GFLIB_ControllerPIrAW_FLT arguments...*continued*

Type	Name	Direction	Description
GFLIB_CONTROLLER_PIAW_R_T_FLT *const	pParam	**input, output**	Pointer to the controller parameters structure.

2.26.5.3 Return

The function returns a single precision floating point value, representing the signal to be applied to the controlled system so that the input error is forced to zero.

2.26.5.4 Implementation details

The output signal limitation is implemented in this controller. The actual output u(k) is bounded not to exceed the given limit values UpperLimit, LowerLimit. This is due to either the bounded power of the actuator or to the physical constraints of the plant.

$$u(k) = \begin{cases} fltUpperLimit & if \quad u_f(k) \geq fltUpperLimit \\ u_f(k) & if \quad fltLoweLimit < u_f(k) < fltUpperLimit \\ fltLowerLimit & if \quad u_f(k) \leq fltLoweLimit \end{cases}$$

Equation GFLIB_ControllerPIrAW_FLT_Eq1

The bounds are described by a limitation element equation eq4_GFLIB_ControllerPIrAW_FLT. When the bounds are exceeded, the non-linear saturation characteristic will take effect and influence the dynamic behavior. The described limitation is implemented on the output sum. Therefore, if the limitation occurs, the controller output is limited to its bounds and the wind-up occurrence of the accumulator is avoided by saturating the output sum.

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT macro.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.26.5.5 Code Example

```
#include "gflib.h"

tFloat fltInErr;
tFloat fltOutput;

GFLIB_CONTROLLER_PIAW_R_T_FLT trMyPI = GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT;

void main(void)
{
    tFloat fltControllerPIrAWOut;

    // Set the input error
    fltInErr = 0.25f;

    // Initialize the controller parameters
    trMyPI.fltCC1sc      = 0.01F;
    trMyPI.fltCC2sc      = 0.02F;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    trMyPI.fltUpperLimit = 1.0F;
    trMyPI.fltLowerLimit = -1.0F;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWInit_FLT(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWInit(&trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIrAWInit(&trMyPI);


    // Initialize the state variables to predefined values
    fltControllerPIrAWOut = 0.03f;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWSetState_FLT(fltControllerPIrAWOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWSetState(fltControllerPIrAWOut, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIrAWSetState(fltControllerPIrAWOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIrAW_FLT(fltInErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIrAW(fltInErr, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    fltOutput = GFLIB_ControllerPIrAW(fltInErr, &trMyPI);
}
```

2.26.6 Function GFLIB_ControllerPIrAWInit

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.26.6.1 Description

This function clears the GFLIB_ControllerPIrAW state variables.

Note: *The input/output pointer must contain a valid address, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.26.6.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

2.26.6.3 Function GFLIB_ControllerPIrAWInit_F32

2.26.6.3.1 Declaration

```
void GFLIB_ControllerPIrAWInit_F32(GFLIB_CONTROLLER_PIAW_R_T_F32
*const pParam);
```

2.26.6.3.2 Arguments

Table 198. GFLIB_ControllerPIrAWInit_F32 arguments

Type	Name	Direction	Description
GFLIB_CONTROLLER_PIAW_R_T_F32 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIrAW state.

2.26.6.3.3 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PIAW_R_T_F32 trMyPI = GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32;

void main(void)
{
    tFrac32 f32ControllerPIrAWOut;

    // Set the input error
    f32InErr = FRAC32(0.25);

    // Initialize the controller parameters
    trMyPI.f32CC1sc      = FRAC32(0.01);
    trMyPI.f32CC2sc      = FRAC32(0.02);
    trMyPI.u16NShift     = 1;
    trMyPI.f32UpperLimit = FRAC32(1.0);
    trMyPI.f32LowerLimit = FRAC32(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWInit_F32(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWInit(&trMyPI, F32);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrAWInit(&trMyPI);


    // Initialize the state variables to predefined values
    f32ControllerPIrAWOut = FRAC32(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWSetState_F32(f32ControllerPIrAWOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWSetState(f32ControllerPIrAWOut, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrAWSetState(f32ControllerPIrAWOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIrAW_F32(f32InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIrAW(f32InErr, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    f32Output = GFLIB_ControllerPIrAW(f32InErr, &trMyPI);
}
```

2.26.6.4 Function GFLIB_ControllerPIrAWInit_F16

2.26.6.4.1 Declaration

```
void GFLIB_ControllerPIrAWInit_F16(GFLIB_CONTROLLER_PIAW_R_T_F16
*const pParam);
```

2.26.6.4.2 Arguments

Table 199. GFLIB_ControllerPIrAWInit_F16 arguments

Type	Name	Direction	Description
GFLIB_CONTROLLER_PIAW_R_T_F16 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIrAW state.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.26.6.4.3 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PIAW_R_T_F16 trMyPI = GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16;

void main(void)
{
    tFrac16 f16ControllerPIrAWOut;

    // Set the input error
    f16InErr = FRAC16(0.25);

    // Initialize the controller parameters
    trMyPI.f16CC1sc      = FRAC16(0.01);
    trMyPI.f16CC2sc      = FRAC16(0.02);
    trMyPI.u16NShift     = 1;
    trMyPI.f16UpperLimit = FRAC16(1.0);
    trMyPI.f16LowerLimit = FRAC16(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWInit_F16(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWInit(&trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrAWInit(&trMyPI);


    // Initialize the state variables to predefined values
    f16ControllerPIrAWOut = FRAC16(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWSetState_F16(f16ControllerPIrAWOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWSetState(f16ControllerPIrAWOut, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrAWSetState(f16ControllerPIrAWOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIrAW_F16(f16InErr, &trMyPI);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIrAW(f16InErr, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    f16Output = GFLIB_ControllerPIrAW(f16InErr, &trMyPI);
}
```

2.26.6.5 Function GFLIB_ControllerPIrAWInit_FLT

2.26.6.5.1 Declaration

```
void GFLIB_ControllerPIrAWInit_FLT(GFLIB_CONTROLLER_PIAW_R_T_FLT
*const pParam);
```

2.26.6.5.2 Arguments

Table 200. GFLIB_ControllerPIrAWInit_FLT arguments

Type	Name	Direction	Description
GFLIB_CONTROLLER_PIAW_R_T_FLT *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIrAW state.

2.26.6.5.3 Code Example

```
#include "gflib.h"

tFloat fltInErr;
tFloat fltOutput;

GFLIB_CONTROLLER_PIAW_R_T_FLT trMyPI = GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT;

void main(void)
{
    tFloat fltControllerPIrAWOut;

    // Set the input error
    fltInErr = 0.25f;

    // Initialize the controller parameters
    trMyPI.fltCC1sc      = 0.01F;
    trMyPI.fltCC2sc      = 0.02F;
    trMyPI.fltUpperLimit = 1.0F;
    trMyPI.fltLowerLimit = -1.0F;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWInit_FLT(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWInit(&trMyPI, FLT);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIrAWInit(&trMyPI);


    // Initialize the state variables to predefined values
    fltControllerPIrAWOut = 0.03f;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWSetState_FLT(fltControllerPIrAWOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWSetState(fltControllerPIrAWOut, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIrAWSetState(fltControllerPIrAWOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIrAW_FLT(fltInErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIrAW(fltInErr, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    fltOutput = GFLIB_ControllerPIrAW(fltInErr, &trMyPI);
}
```

2.26.7 Function GFLIB_ControllerPIrAWSetState

2.26.7.1 Description

This function initializes the GFLIB_ControllerPIrAW state variables to achieve the required output values.

Note: *The input/output pointer must contain a valid address, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.26.7.2 Re-entrancy

The function is re-entrant for a different `pCtrl`.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.26.7.3 Function GFLIB_ControllerPIrAWSetState_F32

2.26.7.3.1 Declaration

```
void GFLIB_ControllerPIrAWSetState_F32(tFrac32
f32ControllerPIrAWOut, GFLIB_CONTROLLER_PIAW_R_T_F32 *const
pParam);
```

2.26.7.3.2 Arguments

Table 201. GFLIB_ControllerPIrAWSetState_F32 arguments

Type	Name	Direction	Description
tFrac32	f32ControllerPIrAWOut	**input**	Required output of the GFLIB_ControllerPIrAW.
GFLIB_CONTROLLER_PIAW_R_T_F32 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIrAW state.

2.26.7.3.3 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PIAW_R_T_F32 trMyPI = GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32;

void main(void)
{
    tFrac32 f32ControllerPIrAWOut;

    // Set the input error
    f32InErr = FRAC32(0.25);

    // Initialize the controller parameters
    trMyPI.f32CC1sc      = FRAC32(0.01);
    trMyPI.f32CC2sc      = FRAC32(0.02);
    trMyPI.u16NShift     = 1;
    trMyPI.f32UpperLimit = FRAC32(1.0);
    trMyPI.f32LowerLimit = FRAC32(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWInit_F32(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWInit(&trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrAWInit(&trMyPI);


    // Initialize the state variables to predefined values
    f32ControllerPIrAWOut = FRAC32(0.03);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWSetState_F32(f32ControllerPIrAWOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWSetState(f32ControllerPIrAWOut, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrAWSetState(f32ControllerPIrAWOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIrAW_F32(f32InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f32Output = GFLIB_ControllerPIrAW(f32InErr, &trMyPI, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    f32Output = GFLIB_ControllerPIrAW(f32InErr, &trMyPI);
}
```

2.26.7.4 Function GFLIB_ControllerPIrAWSetState_F16

2.26.7.4.1 Declaration

```
void GFLIB_ControllerPIrAWSetState_F16(tFrac16
f16ControllerPIrAWOut, GFLIB_CONTROLLER_PIAW_R_T_F16 *const
pParam);
```

2.26.7.4.2 Arguments

Table 202. GFLIB_ControllerPIrAWSetState_F16 arguments

Type	Name	Direction	Description
tFrac16	f16ControllerPIrAWOut	**input**	Required output of the GFLIB_ControllerPIrAW.
GFLIB_CONTROLLER_PIAW_R_T_F16 *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIrAW state.

2.26.7.4.3 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PIAW_R_T_F16 trMyPI = GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
void main(void)
{
    tFrac16 f16ControllerPIrAWOut;

    // Set the input error
    f16InErr = FRAC16(0.25);

    // Initialize the controller parameters
    trMyPI.f16CC1sc      = FRAC16(0.01);
    trMyPI.f16CC2sc      = FRAC16(0.02);
    trMyPI.u16NShift     = 1;
    trMyPI.f16UpperLimit = FRAC16(1.0);
    trMyPI.f16LowerLimit = FRAC16(-1.0);

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWInit_F16(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWInit(&trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrAWInit(&trMyPI);


    // Initialize the state variables to predefined values
    f16ControllerPIrAWOut = FRAC16(0.03);
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWSetState_F16(f16ControllerPIrAWOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWSetState(f16ControllerPIrAWOut, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    GFLIB_ControllerPIrAWSetState(f16ControllerPIrAWOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIrAW_F16(f16InErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f16Output = GFLIB_ControllerPIrAW(f16InErr, &trMyPI, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    f16Output = GFLIB_ControllerPIrAW(f16InErr, &trMyPI);
}
```

2.26.7.5 Function GFLIB_ControllerPIrAWSetState_FLT

2.26.7.5.1 Declaration

```
void GFLIB_ControllerPIrAWSetState_FLT(tFloat
fltControllerPIrAWOut, GFLIB_CONTROLLER_PIAW_R_T_FLT *const
pParam);
```

2.26.7.5.2 Arguments

Table 203. GFLIB_ControllerPIrAWSetState_FLT arguments

Type	Name	Direction	Description
tFloat	fltControllerPIrAWOut	**input**	Required output of the GFLIB_ControllerPIrAW.
GFLIB_CONTROLLER_PIAW_R_T_FLT *const	pParam	**input, output**	Pointer to the structure with GFLIB_ ControllerPIrAW state.

2.26.7.5.3 Code Example

```
#include "gflib.h"

tFloat fltInErr;
tFloat fltOutput;

GFLIB_CONTROLLER_PIAW_R_T_FLT trMyPI = GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT;

void main(void)
{
    tFloat fltControllerPIrAWOut;

    // Set the input error
    fltInErr = 0.25f;

    // Initialize the controller parameters
    trMyPI.fltCC1sc      = 0.01F;
    trMyPI.fltCC2sc      = 0.02F;
    trMyPI.fltUpperLimit = 1.0F;
    trMyPI.fltLowerLimit = -1.0F;

    // Clear the state variables
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWInit_FLT(&trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWInit(&trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIrAWInit(&trMyPI);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Initialize the state variables to predefined values
    fltControllerPIrAWOut = 0.03f;
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWSetState_FLT(fltControllerPIrAWOut, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    GFLIB_ControllerPIrAWSetState(fltControllerPIrAWOut, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    GFLIB_ControllerPIrAWSetState(fltControllerPIrAWOut, &trMyPI);
}

// Periodical function or interrupt - control loop
void ControlLoop(void)
{
    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIrAW_FLT(fltInErr, &trMyPI);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    fltOutput = GFLIB_ControllerPIrAW(fltInErr, &trMyPI, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    fltOutput = GFLIB_ControllerPIrAW(fltInErr, &trMyPI);
}
```

2.27 Function GFLIB_Cos

This function implements an approximation of cosine function.

2.27.1 Description

The GFLIB_Cos function provides a computational method for calculation of the trigonometric cosine function cos(x), using the piece-wise polynomial approximation.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 89. Course of the function GFLIB_Cos

When both sine and cosine of the same argument is needed, use GFLIB_SinCos function instead for better performance.

2.27.2 Re-entrancy

The function is re-entrant.

2.27.3 Function GFLIB_Cos_F32

2.27.3.1 Declaration

```
tFrac32 GFLIB_Cos_F32(tFrac32 f32In, const GFLIB_COS_T_F32 *const
pParam);
```

2.27.3.2 Arguments

Table 204. GFLIB_Cos_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument is a 32-bit number that contains an angle in radians from interval [- π, π) normalized between [-1, 1).
const GFLIB_COS_T_F32 *const	pParam	**input**	Pointer to an array of Taylor coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_COS_DEFAULT_F32 symbol.

2.27.3.3 Return

The function returns the value of the cosine function of the input argument as a fixed point 32-bit number, normalized between [-1, 1).

2.27.3.4 Implementation details

The computational algorithm uses the relation between cosine and sine function as shown in following equation:

$$\cos(x) = sin\left(\frac{\pi}{2} + x\right)$$

Equation GFLIB_Cos_F32_Eq1

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The input values are scaled from [-π, π) radians to [-1, 1) in order to fit in the available fixed-point fractional range. The function uses a 9th order Taylor polynomial approximation; the default polynomial coefficients are provided in the GFLIB_COS_DEFAULT_F32 structure.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.27.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32Angle;
tFrac32 f32Output;

void main(void)
{
    // input angle = 0.25 => pi/4
    f32Angle  = FRAC32(0.25);

    // output should be 0x5A827E94
    f32Output = GFLIB_Cos_F32(f32Angle, GFLIB_COS_DEFAULT_F32);

    // output should be 0x5A827E94
    f32Output = GFLIB_Cos(f32Angle, GFLIB_COS_DEFAULT_F32, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x5A827E94
    f32Output = GFLIB_Cos(f32Angle);
}
```

2.27.4 Function GFLIB_Cos_F16

2.27.4.1 Declaration

```
tFrac16 GFLIB_Cos_F16(tFrac16 f16In, const GFLIB_COS_T_F16 *const
pParam);
```

2.27.4.2 Arguments

Table 205. GFLIB_Cos_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument is a 16-bit number that contains an angle in radians from interval [- π, π) normalized between [-1, 1).
const GFLIB_COS_T_F16 *const	pParam	**input**	Pointer to an array of Taylor coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_COS_DEFAULT_F16 symbol.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.27.4.3 Return

The function returns the value of the cosine function of the input argument as a fixed point 16-bit number, normalized between [-1, 1).

2.27.4.4 Implementation details

The computational algorithm uses the relation between cosine and sine function as shown in following equation:

$$\cos(x) = sin\left(\frac{\pi}{2} + x\right)$$

Equation GFLIB_Cos_F16_Eq1

The input values are scaled from [-π, π) radians to [-1, 1) in order to fit in the available fixed-point fractional range. The function uses a 7th order Taylor polynomial approximation; the default polynomial coefficients are provided in the GFLIB_COS_DEFAULT_F16 structure.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.27.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16Angle;
tFrac16 f16Output;

void main(void)
{
  // input angle = 0.25 => pi/4
  f16Angle  = FRAC16(0.25);

  // output should be 0x5A82
  f16Output = GFLIB_Cos_F16(f16Angle, GFLIB_COS_DEFAULT_F16);

  // output should be 0x5A82
  f16Output = GFLIB_Cos(f16Angle, GFLIB_COS_DEFAULT_F16, F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be 0x5A82
  f16Output = GFLIB_Cos(f16Angle);
}
```

2.27.5 Function GFLIB_Cos_FLT

2.27.5.1 Declaration

```
tFloat GFLIB_Cos_FLT(tFloat fltIn, const GFLIB_COS_T_FLT *const
pParam);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.27.5.2 Arguments

Table 206. GFLIB_Cos_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument is a single precision floating point number that contains an angle in radians from interval [- π, π].
const GFLIB_COS_T_FLT *const	pParam	**input**	Pointer to an array of approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_COS_DEFAULT_FLT symbol.

2.27.5.3 Return

The function returns the value of the cosine function of the input argument as a single precision floating point number.

2.27.5.4 Implementation details

The computational algorithm uses the relation between cosine and sine function as shown in following equation:

$$\cos(x) = sin\left(\frac{\pi}{2} + x\right)$$

Equation GFLIB_Cos_FLT_Eq1

The function uses a 7th order minimax polynomial approximation; the default polynomial coefficients are provided in the GFLIB_COS_DEFAULT_FLT structure.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.

2.27.5.5 Code Example

```
#include "gflib.h"


tFloat fltAngle;
tFloat fltOutput;

void main(void)
{
  // input angle = 0.785398163 = pi/4
  fltAngle  = (tFloat)(0.785398163);

  // output should be 0.70710678
  fltOutput = GFLIB_Cos_FLT(fltAngle, GFLIB_COS_DEFAULT_FLT);

  // output should be 0.70710678
  fltOutput = GFLIB_Cos(fltAngle, GFLIB_COS_DEFAULT_FLT, FLT);

  // ##########################################################
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // Available only if single precision floating point
  // implementation selected as default
  // ##########################################################

  // output should be 0.70710678
  fltOutput = GFLIB_Cos(fltAngle);
}
```

2.28 Function GFLIB_Hyst

2.28.1 Description

The GFLIB_Hyst function provides a computational method for the calculation of a hysteresis (relay) function. The function switches the output between the two predefined values stored in the `OutValOn` and `OutValOff` members of parameter structure. When the value of the input is higher than the upper threshold `HystOn`, then the output value is equal to `OutValOn`. On the other hand, when the input value is lower than the lower threshold `HystOff`, then the output value is equal to `OutValOff`. When the input value is between these two threshold values then the output retains its value (the previous state).

$$OutState(k) = \begin{cases} OutValOn & if \quad In \geq HystOn \\ OutValOff & if \quad In \leq HystOff \\ OutState(k-1) & if \quad otherwise \end{cases}$$

Equation GFLIB_Hyst_Eq1

A graphical description of GFLIB_Hyst functionality is shown in Figure 90 .

Figure 90. Hysteresis function

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.28.2 Re-entrancy

The function is re-entrant for a different `pParam`.

2.28.3 Function GFLIB_Hyst_F32

2.28.3.1 Declaration

```
tFrac32 GFLIB_Hyst_F32(tFrac32 f32In, GFLIB_HYST_T_F32 *const
pParam);
```

2.28.3.2 Arguments

Table 207. GFLIB_Hyst_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input signal in the form of a 32-bit fixed point number, normalized between [-1, 1).
GFLIB_HYST_T_F32 *const	pParam	**input, output**	Pointer to the structure with parameters and states of the hysteresis function. Arguments of the structure contain fixed point 32-bit values, normalized between [-1, 1).

2.28.3.3 Return

The function returns the value of the hysteresis output, which is equal to either `f32OutValOn` or `f32OutValOff` depending on the value of the input and the state of the function output in the previous calculation step. The output value is interpreted as a fixed point 32-bit number, normalized between [-1, 1).

CAUTION: *For correct functionality, the threshold `f32HystOn` value must be greater than the `f32HystOff` value.*

Note: *All parameters and states used by the function can be reset during declaration using the* GFLIB_HYST_DEFAULT_F32 *macro.*

2.28.3.4 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;
GFLIB_HYST_T_F32 f32trMyHyst = GFLIB_HYST_DEFAULT_F32;

void main(void)
{
    // Setting parameters for hysteresis
    f32trMyHyst.f32HystOn = FRAC32(0.1289);
    f32trMyHyst.f32HystOff = FRAC32(-0.3634);
    f32trMyHyst.f32OutValOn = FRAC32(0.589);
    f32trMyHyst.f32OutValOff = FRAC32(-0.123);
    f32trMyHyst.f32OutState = FRAC32(-0.3333);

    // input value = -0.41115
    f32In = FRAC32(-0.41115);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // output should be 0x8FBE76C8 ~ FRAC32(-0.123)
    f32Out = GFLIB_Hyst_F32(f32In,&f32trMyHyst);

    // output should be 0x8FBE76C8 ~ FRAC32(-0.123)
    f32trMyHyst.f32OutState = FRAC32(0);
    f32Out = GFLIB_Hyst(f32In, &f32trMyHyst, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x8FBE76C8 ~ FRAC32(-0.123)
    f32trMyHyst.f32OutState = FRAC32(0);
    f32Out = GFLIB_Hyst(f32In, &f32trMyHyst);
}
```

2.28.4 Function GFLIB_Hyst_F16

2.28.4.1 Declaration

```
tFrac16 GFLIB_Hyst_F16(tFrac16 f16In, GFLIB_HYST_T_F16 *const
pParam);
```

2.28.4.2 Arguments

Table 208. GFLIB_Hyst_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input signal in the form of a 16-bit fixed point number, normalized between [-1, 1).
GFLIB_HYST_T_F16 *const	pParam	**input, output**	Pointer to the structure with parameters and states of the hysteresis function. Arguments of the structure contain fixed point 16-bit values, normalized between [-1, 1).

2.28.4.3 Return

The function returns the value of the hysteresis output, which is equal to either `f16OutValOn` or `f16OutValOff` depending on the value of the input and the state of the function output in the previous calculation step. The output value is interpreted as a fixed point 16-bit number, normalized between [-1, 1).

CAUTION: *For correct functionality, the threshold `f16HystOn` value must be greater than the `f16HystOff` value.*

Note: *All parameters and states used by the function can be reset during declaration using the GFLIB_HYST_DEFAULT_F16 macro.*

2.28.4.4 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;
GFLIB_HYST_T_F16 f16trMyHyst = GFLIB_HYST_DEFAULT_F16;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
void main(void)
{
    // Setting parameters for hysteresis
    f16trMyHyst.f16HystOn = FRAC16(0.1289);
    f16trMyHyst.f16HystOff = FRAC16(-0.3634);
    f16trMyHyst.f16OutValOn = FRAC16(0.589);
    f16trMyHyst.f16OutValOff = FRAC16(-0.123);
    f16trMyHyst.f16OutState = FRAC16(-0.3333);

    // input value = -0.41115
    f16In = FRAC16(-0.41115);

    // output should be 0x8FBE ~ FRAC16(-0.123)
    f16Out = GFLIB_Hyst_F16(f16In,&f16trMyHyst);

    // output should be 0x8FBE ~ FRAC16(-0.123)
    f16trMyHyst.f16OutState = FRAC16(0);
    f16Out = GFLIB_Hyst(f16In, &f16trMyHyst, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x8FBE ~ FRAC16(-0.123)
    f16trMyHyst.f16OutState = FRAC16(0);
    f16Out = GFLIB_Hyst(f16In, &f16trMyHyst);
}
```

2.28.5 Function GFLIB_Hyst_FLT

2.28.5.1 Declaration

```
tFloat GFLIB_Hyst_FLT(tFloat fltIn, GFLIB_HYST_T_FLT *const
pParam);
```

2.28.5.2 Arguments

Table 209. GFLIB_Hyst_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input value, in single precision floating point data format.
GFLIB_HYST_T_FLT *const	pParam	**input, output**	Pointer to the structure with parameters and states of the hysteresis function. Arguments of the structure contain a single precision floating point values.

2.28.5.3 Return

The function returns the value of the hysteresis output, which is equal to either `fltOutValOn` or `fltOutValOff` depending on the value of the input and the state of the function output in the previous calculation step. The output value is in single precision floating point format.

Note: *The function may raise floating-point exceptions (invalid operation, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

CAUTION: *For correct functionality, the threshold* `fltHystOn` *value must be greater than the* `fltHystOff` *value.*

Note: *All parameters and states used by the function can be reset during declaration using the* GFLIB_HYST_DEFAULT_FLT *macro.*

2.28.5.4 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;
GFLIB_HYST_T_FLT flttrMyHyst = GFLIB_HYST_DEFAULT_FLT;

void main(void)
{
    // Setting parameters for hysteresis
    flttrMyHyst.fltHystOn = (tFloat)(0.1289);
    flttrMyHyst.fltHystOff = (tFloat)(-0.3634);
    flttrMyHyst.fltOutValOn = (tFloat)(0.589);
    flttrMyHyst.fltOutValOff = (tFloat)(-0.123);
    flttrMyHyst.fltOutState = (tFloat)(-0.3333);

    // input value = -0.41115
    fltIn = (tFloat)(-0.41115);

    // output should be -0.123
    fltOut = GFLIB_Hyst_FLT(fltIn,&flttrMyHyst);

    // output should be -0.123
    flttrMyHyst.fltOutState = (tFloat)(0);
    fltOut = GFLIB_Hyst(fltIn, &flttrMyHyst, FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be -0.123
    flttrMyHyst.fltOutState = (tFloat)(0);
    fltOut = GFLIB_Hyst(fltIn, &flttrMyHyst);
}
```

2.29 Function GFLIB_IntegratorTR

The function implements discrete integrator using the trapezoidal (bilinear) transform.

2.29.1 Description

The continuous-time integrator is defined as:

$$u(t) = \int_0^t e(t)\,dt$$

Equation GFLIB_IntegratorTR_Eq1

The integrator transfer function can be described using the Laplace transform as follows:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$H(s) = \frac{U(s)}{E(s)} = \frac{1}{s}$$

Equation GFLIB_IntegratorTR_Eq2

The continuous Laplace-domain transfer function GFLIB_IntegratorTR_Eq2 can be related to the discrete Z-domain transfer function using the trapezoidal (bilinear) transform:

$$\mathbb{Z}\{H(s)\} = \mathbb{Z}\left\{\frac{U(s)}{E(s)}\right\} \approx \frac{T_S}{2} \cdot \frac{1+z^{-1}}{1-z^{-1}}$$

Equation GFLIB_IntegratorTR_Eq3

where T_s is the sampling period. The difference equation implementing the Z-domain transfer function in GFLIB_IntegratorTR_Eq3 is as follows:

$$u(k) = u(k-1) + e(k) \cdot \frac{T_S}{2} + e(k-1) \cdot \frac{T_S}{2}$$

Equation GFLIB_IntegratorTR_Eq4

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.29.2 Re-entrancy

The function is re-entrant for a different `pParam`.

2.29.3 Function GFLIB_IntegratorTR_F32

2.29.3.1 Declaration

```
tFrac32 GFLIB_IntegratorTR_F32(tFrac32 f32In,
GFLIB_INTEGRATOR_TR_T_F32 *const pParam);
```

2.29.3.2 Arguments

Table 210. GFLIB_IntegratorTR_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument to be integrated.
GFLIB_INTEGRATOR_TR_T_F32 *const	pParam	**input, output**	Pointer to the integrator parameters structure.

2.29.3.3 Return

The function returns a 32-bit value in format Q1.31, which represents the actual integrated value of the input signal with overflow on range boundaries.

2.29.3.4 Implementation details

Prior to calculating the integration coefficient, it is necessary to set the scaling constants. The input and output of the algorithm are limited to the fractional range [-1, 1), therefore it is necessary to scale the input and output physical quantities (e.g. speed, angle) according to the expected maxima:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

- E^{MAX} - maximal value of the integrator input.
- U^{MAX} - maximal value of the integrator output.

If the integrator input represents the speed of a motor, then the integrator calculates the rotor angle at the output. In this case, the output scale must be set $U^{MAX} = \pi$ or $U^{MAX} = 180°$, depending on the chosen units (radians or degrees). This scale ensures that the output fractional range [-1, 1) of the integrator corresponds to one complete revolution of the rotor, i.e. [-π, π) or [-180°, 180°). Note that the measurement units of the speed must be compatible with the output units and the time units chosen for the sampling period T_S. For example, a compatible set of units is the following:

- Input speed $e(k)$ measured in radians per second.
- Output angle $u(k)$ measured in radians.
- Sampling period T_S measured in seconds.

The input and output signals are scaled to the fractional range as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$
$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation GFLIB_IntegratorTR_F32_Eq1

The discrete implementation of the integrator is given as follows:

$$u_f(k) \cdot U^{MAX} = u_f(k-1) \cdot U^{MAX} + e_f(k) \cdot E^{MAX} \cdot \frac{T_S}{2} + e_f(k-1) \cdot E^{MAX} \cdot \frac{T_S}{2}$$

Equation GFLIB_IntegratorTR_F32_Eq2

which can be rearranged into the following form:

$$u_f(k) = u_f(k-1) + e_f(k) \cdot C1_{SC} + e_f(k-1) \cdot C1_{SC}$$

Equation GFLIB_IntegratorTR_F32_Eq3

where $C1_{SC}$ is the scaled integrator constant:

$$C1_{SC} = \frac{T_S}{2} \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_IntegratorTR_F32_Eq4

In case the coefficient $C1_{SC}$ exceeds the fractional range [-1, 1), its value must be right-shifted by *u16NShift*. The binary shift amount corresponds to

$$u16Nshift = \max\left(0, \left\lceil \log_2 \left| C1_{SC} \right| \right\rceil\right)$$

Equation GFLIB_IntegratorTR_F32_Eq5

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficient fits in the fractional range and can be recorded into the parameters structure. To represent the coefficient in integer format expected by the C language, simply shift the radix point by multiplying the coefficient value by the 31st power of two (corresponds to the Q1.31 encoding):

$$f32C1 = \left(C1_{SC} \cdot 2^{-u16NShift}\right) \cdot 2^{31}$$

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Equation GFLIB_IntegratorTR_F32_Eq6

If the output exceeds the fractional range [-1,1), an overflow occurs. This behavior allows continual integration of an angular velocity of a rotor to obtain the actual rotor position, assuming the output range corresponds to one complete revolution.

Note: *All parameters and states used by the function can be reset during declaration using the* GFLIB_INTEGRATOR_TR_DEFAULT_F32 *macro.*

The specified accuracy of the function is not guaranteed in cases when any of the terms in GFLIB_IntegratorTR_F32_Eq3 *overflows.*

2.29.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;

// Definition of one integrator instance
GFLIB_INTEGRATOR_TR_T_F32 trMyIntegrator = GFLIB_INTEGRATOR_TR_DEFAULT_F32;

void main(void)
{
    // Setting parameters for integrator, Ts = 100e-4, E_MAX=U_MAX=1
    trMyIntegrator.f32C1     = FRAC32((100e-4)/2.0);
    trMyIntegrator.u16NShift = (tU16)0;

    // input value = 0.5
    f32In = FRAC32(0.5);

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    // Initialization of integrator internal state for zero output:
    GFLIB_IntegratorTRSetState_F32(0, &trMyIntegrator);
    // Calculation of one iteration of the integrator:
    f32Out = GFLIB_IntegratorTR_F32(f32In, &trMyIntegrator);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    // Initialization of integrator internal state for zero output:
    GFLIB_IntegratorTRSetState(0, &trMyIntegrator, F32);
    // Calculation of one iteration of the integrator:
    f32Out = GFLIB_IntegratorTR(f32In, &trMyIntegrator, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    // Initialization of integrator internal state for zero output:
    GFLIB_IntegratorTRSetState(0, &trMyIntegrator);
    // Calculation of one iteration of the integrator:
    f32Out = GFLIB_IntegratorTR(f32In, &trMyIntegrator);

    // Expected output value: 0x0051EB85
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.29.4 Function GFLIB_IntegratorTR_F16

2.29.4.1 Declaration

```
tFrac16 GFLIB_IntegratorTR_F16(tFrac16 f16In,
GFLIB_INTEGRATOR_TR_T_F16 *const pParam);
```

2.29.4.2 Arguments

Table 211. GFLIB_IntegratorTR_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument to be integrated.
GFLIB_INTEGRATOR_TR_T_F16 *const	pParam	**input, output**	Pointer to the integrator parameters structure.

2.29.4.3 Return

The function returns a 16-bit value in format Q1.15, which represents the actual integrated value of the input signal with overflow on range boundaries.

2.29.4.4 Implementation details

Prior to calculating the integration coefficient, it is necessary to set the scaling constants. The input and output of the algorithm are limited to the fractional range [-1, 1), therefore it is necessary to scale the input and output physical quantities (e.g. speed, angle) according to the expected maxima:

- E^{MAX} - maximal value of the integrator input.
- U^{MAX} - maximal value of the integrator output.

If the integrator input represents the speed of a motor, then the integrator calculates the rotor angle at the output. In this case, the output scale must be set $U^{MAX} = \pi$ or $U^{MAX} = 180°$, depending on the chosen units (radians or degrees). This scale ensures that the output fractional range [-1, 1) of the integrator corresponds to one complete revolution of the rotor, i.e. [-π, π) or [-180°, 180°). Note that the measurement units of the speed must be compatible with the output units and the time units chosen for the sampling period T_S. For example, a compatible set of units is the following:

- Input speed $e(k)$ measured in radians per second.
- Output angle $u(k)$ measured in radians.
- Sampling period T_S measured in seconds.

The input and output signals are scaled to the fractional range as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation GFLIB_IntegratorTR_F16_Eq1

The discrete implementation of the integrator is given as follows:

$$u_f(k) \cdot U^{MAX} = u_f(k-1) \cdot U^{MAX} + e_f(k) \cdot E^{MAX} \cdot \frac{T_S}{2} + e_f(k-1) \cdot E^{MAX} \cdot \frac{T_S}{2}$$

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Equation GFLIB_IntegratorTR_F16_Eq2

which can be rearranged into the following form:

$$u_f(k) = u_f(k-1) + e_f(k) \cdot C1_{SC} + e_f(k-1) \cdot C1_{SC}$$

Equation GFLIB_IntegratorTR_F16_Eq3

where $C1_{SC}$ is the scaled integrator constant:

$$C1_{SC} = \frac{T_s}{2} \cdot \frac{E^{MAX}}{U^{MAX}}$$

Equation GFLIB_IntegratorTR_F16_Eq4

In case the coefficient $C1_{SC}$ exceeds the fractional range [-1, 1), its value must be right-shifted by *u16NShift*. The binary shift amount corresponds to

$$u16Nshift = \max\left(0, \left\lceil \log_2 \left| C1_{SC} \right| \right\rceil\right)$$

Equation GFLIB_IntegratorTR_F16_Eq5

the ceiling brackets represent rounding towards positive infinity. After the shift, the coefficient fits in the fractional range and can be recorded into the parameters structure. To represent the coefficient in integer format expected by the C language, simply shift the radix point by multiplying the coefficient value by the 15th power of two (corresponds to the Q1.15 encoding):

$$f16C1 = \left(C1_{SC} \cdot 2^{-u16NShift}\right) \cdot 2^{15}$$

Equation GFLIB_IntegratorTR_F16_Eq6

If the output exceeds the fractional range [-1,1), an overflow occurs. This behavior allows continual integration of an angular velocity of a rotor to obtain the actual rotor position, assuming the output range corresponds to one complete revolution.

Note: *All parameters and states used by the function can be reset during declaration using the* GFLIB_INTEGRATOR_TR_DEFAULT_F16 *macro.*

The specified accuracy of the function is not guaranteed in cases when any of the terms in GFLIB_IntegratorTR_F16_Eq3 *overflows.*

2.29.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;

// Definition of one integrator instance
GFLIB_INTEGRATOR_TR_T_F16 trMyIntegrator = GFLIB_INTEGRATOR_TR_DEFAULT_F16;

void main(void)
{
    // Setting parameters for integrator, Ts = 100e-4, E_MAX=U_MAX=1
    trMyIntegrator.f16C1      = FRAC16((100e-4)/2.0);
    trMyIntegrator.u16NShift = (tU16)0;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // input value = 0.5
    f16In = FRAC16(0.5);

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    // Initialization of integrator internal state for zero output:
    GFLIB_IntegratorTRSetState_F16(0, &trMyIntegrator);
    // Calculation of one iteration of the integrator:
    f16Out = GFLIB_IntegratorTR_F16(f16In, &trMyIntegrator);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    // Initialization of integrator internal state for zero output:
    GFLIB_IntegratorTRSetState(0, &trMyIntegrator, F16);
    // Calculation of one iteration of the integrator:
    f16Out = GFLIB_IntegratorTR(f16In, &trMyIntegrator, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    // Initialization of integrator internal state for zero output:
    GFLIB_IntegratorTRSetState(0, &trMyIntegrator);
    // Calculation of one iteration of the integrator:
    f16Out = GFLIB_IntegratorTR(f16In, &trMyIntegrator);

    // Expected output value: 0x0051
}
```

2.29.5 Function GFLIB_IntegratorTR_FLT

2.29.5.1 Declaration

```
tFloat GFLIB_IntegratorTR_FLT(tFloat fltIn,
GFLIB_INTEGRATOR_TR_T_FLT *const pParam);
```

2.29.5.2 Arguments

Table 212. GFLIB_IntegratorTR_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument to be integrated.
GFLIB_INTEGRATOR_TR_T_FLT *const	pParam	**input, output**	Pointer to the integrator parameters structure.

2.29.5.3 Return

The function returns a single precision floating point value which represents the actual integrated value of the input signal.

2.29.5.4 Implementation details

Note that the floating-point implementation does not wrap the output value around any particular interval. To use the integrator to calculate the motor position from angular velocity, it is necessary to add output wrapping around the [-π, π) or [-180°, 180°)

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

interval, depending on the chosen units (radians or degrees). The following code example shows wrapping of the calculated angle *fltAngle* around the interval [-π, π):

```
#include "gflib.h"

GFLIB_INTEGRATOR_TR_T_FLT trMyIntegrator = GFLIB_INTEGRATOR_TR_DEFAULT_FLT;
tFloat fltAngularVelocity;
tFloat fltAngle;

void main(void)
{
  fltAngle = GFLIB_IntegratorTR_FLT(fltAngularVelocity, &trMyIntegrator);
  if (fltAngle >= FLOAT_PI){
    fltAngle = fltAngle - FLOAT_2_PI;
    trMyIntegrator.fltState = fltAngle;
  }else{
    if (fltAngle < -FLOAT_PI){
      fltAngle = fltAngle + FLOAT_2_PI;
      trMyIntegrator.fltState = fltAngle;
    }
  }
}
```

Alternatively, it is possible to use the fixed-point implementation (GFLIB_IntegratorTR_F32 or GFLIB_IntegratorTR_F16) and apply an appropriate scaling and type conversion of the input and output between the tFloat and tFrac32 (or tFrac16) types. The following code example demonstrates usage of the GFLIB_IntegratorTR_F32 function in a floating-point application:

```
#include "gflib.h"

// max. velocity e.g. 1000 rad/s:
#define Emax 1000.0f
// angle in the [-π, π) range:
#define Umax FLOAT_PI
// sampling period e.g. 1 milisecond
#define Ts 1e-3f
tFloat fltAngularVelocity;
tFloat fltAngle;
tFrac32 f32In;
tFrac32 f32Out;
GFLIB_INTEGRATOR_TR_T_F32 trMyIntegrator = GFLIB_INTEGRATOR_TR_DEFAULT_F32;

void main(void)
{
  // Setting parameters for the fixed-point integrator
  trMyIntegrator.f32C1 = FRAC32((Ts/2.0f)*(Emax/Umax));
  trMyIntegrator.u16NShift = (tU16)0;

  // One-time initialization of integrator internal state for zero output:
  GFLIB_IntegratorTRSetState_F32(0, &trMyIntegrator);

  // Conversion of the floating-point input to fixed-point:
  f32In = MLIB_ConvertPU_F32FLT(fltAngularVelocity*(1.0f/Emax));
  // Calculation of one iteration of the fixed-point integrator:
  f32Out = GFLIB_IntegratorTR_F32(f32In, &trMyIntegrator);
  // Conversion of the fixed-point output to floating-point:
  fltAngle = MLIB_ConvertPU_FLTF32(f32Out)*Umax;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
}
```

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

All parameters and states used by the function can be reset during declaration using the GFLIB_INTEGRATOR_TR_DEFAULT_FLT *macro.*

2.29.5.5 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;

// Definition of one integrator instance
GFLIB_INTEGRATOR_TR_T_FLT trMyIntegrator = GFLIB_INTEGRATOR_TR_DEFAULT_FLT;

void main(void)
{
    // Setting parameters for integrator, Ts = 100e-4,
    trMyIntegrator.fltC1 = (tFloat)((100e-4)/2.0);

    // input value = 0.5
    fltIn = (tFloat)0.5;

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    // Initialization of integrator internal state for zero output:
    GFLIB_IntegratorTRSetState_FLT(0.0f, &trMyIntegrator);
    // Calculation of one iteration of the integrator:
    fltOut = GFLIB_IntegratorTR_FLT(fltIn, &trMyIntegrator);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    // Initialization of integrator internal state for zero output:
    GFLIB_IntegratorTRSetState(0.0f, &trMyIntegrator, FLT);
    // Calculation of one iteration of the integrator:
    fltOut = GFLIB_IntegratorTR(fltIn, &trMyIntegrator, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    // Initialization of integrator internal state for zero output:
    GFLIB_IntegratorTRSetState(0.0f, &trMyIntegrator);
    // Calculation of one iteration of the integrator:
    fltOut = GFLIB_IntegratorTR(fltIn, &trMyIntegrator);

    // Expected output value: 2.5e-3
}
```

2.29.6 Function GFLIB_IntegratorTRSetState

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.29.6.1 Description

This function initializes the GFLIB_IntegratorTR state variables to achieve the required output values.

Note: *The input/output pointer must contain a valid address, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.29.6.2 Re-entrancy

The function is re-entrant for a different `pParam`.

2.29.6.3 Function GFLIB_IntegratorTRSetState_F32

2.29.6.3.1 Declaration

```
void GFLIB_IntegratorTRSetState_F32(tFrac32 f32IntegratorTROut,
GFLIB_INTEGRATOR_TR_T_F32 *const pParam);
```

2.29.6.3.2 Arguments

Table 213. GFLIB_IntegratorTRSetState_F32 arguments

Type	Name	Direction	Description
tFrac32	f32IntegratorTROut	**input**	Required output of the GFLIB_IntegratorTR.
GFLIB_INTEGRATOR_TR_T_F32 *const	pParam	**input, output**	Pointer to the structure with GFLIB_IntegratorTR state.

2.29.6.3.3 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;

// Definition of one integrator instance
GFLIB_INTEGRATOR_TR_T_F32 trMyIntegrator = GFLIB_INTEGRATOR_TR_DEFAULT_F32;

void main(void)
{
    // Setting parameters for integrator, Ts = 100e-4, E_MAX=U_MAX=1
    trMyIntegrator.f32C1     = FRAC32((100e-4)/2.0);
    trMyIntegrator.u16NShift = (tU16)0;

    // input value = 0.5
    f32In = FRAC32(0.5);

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    // Initialization of integrator internal state for zero output:
    GFLIB_IntegratorTRSetState_F32(0, &trMyIntegrator);
    // Calculation of one iteration of the integrator:
    f32Out = GFLIB_IntegratorTR_F32(f32In, &trMyIntegrator);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    // Initialization of integrator internal state for zero output:
    GFLIB_IntegratorTRSetState(0, &trMyIntegrator, F32);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Calculation of one iteration of the integrator:
    f32Out = GFLIB_IntegratorTR(f32In, &trMyIntegrator, F32);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    // Initialization of integrator internal state for zero output:
    GFLIB_IntegratorTRSetState(0, &trMyIntegrator);
    // Calculation of one iteration of the integrator:
    f32Out = GFLIB_IntegratorTR(f32In, &trMyIntegrator);

    // Expected output value: 0x0051EB85
}
```

2.29.6.4 Function GFLIB_IntegratorTRSetState_F16

2.29.6.4.1 Declaration

```
void GFLIB_IntegratorTRSetState_F16(tFrac16 f16IntegratorTROut,
GFLIB_INTEGRATOR_TR_T_F16 *const pParam);
```

2.29.6.4.2 Arguments

Table 214. GFLIB_IntegratorTRSetState_F16 arguments

Type	Name	Direction	Description
tFrac16	f16IntegratorTROut	**input**	Required output of the GFLIB_IntegratorTR.
GFLIB_INTEGRATOR_TR_T_F16 *const	pParam	**input, output**	Pointer to the structure with GFLIB_IntegratorTR state.

2.29.6.4.3 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;

// Definition of one integrator instance
GFLIB_INTEGRATOR_TR_T_F16 trMyIntegrator = GFLIB_INTEGRATOR_TR_DEFAULT_F16;

void main(void)
{
    // Setting parameters for integrator, Ts = 100e-4, E_MAX=U_MAX=1
    trMyIntegrator.f16C1     = FRAC16((100e-4)/2.0);
    trMyIntegrator.u16NShift = (tU16)0;

    // input value = 0.5
    f16In = FRAC16(0.5);

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    // Initialization of integrator internal state for zero output:
    GFLIB_IntegratorTRSetState_F16(0, &trMyIntegrator);
    // Calculation of one iteration of the integrator:
    f16Out = GFLIB_IntegratorTR_F16(f16In, &trMyIntegrator);

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Initialization of integrator internal state for zero output:
    GFLIB_IntegratorTRSetState(0, &trMyIntegrator, F16);
    // Calculation of one iteration of the integrator:
    f16Out = GFLIB_IntegratorTR(f16In, &trMyIntegrator, F16);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    // Initialization of integrator internal state for zero output:
    GFLIB_IntegratorTRSetState(0, &trMyIntegrator);
    // Calculation of one iteration of the integrator:
    f16Out = GFLIB_IntegratorTR(f16In, &trMyIntegrator);

    // Expected output value: 0x0051
}
```

2.29.6.5 Function GFLIB_IntegratorTRSetState_FLT

2.29.6.5.1 Declaration

```
void GFLIB_IntegratorTRSetState_FLT(tFloat fltIntegratorTROut,
GFLIB_INTEGRATOR_TR_T_FLT *const pParam);
```

2.29.6.5.2 Arguments

Table 215. GFLIB_IntegratorTRSetState_FLT arguments

Type	Name	Direction	Description
tFloat	fltIntegratorTROut	**input**	Required output of the GFLIB_IntegratorTR.
GFLIB_INTEGRATOR_TR_T_FLT *const	pParam	**input, output**	Pointer to the structure with GFLIB_IntegratorTR state.

2.29.6.5.3 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;

// Definition of one integrator instance
GFLIB_INTEGRATOR_TR_T_FLT trMyIntegrator = GFLIB_INTEGRATOR_TR_DEFAULT_FLT;

void main(void)
{
    // Setting parameters for integrator, Ts = 100e-4,
    trMyIntegrator.fltC1 = (tFloat)((100e-4)/2.0);

    // input value = 0.5
    fltIn = (tFloat)0.5;

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    // Initialization of integrator internal state for zero output:
    GFLIB_IntegratorTRSetState_FLT(0.0f, &trMyIntegrator);
    // Calculation of one iteration of the integrator:
    fltOut = GFLIB_IntegratorTR_FLT(fltIn, &trMyIntegrator);

    // Alternative 2: API call with implementation parameter
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // (only one alternative shall be used).
    // Initialization of integrator internal state for zero output:
    GFLIB_IntegratorTRSetState(0.0f, &trMyIntegrator, FLT);
    // Calculation of one iteration of the integrator:
    fltOut = GFLIB_IntegratorTR(fltIn, &trMyIntegrator, FLT);

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if single precision floating-point implementation is selected
    // as default.
    // Initialization of integrator internal state for zero output:
    GFLIB_IntegratorTRSetState(0.0f, &trMyIntegrator);
    // Calculation of one iteration of the integrator:
    fltOut = GFLIB_IntegratorTR(fltIn, &trMyIntegrator);

    // Expected output value: 2.5e-3
}
```

2.30 Function GFLIB_Limit

This function tests whether the input value is within the upper and lower limits.

2.30.1 Description

The GFLIB_Limit function tests whether the input value is within the upper and lower limits. If so, the input value will be returned. If the input value is above the upper limit, the upper limit will be returned. If the input value is below the lower limit, the lower limit will be returned.

The upper and lower limits can be found in the limits structure, supplied to the function as a pointer `pParam`.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.30.2 Re-entrancy

The function is re-entrant.

2.30.3 Function GFLIB_Limit_F32

2.30.3.1 Declaration

```
tFrac32 GFLIB_Limit_F32(tFrac32 f32In, const GFLIB_LIMIT_T_F32
*const pParam);
```

2.30.3.2 Arguments

Table 216. GFLIB_Limit_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input value.
const GFLIB_LIMIT_T_F32 *const	pParam	**input**	Pointer to the limits structure.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.30.3.3 Return

The input value in case the input value is below the limits, or the upper or lower limit if the input value is above these limits.

Note: *The function assumes that the upper limit `f32UpperLimit` is greater than the lower limit `f32LowerLimit`. Otherwise, the function returns an undefined value.*

2.30.3.4 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;
GFLIB_LIMIT_T_F32 f32trMyLimit = GFLIB_LIMIT_DEFAULT_F32;

void main(void)
{
    // upper/lower limits
    f32trMyLimit.f32UpperLimit = FRAC32(0.5);
    f32trMyLimit.f32LowerLimit = FRAC32(-0.5);

    // input value = 0.75
    f32In = FRAC32(0.75);

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = GFLIB_Limit_F32(f32In, &f32trMyLimit);

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = GFLIB_Limit(f32In, &f32trMyLimit, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = GFLIB_Limit(f32In, &f32trMyLimit);
}
```

2.30.4 Function GFLIB_Limit_F16

2.30.4.1 Declaration

```
tFrac16 GFLIB_Limit_F16(tFrac16 f16In, const GFLIB_LIMIT_T_F16
*const pParam);
```

2.30.4.2 Arguments

Table 217. GFLIB_Limit_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input value.
const GFLIB_LIMIT_T_F16 *const	pParam	**input**	Pointer to the limits structure.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.30.4.3 Return

The input value in case the input value is below the limits, or the upper or lower limit if the input value is above these limits.

Note: *The function assumes that the upper limit `f16UpperLimit` is greater than the lower limit `f16LowerLimit`. Otherwise, the function returns an undefined value.*

2.30.4.4 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;
GFLIB_LIMIT_T_F16 f16trMyLimit = GFLIB_LIMIT_DEFAULT_F16;

void main(void)
{
    // upper/lower limits
    f16trMyLimit.f16UpperLimit = FRAC16(0.5);
    f16trMyLimit.f16LowerLimit = FRAC16(-0.5);

    // input value = 0.75
    f16In = FRAC16(0.75);

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = GFLIB_Limit_F16(f16In, &f16trMyLimit);

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = GFLIB_Limit(f16In, &f16trMyLimit, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = GFLIB_Limit(f16In, &f16trMyLimit);
}
```

2.30.5 Function GFLIB_Limit_FLT

2.30.5.1 Declaration

```
tFloat GFLIB_Limit_FLT(tFloat fltIn, const GFLIB_LIMIT_T_FLT
*const pParam);
```

2.30.5.2 Arguments

Table 218. GFLIB_Limit_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input value.
const GFLIB_LIMIT_T_FLT *const	pParam	**input**	Pointer to the limits structure.

© NXP B.V. 2023. All rights reserved.

2.30.5.3 Return

The input value in case the input value is below the limits, or the upper or lower limit if the input value is above these limits.

Note: *The function may raise floating-point exceptions (invalid operation, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

The function assumes that the upper limit `fltUpperLimit` is greater than the lower limit `fltLowerLimit`. Otherwise, the function returns an undefined value.

2.30.5.4 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;
GFLIB_LIMIT_T_FLT flttrMyLimit = GFLIB_LIMIT_DEFAULT_FLT;

void main(void)
{
    // upper/lower limits
    flttrMyLimit.fltUpperLimit = 0.5;
    flttrMyLimit.fltLowerLimit = -0.5;

    // input value = 0.75
    fltIn = 0.75;

    // output should be 0.5
    fltOut = GFLIB_Limit_FLT(fltIn, &flttrMyLimit);

    // output should be 0.5
    fltOut = GFLIB_Limit(fltIn, &flttrMyLimit, FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be 0.5
    fltOut = GFLIB_Limit(fltIn, &flttrMyLimit);
}
```

2.31 Function GFLIB_Log10_FLT

This function calculates a base-10 logarithm of an absolute value.

2.31.1 Declaration

```
tFloat GFLIB_Log10_FLT(tFloat fltIn, const GFLIB_LOG10_T_FLT
*const pParam);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.31.2 Arguments

Table 219. GFLIB_Log10_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument is a 32-bit number that contains a single precision floating point value.
const GFLIB_LOG10_T_FLT *const	pParam	**input**	Pointer to an array of approximation coefficients.

2.31.3 Return

The function returns log10(abs(fltIn)) as a single precision floating point number.

2.31.4 Description

The function calculates a base-10 logarithm of the absolute value of the input. The default polynomial coefficients are provided in the GFLIB_LOG10_DEFAULT_FLT structure.

Use GFLIB_VLog10_FLT vectorized function instead if there is an array of inputs to be calculated. The vectorized function can process large arrays faster.

Note: *The function may raise floating-point exceptions (overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.31.5 Code Example

```
#include "gflib.h"

tFloat fltInput, fltOutput;

void main(void)
{
  // input value = 1.0
  fltInput  = 1.0F;

  // output should be 0
  fltOutput = GFLIB_Log10_FLT(fltInput, GFLIB_LOG10_DEFAULT_FLT);

  // output should be 0
  fltOutput = GFLIB_Log10(fltInput, GFLIB_LOG10_DEFAULT_FLT, FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output should be 0
  fltOutput = GFLIB_Log10(fltInput);
}
```

2.32 Function GFLIB_LowerLimit

This function tests whether the input value is above the lower limit.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.32.1 Description

The function tests whether the input value is above the lower limit. If so, the input value will be returned. Otherwise, if the input value is below the lower limit, the lower limit will be returned.

The lower limit `LowerLimit` can be found in the limits structure, supplied to the function as a pointer `pParam`.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.32.2 Re-entrancy

The function is re-entrant.

2.32.3 Function GFLIB_LowerLimit_F32

2.32.3.1 Declaration

```
tFrac32 GFLIB_LowerLimit_F32(tFrac32 f32In, const
GFLIB_LOWERLIMIT_T_F32 *const pParam);
```

2.32.3.2 Arguments

Table 220. GFLIB_LowerLimit_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input value.
const GFLIB_LOWERLIMIT_T_F32 *const	pParam	**input**	Pointer to the limits structure.

2.32.3.3 Return

The input value in case the input value is above the limit, or the lower limit if the input value is below the limit.

2.32.3.4 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;
GFLIB_LOWERLIMIT_T_F32 f32trMyLowerLimit = GFLIB_LOWERLIMIT_DEFAULT_F32;

void main(void)
{
    // lower limit
    f32trMyLowerLimit.f32LowerLimit = FRAC32(0.5);

    // input value = 0.75
    f32In = FRAC32(0.75);

    // output should be 0x60000000 ~ FRAC32(0.75)
    f32Out = GFLIB_LowerLimit_F32(f32In,&f32trMyLowerLimit);

    // output should be 0x60000000 ~ FRAC32(0.75)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    f32Out = GFLIB_LowerLimit(f32In,&f32trMyLowerLimit,F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x60000000 ~ FRAC32(0.75)
    f32Out = GFLIB_LowerLimit(f32In,&f32trMyLowerLimit);
}
```

2.32.4 Function GFLIB_LowerLimit_F16

2.32.4.1 Declaration

```
tFrac16 GFLIB_LowerLimit_F16(tFrac16 f16In, const
GFLIB_LOWERLIMIT_T_F16 *const pParam);
```

2.32.4.2 Arguments

Table 221. GFLIB_LowerLimit_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input value.
const GFLIB_LOWERLIMIT_T_F16 *const	pParam	**input**	Pointer to the limits structure.

2.32.4.3 Return

The input value in case the input value is above the limit, or the lower limit if the input value is below the limit.

2.32.4.4 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;
GFLIB_LOWERLIMIT_T_F16 f16trMyLowerLimit = GFLIB_LOWERLIMIT_DEFAULT_F16;

void main(void)
{
    // lower limit
    f16trMyLowerLimit.f16LowerLimit = FRAC16(0.5);

    // input value = 0.75
    f16In = FRAC16(0.75);

    // output should be 0x6000 ~ FRAC16(0.75)
    f16Out = GFLIB_LowerLimit_F16(f16In,&f16trMyLowerLimit);

    // output should be 0x6000 ~ FRAC16(0.75)
    f16Out = GFLIB_LowerLimit(f16In,&f16trMyLowerLimit,F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // ##############################################################

    // output should be 0x6000 ~ FRAC16(0.75)
    f16Out = GFLIB_LowerLimit(f16In,&f16trMyLowerLimit);
}
```

2.32.5 Function GFLIB_LowerLimit_FLT

2.32.5.1 Declaration

```
tFloat GFLIB_LowerLimit_FLT(tFloat fltIn, const
GFLIB_LOWERLIMIT_T_FLT *const pParam);
```

2.32.5.2 Arguments

Table 222. GFLIB_LowerLimit_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input value.
const GFLIB_LOWERLIMIT_T_FLT *const	pParam	**input**	Pointer to the limits structure.

2.32.5.3 Return

The input value in case the input value is above the limit, or the lower limit if the input value is below the limit.

Note: *The function may raise floating-point exceptions (invalid operation, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.32.5.4 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;
GFLIB_LOWERLIMIT_T_FLT flttrMyLowerLimit = GFLIB_LOWERLIMIT_DEFAULT_FLT;

void main(void)
{
    // lower limit
    flttrMyLowerLimit.fltLowerLimit = 0.5;

    // input value = 0.75
    fltIn = (tFloat) 0.75;

    // output should be 0.75
    fltOut = GFLIB_LowerLimit_FLT(fltIn,&flttrMyLowerLimit);

    // output should be 0.75
    fltOut = GFLIB_LowerLimit(fltIn,&flttrMyLowerLimit,FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // output should be 0.75
    fltOut = GFLIB_LowerLimit(fltIn,&flttrMyLowerLimit);
}
```

2.33 Function GFLIB_Lut1D

This function implements the one-dimensional look-up table.

2.33.1 Description

The GFLIB_Lut1D function performs one dimensional linear interpolation over a 1D table of data. The table must conform to the following constraints:

- The length of the table must be a power of two plus one, i.e. the allowed number of table values is 3, 5, 9, 17, 33, 65, 129, etc.
- The table values must be evenly spaced.

The following interpolation formula is used:

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1} \cdot (x - x_1)$$

Equation GFLIB_Lut1D_Eq1

where:

- y is the interpolated value
- y_1 and y_2 are the ordinate values at, respectively, the beginning and the end of the interpolating interval
- x_1 and x_2 are the abscissa values at, respectively, the beginning and the end of the interpolating interval
- x is the input value provided to the function in the In argument

Regardless of the actual spacing of the table values, the input value In is scaled to always fit in an interval with length one. If the table pointer is chosen to point to the first table element, then the input In falls within the interval [0, 1). If the table pointer is chosen in the middle of the table, then the input falls within the interval [-0.5, 0.5]. Other choices of the table pointer shift the input interval accordingly.

Note: *The input pointer must contain a valid address and the range of the input abscissa values must fall within the range of the interpolating data table otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.33.2 Re-entrancy

The function is re-entrant.

2.33.3 Function GFLIB_Lut1D_F32

2.33.3.1 Declaration

```
tFrac32 GFLIB_Lut1D_F32(tFrac32 f32In, const GFLIB_LUT1D_T_F32
*const pParam);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.33.3.2 Arguments

Table 223. GFLIB_Lut1D_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	The abscissa for which 1D interpolation is performed.
const GFLIB_LUT1D_T_F32 *const	pParam	**input**	Pointer to the parameters structure with parameters of the look-up table function.

2.33.3.3 Return

The interpolated value from the look-up table.

2.33.3.4 Implementation details

The look-up table is provided in the `pf32Table` member of the parameters structure. The table contains ordinate values consecutively over the entire interpolating range. The abscissa values are assumed to be defined implicitly by a single interpolating interval length and a table index, while the interpolating index zero is the table element pointed to by the `pf32Table` parameter. The abscissa value is equal to the multiplication of the interpolating index and the interpolating interval length. For example, let's consider the following interpolating table:

Table 224. GFLIB_Lut1D example table

ordinate (y)	interpolating index	abscissa (x)
-0.5	-1	$-1*(2^{-1})$
0.0	0	$0*(2^{-1})$
0.25	1	$1*(2^{-1})$
0.5	N/A	$2*(2^{-1})$

The Table 224 contains 4 interpolating points (note four rows). The interpolating interval length in this example is equal to 2^{-1} . The `pf32Table` parameter points to the second row, defining also the interpolating index 0. The x-coordinates of the interpolating points are calculated in the right column.

It should be noted that the `pf32Table` pointer does not have to point to the start of the memory area of ordinate values. Therefore, the interpolating index can be positive or negative or, even, does not have to have zero in its range.

A special algorithm is used to make the computation efficient, however, under some additional assumptions, as provided below:

- fIn is in range [a, b], where |a| + |b| = 1
- the values of the interpolated function are in the 32-bit fixed point format
- the length of each interpolating interval is equal to $2^{-uShamOffset}$, where uShamOffset is an integer in the range of 1, 2, ... 29
- the provided abscissa for interpolation is in the 32-bit fixed point format

The algorithm performs the following steps:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

1. Compute the index representing the interval sIntvl, in which the linear interpolation will be performed:

$$sIntvl = \left\lfloor \frac{fIn}{2^{-uShamOffset}} \right\rfloor$$

Equation GFLIB_Lut1D_F32_Eq1

where the `uShamOffset` is the shift amount provided in the parameters structure as the member `pParam->u32ShamOffset`.

2. Compute the abscissa offset within an interpolating interval Δx:

$$\Delta x = \frac{x}{2^{-uShamOffset}} - sIntvl$$

Equation GFLIB_Lut1D_F32_Eq2

where the `uShamOffset` is the shift amount provided in the parameters structure as the member `pParam->u32ShamOffset`.

3. Compute the interpolated value by the linear interpolation between the ordinates read from the table at the start and the end of the computed interval. The computed abscissa offset is used for the linear interpolation.

$$y_1 = pTable[sIntvl]$$
$$y_2 = pTable[sIntvl + 1]$$
$$y = y_1 + (y_2 - y_1) \cdot \Delta x$$

Equation GFLIB_Lut1D_F32_Eq3

where y, y_1 and y_2 are, respectively, the interpolated value, the ordinate at the start of the interpolating interval, the ordinate at the end of the interpolating interval. The pTable is the table containing ordinate values and it is provided as a pointer in the parameters structure `pParam->pf32Table`.

The computations are performed with a 16-bit accuracy. In particular, the 16 least significant bits are ignored in all multiplications.

It should be noted that the input abscissa value can be positive or negative. If it is, positive then the ordinate values are read as in the ordinary data array, that is, at or after the data pointer provided in the parameters structure (`pParam->pf32Table`). However, if it is negative, then the ordinate values are read from the memory with lower address then the `pParam->pfltTable` pointer.

Note: *The function performs a linear interpolation.*

CAUTION: *The function does not check whether the input abscissa value is within the range allowed by the interpolating data table `pParam->pf32Table`. If the computed interval index points to data outside the provided data table, then the interpolation will be computed with invalid data. The range of the input abscissa value depends on the position of the pointer in the interpolating data table. Sum of the absolute values of the lower and upper border values is equal to the FRACT_MAX. For a better understanding, please, see the extended code example.*

2.33.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;
GFLIB_LUT1D_T_F32 trf32MyLut1D = GFLIB_LUT1D_DEFAULT_F32;
tFrac32 pf32Table1D[9] = {FRAC32(0.8),FRAC32(0.1),FRAC32(-0.2),FRAC32(0.7),
                          FRAC32(0.2),FRAC32(-0.3),FRAC32(-0.8),
                          FRAC32(0.91),FRAC32(0.99)};
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
void main(void)
{
    // ##################################################################
    // Pointer is located in the middle of the interpolating data table.
    // ##################################################################
    // setting parameters for Lut1D function
    trf32MyLut1D.u32ShamOffset = (tU32)3;
    trf32MyLut1D.pf32Table = &(pf32Table1D[4]);

    // input vector = -0.5
    f32In = FRAC32(-0.5);

    // output should be 0x66666666 ~ FRAC32(0.8)
    f32Out = GFLIB_Lut1D_F32(f32In,&trf32MyLut1D);

    // output should be 0x66666666 ~ FRAC32(0.8)
    f32Out = GFLIB_Lut1D(f32In,&trf32MyLut1D,F32);

    // available only if 32-bit fractional implementation
    // selected as default
    // output should be 0x66666666 ~ FRAC32(0.8)
    f32Out = GFLIB_Lut1D(f32In,&trf32MyLut1D);

    // ##################################################################
    // Pointer is located at the beginning of the interpolating data table.
    // ##################################################################
    // setting parameters for Lut1D function
    trf32MyLut1D.u32ShamOffset = (tU32)3;
    trf32MyLut1D.pf32Table = &(pf32Table1D[0]);

    // input vector = 0
    f32In = FRAC32(0);

    // output should be 0x66666666 ~ FRAC32(0.8)
    f32Out = GFLIB_Lut1D_F32(f32In,&trf32MyLut1D);

    // output should be 0x66666666 ~ FRAC32(0.8)
    f32Out = GFLIB_Lut1D(f32In,&trf32MyLut1D,F32);

    // available only if 32-bit fractional implementation
    // selected as default
    // output should be 0x66666666 ~ FRAC32(0.8)
    f32Out = GFLIB_Lut1D(f32In,&trf32MyLut1D);

    // ##################################################################
    // Pointer is located at the end of the interpolating data table.
    // ##################################################################
    // setting parameters for Lut1D function
    trf32MyLut1D.u32ShamOffset = (tU32)3;
    trf32MyLut1D.pf32Table = &(pf32Table1D[8]);

    // input vector = -1
    f32In = FRAC32(-1);

    // output should be 0x66666666 ~ FRAC32(0.8)
    f32Out = GFLIB_Lut1D_F32(f32In,&trf32MyLut1D);

    // output should be 0x66666666 ~ FRAC32(0.8)
    f32Out = GFLIB_Lut1D(f32In,&trf32MyLut1D,F32);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // available only if 32-bit fractional implementation
    // selected as default
    // output should be 0x66666666 ~ FRAC32(0.8)
    f32Out = GFLIB_Lut1D(f32In,&trf32MyLut1D);
}
```

2.33.4 Function GFLIB_Lut1D_F16

2.33.4.1 Declaration

```
tFrac16 GFLIB_Lut1D_F16(tFrac16 f16In, const GFLIB_LUT1D_T_F16
*const pParam);
```

2.33.4.2 Arguments

Table 225. GFLIB_Lut1D_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	The abscissa for which 1D interpolation is performed.
const GFLIB_LUT1D_T_F16 *const	pParam	**input**	Pointer to the parameters structure with parameters of the look-up table function.

2.33.4.3 Return

The interpolated value from the look-up table.

2.33.4.4 Implementation details

The look-up table is provided in the `pf16Table` member of the parameters structure. The table contains ordinate values consecutively over the entire interpolating range. The abscissa values are assumed to be defined implicitly by a single interpolating interval length and a table index, while the interpolating index zero is the table element pointed to by the `pf16Table` parameter. The abscissa value is equal to the multiplication of the interpolating index and the interpolating interval length. For example, let's consider the following interpolating table:

Table 226. GFLIB_Lut1D example table

ordinate (y)	interpolating index	abscissa (x)
-0.5	-1	$-1*(2^{-1})$
0.0	0	$0*(2^{-1})$
0.25	1	$1*(2^{-1})$
0.5	N/A	$2*(2^{-1})$

The Table 226 contains 4 interpolating points (note four rows). The interpolating interval length in this example is equal to 2^{-1}. The `pf16Table` parameter points to the second row, defining also the interpolating index 0. The x-coordinates of the interpolating points are calculated in the right column.

It should be noted that the `pf16Table` pointer does not have to point to the start of the memory area of ordinate values. Therefore, the interpolating index can be positive or negative or, even, does not have to have zero in its range.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

A special algorithm is used to make the computation efficient, however, under some additional assumptions, as provided below:

- fIn is in range [a, b], where |a| + |b| = 1
- the values of the interpolated function are in the 16-bit fixed point format
- the length of each interpolating interval is equal to 2^{-n}, where n is an integer in the range of 1, 2, ... 13
- the provided abscissa for interpolation is in the 16-bit fixed point format

The algorithm performs the following steps:

1. Compute the index representing the interval sIntvl, in which the linear interpolation will be performed:

$$sIntvl = \left\lfloor \frac{fIn}{2^{uShamOffset}} \right\rfloor$$

Equation GFLIB_Lut1D_F16_Eq1

   where the `uShamOffset` is the shift amount provided in the parameters structure as the member `pParam->u16ShamOffset`.
2. Compute the abscissa offset within an interpolating interval Δx:

$$\Delta x = \frac{x}{2^{uShamOffset}} - sIntvl$$

Equation GFLIB_Lut1D_F16_Eq2

   where the `uShamOffset` is the shift amount provided in the parameters structure as the member `pParam->u16ShamOffset`.
3. Compute the interpolated value by the linear interpolation between the ordinates read from the table at the start and the end of the computed interval. The computed abscissa offset is used for the linear interpolation.

$$y_1 = pTable[sIntvl]$$
$$y_2 = pTable[sIntvl + 1]$$
$$y = y_1 + (y_2 - y_1) \cdot \Delta x$$

Equation GFLIB_Lut1D_F16_Eq3

   where y, y_1 and y_2 are, respectively, the interpolated value, the ordinate at the start of the interpolating interval, the ordinate at the end of the interpolating interval. The pTable is the table containing ordinate values and it is provided as a pointer in the parameters structure `pParam->pf16Table`.

It should be noted that the input abscissa value can be positive or negative. If it is, positive then the ordinate values are read as in the ordinary data array, that is, at or after the data pointer provided in the parameters structure (`pParam->pf16Table`). However, if it is negative, then the ordinate values are read from the memory, which is located behind the `pParam->pf16Table` pointer.

Note: *The function performs a linear interpolation.*

CAUTION: *The function does not check whether the input abscissa value is within the range allowed by the interpolating data table `pParam->pf16Table`. If the computed interval index points to data outside the provided data table, then the interpolation will be computed with invalid data. The range of the input abscissa value depends on the position of the pointer in the interpolating data table. Sum of the absolute values of the lower and upper border values is equal to the SFRACT_MAX. For a better understanding, please, see the extended code example.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.33.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;
GFLIB_LUT1D_T_F16 trf16MyLut1D = GFLIB_LUT1D_DEFAULT_F16;
tFrac16 pf16Table1D[9] = {FRAC16(0.8),FRAC16(0.1),FRAC16(-0.2),FRAC16(0.7),
                          FRAC16(0.2),FRAC16(-0.3),FRAC16(-0.8),
                          FRAC16(0.91),FRAC16(0.99)};

void main(void)
{
    // ##################################################################
    // Pointer is located in the middle of the interpolating data table.
    // ##################################################################
    // setting parameters for Lut1D function
    trf16MyLut1D.u16ShamOffset = (tU16)3;
    trf16MyLut1D.pf16Table = &(pf16Table1D[4]);

    // input vector = -0.5
    f16In = FRAC16(-0.5);

    // output should be 0x6666 ~ FRAC16(0.8)
    f16Out = GFLIB_Lut1D_F16(f16In,&trf16MyLut1D);

    // output should be 0x6666 ~ FRAC16(0.8)
    f16Out = GFLIB_Lut1D(f16In,&trf16MyLut1D,F16);

    // available only if 16-bit fractional implementation
    // selected as default
    // output should be 0x6666 ~ FRAC16(0.8)
    f16Out = GFLIB_Lut1D(f16In,&trf16MyLut1D);

    // ##################################################################
    // Pointer is located at the beginning of the interpolating data table.
    // ##################################################################
    // setting parameters for Lut1D function
    trf16MyLut1D.u16ShamOffset = (tU16)3;
    trf16MyLut1D.pf16Table = &(pf16Table1D[0]);

    // input vector = 0
    f16In = FRAC16(0);

    // output should be 0x6666 ~ FRAC16(0.8)
    f16Out = GFLIB_Lut1D_F16(f16In,&trf16MyLut1D);

    // output should be 0x6666 ~ FRAC16(0.8)
    f16Out = GFLIB_Lut1D(f16In,&trf16MyLut1D,F16);

    // available only if 16-bit fractional implementation
    // selected as default
    // output should be 0x6666 ~ FRAC16(0.8)
    f16Out = GFLIB_Lut1D(f16In,&trf16MyLut1D);

    // ##################################################################
    // Pointer is located at the end of the interpolating data table.
    // ##################################################################
    // setting parameters for Lut1D function
    trf16MyLut1D.u16ShamOffset = (tU16)3;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    trf16MyLut1D.pf16Table = &(pf16Table1D[8]);

    // input vector = -1
    f16In = FRAC16(-1);

    // output should be 0x6666 ~ FRAC16(0.8)
    f16Out = GFLIB_Lut1D_F16(f16In,&trf16MyLut1D);

    // output should be 0x6666 ~ FRAC16(0.8)
    f16Out = GFLIB_Lut1D(f16In,&trf16MyLut1D,F16);

    // available only if 16-bit fractional implementation
    // selected as default
    // output should be 0x6666 ~ FRAC16(0.8)
    f16Out = GFLIB_Lut1D(f16In,&trf16MyLut1D);
}
```

2.33.5 Function GFLIB_Lut1D_FLT

2.33.5.1 Declaration

```
tFloat GFLIB_Lut1D_FLT(tFloat fltIn, const GFLIB_LUT1D_T_FLT
*const pParam);
```

2.33.5.2 Arguments

Table 227. GFLIB_Lut1D_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	The abscissa for which 1D interpolation is performed.
const GFLIB_LUT1D_T_FLT *const	pParam	**input**	Pointer to the parameters structure with parameters of the look-up table function.

2.33.5.3 Return

The interpolated value from the look-up table.

2.33.5.4 Implementation details

The look-up table is specified by the pointer `pfltTable` in the parameters structure. The table contains ordinate values consecutively over the entire interpolating range. The abscissa values are assumed to be defined implicitly by a single interpolating interval length. The abscissa value is equal to the multiplication of the interpolating index and the interpolating interval length. For example, a table contains 4 interpolating points. The interpolating interval length in this example is equal to 2^{-1}=0.5 .

It should be noted that the `pfltTable` pointer does not have to point to the start of the memory area of ordinate values. Therefore, the interpolating index can be positive or negative or, even, does not have to have zero in its range.

A special algorithm is used to make the computation efficient, however, under some additional assumptions, as provided below:

- x is in range [a, b], where |a| + |b| = 1
- the values of the interpolated function are in the single precision floating point format

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

- the length of each interpolating interval is equal to $2^{-uShamOffset}$, where uShamOffset is an integer in the range of 1, 2, ... 29
- the abscissa provided for interpolation is in the single precision floating point format

The algorithm performs the following steps:

1. Count the number of equidistant interpolating intervals fSegmants:

$$fSegmants = 2^{uShamOffset}$$

Equation GFLIB_Lut1D_FLT_Eq1

   where the `uShamOffset` is the shift amount provided in the parameters structure as the member `pParam->u32ShamOffset`.
2. Compute the interpolating interval, in which the linear interpolation will be performed:

$$sIntvl = \lfloor fSegments \cdot fIn \rfloor$$

Equation GFLIB_Lut1D_FLT_Eq2

   where `sIntvl` is the position of the first ordinate value at the beginning of the interpolating interval, the `fSegments` is the equidistant interpolating interval and the `fIn` is the input value provided to the function as an argument.
3. Compute the interpolated value by the linear interpolation between the ordinates read from the table at the start and the end of the computed interval. The computed abscissa offset is used for the linear interpolation.

$$y_1 = pTable[sIntvl]$$
$$y_2 = pTable[sIntvl + 1]$$
$$\Delta x = (fIn \cdot fSegmants) - sIntvl$$
$$y = y_1 + (y_2 - y_1) \cdot \Delta x$$

Equation GFLIB_Lut1D_FLT_Eq3

   where y, y1 and y2 are, respectively, the interpolated value, the ordinate at the start of the interpolating interval, the ordinate at the end of the interpolating interval. The pTable is the table containing ordinate values and it is provided as a pointer in the parameters structure `pParam->pfltTable`. The `fIn` is the input value provided to the function.

It should be noted that the input value can be positive or negative. If it is positive then the ordinate values are read as in the ordinary data array, that is, at or after the data pointer provided in the parameters structure `pParam->pfltTable`. However, if it is negative, then the ordinate values are read from the memory with lower address then the `pParam->pfltTable` pointer.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

The function performs a linear interpolation.

CAUTION: *The function does not check whether the input abscissa value is within the range allowed by the interpolating data table `pParam->pfltTable`. If the computed interval index points to data outside the provided data table, then the interpolation will be computed with invalid data. The range of the input abscissa value depends on the position of the pointer in the interpolating data table. Sum of the absolute values of the lower and upper border values is equal to the 1. For a better understanding, please, see the extended code example.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.33.5.5 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;
GFLIB_LUT1D_T_FLT trfltMyLut1D = GFLIB_LUT1D_DEFAULT_FLT;
tFloat pfltTable1D[9] = {0.8,0.1,-0.2,0.7,0.2,-0.3,-0.8,0.91,0.99};

void main(void)
{
    // ##################################################################
    // Pointer is located near the middle of the interpolating data table.
    // ##################################################################
    // setting parameters for Lut1D function
    trfltMyLut1D.pfltTable = &(pfltTable1D[5]);
    trfltMyLut1D.u32ShamOffset = (tU32)3;

    // input vector = -0.5
    fltIn = (tFloat)-0.5;

    // output should be 0.1
    fltOut = GFLIB_Lut1D_FLT(fltIn,&trfltMyLut1D);

    // output should be 0.1
    fltOut = GFLIB_Lut1D(fltIn,&trfltMyLut1D,FLT);

    // available only if single precision floating point implementation
    // selected as default
    // output should be 0.1
    fltOut = GFLIB_Lut1D(fltIn,&trfltMyLut1D);

    // ##################################################################
    // Pointer is located at the beginning of the interpolating data table.
    // ##################################################################
    // setting parameters for Lut1D function
    trfltMyLut1D.pfltTable = &(pfltTable1D[0]);
    trfltMyLut1D.u32ShamOffset = (tU32)3;

    // input vector = 0
    fltIn = (tFloat)0;

    // output should be 0.8
    fltOut = GFLIB_Lut1D_FLT(fltIn,&trfltMyLut1D);

    // output should be 0.8
    fltOut = GFLIB_Lut1D(fltIn,&trfltMyLut1D,FLT);

    // available only if single precision floating point implementation
    // selected as default
    // output should be 0.8
    fltOut = GFLIB_Lut1D(fltIn,&trfltMyLut1D);

    // ##################################################################
    // Pointer is located at the end of the interpolating data table.
    // ##################################################################
    // setting parameters for Lut1D function
    trfltMyLut1D.pfltTable = &(pfltTable1D[8]);
    trfltMyLut1D.u32ShamOffset = (tU32)3;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // input vector = -1
    fltIn = (tFloat)-1;

    // output should be 0.8
    fltOut = GFLIB_Lut1D_FLT(fltIn,&trfltMyLut1D);

    // output should be 0.8
    fltOut = GFLIB_Lut1D(fltIn,&trfltMyLut1D,FLT);

    // available only if single precision floating point implementation
    // selected as default
    // output should be 0.8
    fltOut = GFLIB_Lut1D(fltIn,&trfltMyLut1D);
}
```

2.34 Function GFLIB_Lut2D

This function implements the two-dimensional look-up table.

2.34.1 Description

The GFLIB_Lut2D function performs two dimensional linear interpolation over a 2D table of data. The table must conform to the following constraints:

- The width and the height of the table must be a power of two plus one, i.e. the allowed number of rows and columns is 3, 5, 9, 17, 33, 65, 129, etc.
- The table must form a uniform grid, i.e. the rows and columns must be evenly spaced.

The following interpolation formulas are used:

$$f(x, y_1) = f(x_1, y_1) + \frac{f(x_2, y_1) - f(x_1, y_1)}{x_2 - x_1} \cdot (x - x_1)$$

$$f(x, y_2) = f(x_1, y_2) + \frac{f(x_2, y_2) - f(x_1, y_2)}{x_2 - x_1} \cdot (x - x_1)$$

$$f(x, y) = f(x, y_1) + \frac{f(x, y_2) - f(x, y_1)}{y_2 - y_1} \cdot (y - y_1)$$

Equation GFLIB_Lut2D_Eq1

where:

- x, y are the coordinates of the input point provided to the function in the In1 and In2 arguments
- x_1, x_2, y_1, and y_2 are the coordinates of the four table values which are closest to the input point (x, y)
- $f(x_1, y_1)$, $f(x_2, y_1)$, $f(x_1, y_2)$, $f(x_2, y_2)$ are the four table values at the corresponding coordinates
- $f(x, y_1)$ and $f(x, y_2)$ are intermediate interpolated values in one dimension
- f(x, y) is the final interpolated value

The graphical representation of the interpolated area is shown in the following figure.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 91. Interpolated area

Regardless of the actual spacing of the table values, the input coordinates In1 and In2 are scaled to always fit in an interval with length one. If the table pointer is chosen to point to the table element in the top left corner (i.e. x = 0, y = 0), then the inputs In1 and In2 fall within the interval [0, 1). If the table pointer is chosen in the middle of the table, then the inputs fall within the interval [-0.5, 0.5]. Other choices of the table pointer shift the input interval accordingly.

2.34.2 Re-entrancy

The function is re-entrant.

2.34.3 Function GFLIB_Lut2D_F32

2.34.3.1 Declaration

```
tFrac32 GFLIB_Lut2D_F32(tFrac32 f32In1, tFrac32 f32In2, const
GFLIB_LUT2D_T_F32 *const pParam);
```

2.34.3.2 Arguments

Table 228. GFLIB_Lut2D_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In1	**input**	X-coordinate (across columns of the table) of the point for which the 2D interpolation is performed.
tFrac32	f32In2	**input**	Y-coordinate (across rows of the table) of the point for which the 2D interpolation is performed.
const GFLIB_LUT2D_T_F32 *const	pParam	**input**	Pointer to the parameters structure with specification of the two dimensional look-up table function.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.34.3.3 Return

The interpolated value from the look-up table.

2.34.3.4 Implementation details

The look-up table is specified by the pointer `pf32Table` in the parameters structure. The table elements are stored in column-major format. The number of table rows and columns must be a power of two plus one. The total number of elements in the table can be expressed as:

$$table\ size = \left(2^{u32ShamOffset1} + 1\right) * \left(2^{u32ShamOffset2} + 1\right)$$

Equation GFLIB_Lut2D_F32_Eq1

where `u16ShamOffset1` and `u16ShamOffset2` are integers found in the parameters structure. Based on the number of table rows M and the number of columns N, these integers can be calculated as follows:

$$u32ShamOffset1 = \log_2(N - 1)$$
$$u32ShamOffset2 = \log_2(M - 1)$$

Equation GFLIB_Lut2D_F32_Eq2

Inputs `f32In1` and `f32In2` are scaled to always fit in an interval with length one. The zero point in the input interval corresponds to the position in the table pointed to by `pf32Table` relative to the table boundaries. If `pf32Table` points to the middle of the column/row, than the f32In1/f32In2 range is [-0.5, 0.5]. If `pf32Table` points to the first element of the column/row, than the f32In1/f32In2 range is [0, 1). Note that in this case the upper limit cannot be exactly expressed in the tFrac32 type (the largest positive tFrac32 value is 0.9999999995343387).

Let's consider a battery management example use case where we need to determine the maximum allowed charging current depending on the temperature and state of charge. The columns in the table represent the state of charge (SoC) in the range [0%, 80%] and the rows represent temperature (T) in the range [-30°C, 30°C]. The table contains 5 columns and 3 rows. These dimensions imply `u16ShamOffset1` = 2 and `u16ShamOffset2` = 1. The `pf32Table` is chosen to point to the second row and fourth column, i.e. the table value 0.31. The column-major arrangement of the table data in memory means that the `pf32Table` pointer will be offset from the first table entry by 10.

Table 229. GFLIB_Lut2D example table

		SoC	**0%**	**20%**	**40%**	**60%**	**80%**
		f32In1	**-0.75**	**-0.5**	**-0.25**	**0**	**0.25**
T	**f32In2**						
-30°C	**-0.5**		0	0.1	0.2	0.3	0.4
0°C	**0.0**		0.01	0.11	0.21	0.31	0.41
30°C	**0.5**		0.02	0.12	0.22	0.32	0.42

The range of the inputs `f32In1` and `f32In2` depends on the coordinate system origin pointed to by `pf32Table` relative to the table boundaries. The range can be determined by following equations:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$f32In1_{min} = \frac{-OriginIdxX}{2^{u32ShamOffset1}}$$
$$f32In1_{max} = f32In1_{min} + 1$$
$$f32In2_{min} = \frac{-OriginIdxY}{2^{u32ShamOffset2}}$$
$$f32In2_{max} = f32In2_{min} + 1$$

Equation GFLIB_Lut2D_F32_Eq3

where `OriginIdxX` and `OriginIdxY`, respectively, are the x and y coordinates of the origin in the look-up table (i.e. the column/row coordinates of `pf32Table` relative to the first table element). In the example above the `f32In1` must be in the range [-0.75; 0.25] and `f32In2` in the range [-0.5; 0.5].

Note: *The input pointer must contain a valid address and the range of the input values must fall within the range of the look-up table otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

CAUTION: *The function does not check whether the input values are within a range allowed by the look-up table pParam->pf32Table. If the computed interval index points to data outside the provided data table then the interpolation will be computed with invalid data.*

2.34.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32Out;
GFLIB_LUT2D_T_F32 tr32tMyLut2D = GFLIB_LUT2D_DEFAULT_F32;
// Note that GFLIB_Lut2D_F32 interprets the table data
// as column-major, therefore the table below is
// displayed transposed
tFrac32 pf32Table2D[15] = {
FRAC32(0),FRAC32(0.01),FRAC32(0.02),
FRAC32(0.1),FRAC32(0.11),FRAC32(0.12),
FRAC32(0.2),FRAC32(0.21),FRAC32(0.22),
FRAC32(0.3),FRAC32(0.31),FRAC32(0.32),
FRAC32(0.4),FRAC32(0.41),FRAC32(0.42)};

void main(void)
{
    // setting parameters for Lut2D function
    tr32tMyLut2D.u32ShamOffset1 = (tU32)2;
    tr32tMyLut2D.u32ShamOffset2 = (tU32)1;
    tr32tMyLut2D.pf32Table = &(pf32Table2D[10]);

    // input vector
    f32In1 = FRAC32(-0.5);
    f32In2 = FRAC32(-0.5);

    // output should be 0xCCCCCCC ~ FRAC32(0.1)
    f32Out = GFLIB_Lut2D_F32(f32In1,f32In2,&tr32tMyLut2D);

    // output should be 0xCCCCCCC ~ FRAC32(0.1)
    f32Out = GFLIB_Lut2D(f32In1,f32In2,&tr32tMyLut2D,F32);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // available only if 32-bit fractional implementation
    // selected as default
    // output should be 0xCCCCCCC ~ FRAC32(0.1)
    f32Out = GFLIB_Lut2D(f32In1,f32In2,&tr32tMyLut2D);
}
```

2.34.4 Function GFLIB_Lut2D_F16

2.34.4.1 Declaration

```
tFrac16 GFLIB_Lut2D_F16(tFrac16 f16In1, tFrac16 f16In2, const
GFLIB_LUT2D_T_F16 *const pParam);
```

2.34.4.2 Arguments

Table 230. GFLIB_Lut2D_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In1	**input**	X-coordinate (across columns of the table) of the point for which the 2D interpolation is performed.
tFrac16	f16In2	**input**	Y-coordinate (across rows of the table) of the point for which the 2D interpolation is performed.
const GFLIB_LUT2D_T_F16 *const	pParam	**input**	Pointer to the parameters structure with parameters of the two dimensional look-up table function.

2.34.4.3 Return

The interpolated value from the look-up table.

2.34.4.4 Implementation details

The look-up table is specified by the pointer `pf16Table` in the parameters structure. The table elements are stored in column-major format. The number of table rows and columns must be a power of two plus one. The total number of elements in the table can be expressed as:

$$table\ size = \left(2^{u16ShamOffset1} + 1\right) * \left(2^{u16ShamOffset2} + 1\right)$$

Equation GFLIB_Lut2D_F16_Eq1

where `u16ShamOffset1` and `u16ShamOffset2` are integers found in the parameters structure. Based on the number of table rows M and the number of columns N, these integers can be calculated as follows:

$$u16ShamOffset1 = \log_2(N-1)$$

$$u16ShamOffset2 = \log_2(M-1)$$

Equation GFLIB_Lut2D_F16_Eq2

Inputs `f16In1` and `f16In2` are scaled to always fit in an interval with length one. The zero point in the input interval corresponds to the position in the table pointed to by

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

`pf16Table` relative to the table boundaries. If `pf16Table` points to the middle of the column/row, than the f16In1/f16In2 range is [-0.5, 0.5]. If `pf16Table` points to the first element of the column/row, than the f16In1/f16In2 range is [0, 1). Note that in this case the upper limit cannot be exactly expressed in the tFrac16 type (the largest positive tFrac16 value is 0.999969482421875).

Let's consider a battery management example use case where we need to determine the maximum allowed charging current depending on the temperature and state of charge. The columns in the table represent the state of charge (SoC) in the range [0%, 80%] and the rows represent temperature (T) in the range [-30°C, 30°C]. The table contains 5 columns and 3 rows. These dimensions imply `u16ShamOffset1` = 2 and `u16ShamOffset2` = 1. The `pf16Table` is chosen to point to the second row and fourth column, i.e. the table value 0.31. The column-major arrangement of the table data in memory means that the `pf16Table` pointer will be offset from the first table entry by 10.

Table 231. GFLIB_Lut2D example table

		SoC	**0%**	**20%**	**40%**	**60%**	**80%**
		f16In1	**-0.75**	**-0.5**	**-0.25**	**0**	**0.25**
T	**f16In2**						
-30°C	**-0.5**		0	0.1	0.2	0.3	0.4
0°C	**0.0**		0.01	0.11	0.21	0.31	0.41
30°C	**0.5**		0.02	0.12	0.22	0.32	0.42

The range of the inputs `f16In1` and `f16In2` depends on the coordinate system origin pointed to by `pf16Table` relative to the table boundaries. The range can be determined by following equations:

$$f16In1_{min} = \frac{-OriginIdxX}{2^{u16ShamOffset1}}$$
$$f16In1_{max} = f16In1_{min} + 1$$
$$f16In2_{min} = \frac{-OriginIdxY}{2^{u16ShamOffset2}}$$
$$f16In2_{max} = f16In2_{min} + 1$$

Equation GFLIB_Lut2D_F16_Eq3

where `OriginIdxX` and `OriginIdxY`, respectively, are the x and y coordinates of the origin in the look-up table (i.e. the column/row coordinates of `pf16Table` relative to the first table element). In the example above the `f16In1` must be in the range [-0.75; 0.25] and `f16In2` in the range [-0.5; 0.5].

Note: *The input pointer must contain a valid address and the range of the input values must fall within the range of the look-up table otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

CAUTION: *The function does not check whether the input values are within a range allowed by the look-up table pParam->pf16Table. If the computed interval index points to data outside the provided data table then the interpolation will be computed with invalid data.*

2.34.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16In1;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
tFrac16 f16In2;
tFrac16 f16Out;
GFLIB_LUT2D_T_F16 tr16tMyLut2D = GFLIB_LUT2D_DEFAULT_F16;
// Note that GFLIB_Lut2D_F16 interprets the table data
// as column-major, therefore the table below is
// displayed transposed
tFrac16 pf16Table2D[15] = {
FRAC16(0),FRAC16(0.01),FRAC16(0.02),
FRAC16(0.1),FRAC16(0.11),FRAC16(0.12),
FRAC16(0.2),FRAC16(0.21),FRAC16(0.22),
FRAC16(0.3),FRAC16(0.31),FRAC16(0.32),
FRAC16(0.4),FRAC16(0.41),FRAC16(0.42)};

void main(void)
{
    // setting parameters for Lut2D function
    tr16tMyLut2D.u16ShamOffset1 = (tU16)2;
    tr16tMyLut2D.u16ShamOffset2 = (tU16)1;
    tr16tMyLut2D.pf16Table = &(pf16Table2D[10]);

    // input vector
    f16In1 = FRAC16(-0.5);
    f16In2 = FRAC16(-0.5);

    // output should be 0xCCCC ~ FRAC16(0.1)
    f16Out = GFLIB_Lut2D_F16(f16In1,f16In2,&tr16tMyLut2D);

    // output should be 0xCCCC ~ FRAC16(0.1)
    f16Out = GFLIB_Lut2D(f16In1,f16In2,&tr16tMyLut2D,F16);

    // available only if 16-bit fractional implementation
    // selected as default
    // output should be 0xCCCC ~ FRAC16(0.1)
    f16Out = GFLIB_Lut2D(f16In1,f16In2,&tr16tMyLut2D);
}
```

2.34.5 Function GFLIB_Lut2D_FLT

2.34.5.1 Declaration

```
tFloat GFLIB_Lut2D_FLT(tFloat fltIn1, tFloat fltIn2, const
GFLIB_LUT2D_T_FLT *const pParam);
```

2.34.5.2 Arguments

Table 232. GFLIB_Lut2D_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn1	**input**	X-coordinate (across columns of the table) of the point for which the 2D interpolation is performed. Input value is in single precision floating data format.
tFloat	fltIn2	**input**	Y-coordinate (across rows of the table) of the point for which the 2D interpolation is performed. Input value is in single precision floating data format.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 232. GFLIB_Lut2D_FLT arguments...*continued*

Type	Name	Direction	Description
const GFLIB_LUT2D_T_FLT *const	pParam	**input**	Pointer to the parameters structure with parameters of the two dimensional look-up table function.

2.34.5.3 Return

The interpolated value from the look-up table.

2.34.5.4 Implementation details

The look-up table is specified by the pointer `pfltTable` in the parameters structure. The table elements are stored in column-major format. The number of table rows and columns must be a power of two plus one. The total number of elements in the table can be expressed as:

$$N = \left(2^{u32ShamOffset1} + 1\right) * \left(2^{u32ShamOffset2} + 1\right)$$

Equation GFLIB_Lut2D_FLT_Eq1

where `u32ShamOffset1` and `u32ShamOffset2` are integers found in the parameters structure. Based on the number of table rows M and the number of columns N, these integers can be calculated as follows:

$$u32ShamOffset1 = \log_2(N - 1)$$

$$u32ShamOffset2 = \log_2(M - 1)$$

Equation GFLIB_Lut2D_FLT_Eq2

Inputs `fltIn1` and `fltIn2` are scaled to always fit in an interval with length one. The zero point in the input interval corresponds to the position in the table pointed to by `pfltTable` relative to the table boundaries. If `pfltTable` points to the middle of the column/row, than the fltIn1/fltIn2 range is [-0.5, 0.5]. If `pfltTable` points to the first element of the column/row, than the fltIn1/fltIn2 range is [0, 1].

Let's consider a battery management example use case where we need to determine the maximum allowed charging current depending on the temperature and state of charge. The columns in the table represent the state of charge (SoC) in the range [0%, 80%] and the rows represent temperature (T) in the range [-30°C, 30°C]. The table contains 5 columns and 3 rows. These dimensions imply `u32ShamOffset1` = 2 and `u32ShamOffset2` = 1. The `pfltTable` is chosen to point to the second row and fourth column, i.e. the table value 0.31. The column-major arrangement of the table data in memory means that the `pfltTable` pointer will be offset from the first table entry by 10.

Table 233. GFLIB_Lut2D example table

		SoC	**0%**	**20%**	**40%**	**60%**	**80%**
		fltIn1	**-0.75**	**-0.5**	**-0.25**	**0**	**0.25**
T	**fltIn2**						
-30°C	**-0.5**		0	0.1	0.2	0.3	0.4
0°C	**0.0**		0.01	0.11	0.21	0.31	0.41
30°C	**0.5**		0.02	0.12	0.22	0.32	0.42

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The range of the inputs `fltIn1` and `fltIn2` depends on the coordinate system origin pointed to by `pfltTable` relative to the table boundaries. The range can be determined by following equations:

$$fltIn1_{min} = \frac{-OriginIdxX}{2^{u16ShamOffset1}}$$
$$fltIn1_{max} = fltIn1_{min} + 1$$
$$fltIn2_{min} = \frac{-OriginIdxY}{2^{u16ShamOffset2}}$$
$$fltIn2_{max} = fltIn2_{min} + 1$$

Equation GFLIB_Lut2D_FLT_Eq3

where `OriginIdxX` and `OriginIdxY`, respectively, are the x and y coordinates of the origin in the look-up table (i.e. the column/row coordinates of `pfltTable` relative to the first table element). In the example above the `fltIn1` must be in the range [-0.75; 0.25] and `fltIn2` in the range [-0.5; 0.5].

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

The input pointer must contain a valid address and the range of the input values must fall within the range of the look-up table otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).

CAUTION: *The function does not check whether the input values are within the range allowed by the look-up table `pParam->pfltTable`. If the computed interval index points to data outside the provided data table then the interpolation will be computed with invalid data.*

2.34.5.5 Code Example

```
#include "gflib.h"

tFloat fltIn1;
tFloat fltIn2;
tFloat fltOut;
GFLIB_LUT2D_T_FLT trfltMyLut2D = GFLIB_LUT2D_DEFAULT_FLT;
// Note that GFLIB_Lut2D_FLT interprets the table data
// as column-major, therefore the table below is
// displayed transposed
tFloat pfltTable2D[15] = {
0, 0.01, 0.02,
0.1, 0.11, 0.12,
0.2, 0.21, 0.22,
0.3, 0.31, 0.32,
0.4, 0.41, 0.42};

void main(void)
{
    // setting parameters for Lut2D function
    trfltMyLut2D.u32ShamOffset1 = (tU32)2;
    trfltMyLut2D.u32ShamOffset2 = (tU32)1;
    trfltMyLut2D.pfltTable = &(pfltTable2D[10]);

    // input vector
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    fltIn1 = (tFloat)-0.5;
    fltIn2 = (tFloat)-0.5;

    // output should be 0.1
    fltOut = GFLIB_Lut2D_FLT(fltIn1,fltIn2,&trfltMyLut2D);

    // output should be 0.1
    fltOut = GFLIB_Lut2D(fltIn1,fltIn2,&trfltMyLut2D,FLT);

    // available only if single precision floating point implementation
    // selected as default
    // output should be 0.1
    fltOut = GFLIB_Lut2D(fltIn1,fltIn2,&trfltMyLut2D);
}
```

2.35 Function GFLIB_Ramp

The function calculates the up/down ramp with the step increment/decrement defined in the pParam structure.

2.35.1 Description

The GFLIB_Ramp function limits the rate of change of the input signal.

If the absolute value of the desired (input) value is greater than the absolute value of the ramp state, the function adds the RampUp coefficient to the actual output value. The absolute value of the output cannot be greater than the absolute value of the desired value.

If the absolute value of the desired value is lower than the absolute value of the actual ramp state, the function subtracts the RampDown coefficient from the actual output value. The absolute value of the output cannot be lower than the absolute value of the desired value.

Functionality of the implemented ramp algorithm can be explained with use of Figure 92

Figure 92. GFLIB_Ramp functionality

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The upper and lower limits can be found in the limits structure, supplied to the function as a pointer `pParam`.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.35.2 Re-entrancy

The function is re-entrant for a different `pParam`.

2.35.3 Function GFLIB_Ramp_F32

2.35.3.1 Declaration

```
tFrac32 GFLIB_Ramp_F32(tFrac32 f32In, GFLIB_RAMP_T_F32 *const
pParam);
```

2.35.3.2 Arguments

Table 234. GFLIB_Ramp_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument representing the desired output value.
GFLIB_RAMP_T_F32 *const	pParam	**input, output**	Pointer to the ramp parameters structure.

2.35.3.3 Return

The function returns a 32-bit value in format Q1.31, which represents the actual ramp output value. This, in time, is approaching the desired (input) value by step increments defined in the pParam structure.

Note: *All parameters and states used by the function can be reset during declaration using the* GFLIB_RAMP_DEFAULT_F32 *macro.*

2.35.3.4 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;
GFLIB_RAMP_T_F32 f32trMyRamp = GFLIB_RAMP_DEFAULT_F32;

void main(void)
{
    // increment/decrement coefficients
    f32trMyRamp.f32RampUp = FRAC32(0.1);
    f32trMyRamp.f32RampDown = FRAC32(0.03333333);

    // input value = 0.5
    f32In  = FRAC32(0.5);

    // output should be 0x0CCCCCCC ~ FRAC32(0.1)
    f32Out = GFLIB_Ramp_F32(f32In, &f32trMyRamp);

    // clearing of the internal states
    f32trMyRamp.f32State = (tFrac32)0;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // output should be 0x0CCCCCCC ~ FRAC32(0.1)
    f32Out = GFLIB_Ramp(f32In, &f32trMyRamp, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // clearing of the internal states
    f32trMyRamp.f32State = (tFrac32)0;
    // output should be 0x0CCCCCCC ~ FRAC32(0.1)
    f32Out = GFLIB_Ramp(f32In, &f32trMyRamp);
}
```

2.35.4 Function GFLIB_Ramp_F16

2.35.4.1 Declaration

```
tFrac16 GFLIB_Ramp_F16(tFrac16 f16In, GFLIB_RAMP_T_F16 *const
pParam);
```

2.35.4.2 Arguments

Table 235. GFLIB_Ramp_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument representing the desired output value.
GFLIB_RAMP_T_F16 *const	pParam	**input, output**	Pointer to the ramp parameters structure.

2.35.4.3 Return

The function returns a 16-bit value in format Q1.15, which represents the actual ramp output value. This, in time, is approaching the desired (input) value by step increments defined in the pParam structure.

Note: *All parameters and states used by the function can be reset during declaration using the* GFLIB_RAMP_DEFAULT_F16 *macro.*

2.35.4.4 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;
GFLIB_RAMP_T_F16 f16trMyRamp = GFLIB_RAMP_DEFAULT_F16;

void main(void)
{
    // increment/decrement coefficients
    f16trMyRamp.f16RampUp = FRAC16(0.1);
    f16trMyRamp.f16RampDown = FRAC16(0.03333333);

    // input value = 0.5
    f16In  = FRAC16(0.5);

    // output should be 0x0CCC ~ FRAC16(0.1)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    f16Out = GFLIB_Ramp_F16(f16In, &f16trMyRamp);

    // clearing of the internal states
    f16trMyRamp.f16State = (tFrac16)0;
    // output should be 0x0CCC ~ FRAC16(0.1)
    f16Out = GFLIB_Ramp(f16In, &f16trMyRamp, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // clearing of the internal states
    f16trMyRamp.f16State = (tFrac16)0;
    // output should be 0x0CCC ~ FRAC16(0.1)
    f16Out = GFLIB_Ramp(f16In, &f16trMyRamp);
}
```

2.35.5 Function GFLIB_Ramp_FLT

2.35.5.1 Declaration

```
tFloat GFLIB_Ramp_FLT(tFloat fltIn, GFLIB_RAMP_T_FLT *const
pParam);
```

2.35.5.2 Arguments

Table 236. GFLIB_Ramp_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument representing the desired output value. Input value is in single precision floating data format.
GFLIB_RAMP_T_FLT *const	pParam	**input, output**	Pointer to the ramp parameters structure. Arguments of the structure contain single precision floating point values.

2.35.5.3 Return

The function returns a value in single precision floating point format, which represents the actual ramp output. This value can be also described as a Ramp state value increased/decreased by a slope in order to achieve the desired input value.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

All parameters and states used by the function can be reset during declaration using the GFLIB_RAMP_DEFAULT_FLT *macro.*

2.35.5.4 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;
GFLIB_RAMP_T_FLT flttrMyRamp = GFLIB_RAMP_DEFAULT_FLT;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
void main(void)
{
    // increment/decrement coefficients
    flttrMyRamp.fltRampUp = (tFloat)(0.1);
    flttrMyRamp.fltRampDown = (tFloat)(0.03333333);

    // input value = 0.5
    fltIn  = (tFloat)(0.5);

    // output should be 0.1
    fltOut = GFLIB_Ramp_FLT(fltIn, &flttrMyRamp);

    // clearing of the internal states
    flttrMyRamp.fltState = (tFloat)0;
    // output should be 0.1
    fltOut = GFLIB_Ramp(fltIn, &flttrMyRamp, FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // clearing of the internal states
    flttrMyRamp.fltState = (tFloat)0;
    // output should be 0.1
    fltOut = GFLIB_Ramp(fltIn, &flttrMyRamp);
}
```

2.36 Function GFLIB_Sign

This function returns the signum of input value.

2.36.1 Description

The GFLIB_Sign function calculates the sign of the input argument according to the following equation:

$$y_{out} = \begin{cases} 1 & if \quad x_{in} > 0 \\ 0 & if \quad x_{in} = 0 \\ -1 & if \quad x_{in} < 0 \end{cases}$$

Equation GFLIB_Sign_Eq1

where:

- y_{out} is the return value
- x_{in} is the input value provided as the In parameter

2.36.2 Re-entrancy

The function is re-entrant.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.36.3 Function GFLIB_Sign_F32

2.36.3.1 Declaration

```
tFrac32 GFLIB_Sign_F32(tFrac32 f32In);
```

2.36.3.2 Arguments

Table 237. GFLIB_Sign_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument.

2.36.3.3 Return

The function returns the sign of the input argument.

2.36.3.4 Implementation details

If the input value is negative, then the return value will be set to "-1" (0x80000000 hex), if the input value is zero, then the function returns "0" (0x0 hex), otherwise if the input value is greater than zero, the return value will be "1" (0x7fffffff hex).

Note: *The input and the output values are in the 32-bit fixed point fractional data format.*

2.36.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;

void main(void)
{
    // input value = 0.5
    f32In  = FRAC32(0.5);

    // output should be 0x7FFFFFFF ~ FRAC32(1-(2^-31))
    f32Out = GFLIB_Sign_F32(f32In);

    // output should be 0x7FFFFFFF ~ FRAC32(1-(2^-31))
    f32Out = GFLIB_Sign(f32In,F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x7FFFFFFF ~ FRAC32(1-(2^-31))
    f32Out = GFLIB_Sign(f32In);
}
```

2.36.4 Function GFLIB_Sign_F16

2.36.4.1 Declaration

```
tFrac16 GFLIB_Sign_F16(tFrac16 f16In);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.36.4.2 Arguments

Table 238. GFLIB_Sign_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument.

2.36.4.3 Return

The function returns the sign of the input argument.

2.36.4.4 Implementation details

If the input value is negative, then the return value will be set to `"-1"` (0x8000 hex), if the input value is zero, then the function returns `"0"` (0x0 hex), otherwise if the input value is greater than zero, the return value will be `"1"` (0x7fff hex).

Note: *The input and the output values are in the 16-bit fixed point fractional data format.*

2.36.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
{
    // input value = 0.5
    f16In  = FRAC16(0.5);

    // output should be 0x7FFF ~ FRAC16(1-(2^-15))
    f16Out = GFLIB_Sign_F16(f16In);

    // output should be 0x7FFF ~ FRAC16(1-(2^-15))
    f16Out = GFLIB_Sign(f16In,F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x7FFF ~ FRAC16(1-(2^-15))
    f16Out = GFLIB_Sign(f16In);
}
```

2.36.5 Function GFLIB_Sign_FLT

2.36.5.1 Declaration

```
tFloat GFLIB_Sign_FLT(tFloat fltIn);
```

2.36.5.2 Arguments

Table 239. GFLIB_Sign_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.36.5.3 Return

The function returns the sign of the input argument.

Note: *The function may raise floating-point exceptions (invalid operation, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

The input and the output are in single precision floating point data format.

2.36.5.4 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;

void main(void)
{
    // input value = 0.5
    fltIn  = (tFloat)(0.5);

    // output should be 1
    fltOut = GFLIB_Sign_FLT(fltIn);

    // output should be 1
    fltOut = GFLIB_Sign(fltIn,FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be 1
    fltOut = GFLIB_Sign(fltIn);
}
```

2.37 Function GFLIB_Sin

This function implements an approximation of sine function.

2.37.1 Description

The GFLIB_Sin function provides a computational method for calculation of the trigonometric sine function sin(x), using the piece-wise polynomial approximation.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 93. Course of the function GFLIB_Sin

When both sine and cosine of the same argument is needed, use GFLIB_SinCos function instead for better performance.

2.37.2 Re-entrancy

The function is re-entrant.

2.37.3 Function GFLIB_Sin_F32

2.37.3.1 Declaration

```
tFrac32 GFLIB_Sin_F32(tFrac32 f32In, const GFLIB_SIN_T_F32 *const
pParam);
```

2.37.3.2 Arguments

Table 240. GFLIB_Sin_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument is a 32-bit number that contains an angle in radians from interval [- π, π) normalized between [-1, 1).
const GFLIB_SIN_T_F32 *const	pParam	**input**	Pointer to an array of Taylor coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SIN_DEFAULT_F32 symbol.

2.37.3.3 Return

The function returns the value of the sine function of the input argument as a fixed point 32-bit number, normalized between [-1, 1).

2.37.3.4 Implementation details

The input values are scaled from [-π, π) radians to [-1, 1) in order to fit in the available fixed-point fractional range. The function uses a 9th order Taylor polynomial approximation; the default polynomial coefficients are provided in the GFLIB_SIN_DEFAULT_F32 structure.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.37.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32Angle;
tFrac32 f32Output;

void main(void)
{
    // input angle = 0.5 => pi/2
    f32Angle  = FRAC32(0.5);

    // output should be 0x7FFFFFFF
    f32Output = GFLIB_Sin_F32(f32Angle, GFLIB_SIN_DEFAULT_F32);

    // output should be 0x7FFFFFFF
    f32Output = GFLIB_Sin(f32Angle, GFLIB_SIN_DEFAULT_F32, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x7FFFFFFF
    f32Output = GFLIB_Sin(f32Angle);
}
```

2.37.4 Function GFLIB_Sin_F16

2.37.4.1 Declaration

```
tFrac16 GFLIB_Sin_F16(tFrac16 f16In, const GFLIB_SIN_T_F16 *const pParam);
```

2.37.4.2 Arguments

Table 241. GFLIB_Sin_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument is a 16-bit number that contains an angle in radians from interval [- π, π) normalized between [-1, 1).
const GFLIB_SIN_T_F16 *const	pParam	**input**	Pointer to an array of Taylor coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SIN_DEFAULT_F16 symbol.

2.37.4.3 Return

The function returns the value of the sine function of the input argument as a fixed point 16-bit number, normalized between [-1, 1).

2.37.4.4 Implementation details

The input values are scaled from [-π, π) radians to [-1, 1) in order to fit in the available fixed-point fractional range. The function uses a 7th order Taylor

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

polynomial approximation; the default polynomial coefficients are provided in the GFLIB_SIN_DEFAULT_F16 structure.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.37.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16Angle;
tFrac16 f16Output;

void main(void)
{
    // input angle = 0.5 => pi/2
    f16Angle  = FRAC16(0.5);

    // output should be 0x7FFF
    f16Output = GFLIB_Sin_F16(f16Angle, GFLIB_SIN_DEFAULT_F16);

    // output should be 0x7FFF
    f16Output = GFLIB_Sin(f16Angle, GFLIB_SIN_DEFAULT_F16, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x7FFF
    f16Output = GFLIB_Sin(f16Angle);
}
```

2.37.5 Function GFLIB_Sin_FLT

2.37.5.1 Declaration

```
tFloat GFLIB_Sin_FLT(tFloat fltIn, const GFLIB_SIN_T_FLT *const
pParam);
```

2.37.5.2 Arguments

Table 242. GFLIB_Sin_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument is a single precision floating point number that contains an angle in radians from interval [- π, π].
const GFLIB_SIN_T_FLT *const	pParam	**input**	Pointer to an array of approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SIN_DEFAULT_FLT symbol.

2.37.5.3 Return

The function returns the value of the sine function of the input argument as a single precision floating point number.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.37.5.4 Implementation details

The function uses a 7th order minimax polynomial approximation; the default polynomial coefficients are provided in the GFLIB_SIN_DEFAULT_FLT structure.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.

2.37.5.5 Code Example

```
#include "gflib.h"

tFloat fltAngle;
tFloat fltOutput;

void main(void)
{
    // input angle = 1.5707963 => pi/2
    fltAngle  = (tFloat)(1.5707963);

    // output should be 1
    fltOutput = GFLIB_Sin_FLT(fltAngle, GFLIB_SIN_DEFAULT_FLT);

    // output should be 1
    fltOutput = GFLIB_Sin(fltAngle, GFLIB_SIN_DEFAULT_FLT, FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be 1
    fltOutput = GFLIB_Sin(fltAngle);
}
```

2.38 Function GFLIB_SinCos

This function implements polynomial approximation of the sine and cosine function.

2.38.1 Description

The function GFLIB_SinCos calculates the trigonometric sine and cosine function of the same argument using a polynomial approximation. GFLIB_SinCos performs faster than the combination of GFLIB_Sin and GFLIB_Cos.

2.38.2 Re-entrancy

The function is re-entrant.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.38.3 Function GFLIB_SinCos_F32

2.38.3.1 Declaration

```
void GFLIB_SinCos_F32(tFrac32 f32In, SWLIBS_2Syst_F32 *pOut,
const GFLIB_SINCOS_T_F32 *const pParam);
```

2.38.3.2 Arguments

Table 243. GFLIB_SinCos_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument is a 32-bit number that contains an angle in radians from interval [- π, π) normalized between [-1, 1).
SWLIBS_2Syst_F32 *	pOut	**output**	Pointer to the structure where the values of the sine and cosine of the input angle are stored. The function returns the sine and cosine of the input argument as a fixed point 32-bit number, normalized between [-1, 1). The *sine* of input angle is returned in first item of the structure and the *cosine* of input angle is returned in second item of the structure.
const GFLIB_SINCOS_T_F32 *const	pParam	**input**	Pointer to an array of Taylor coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SINCOS_DEFAULT_F32 symbol.

2.38.3.3 Return

void

2.38.3.4 Implementation details

The input values are scaled from [-π, π) radians to [-1, 1) in order to fit in the available fixed-point fractional range. The function uses a 9th order Taylor polynomial approximation; the default polynomial coefficients are provided in the GFLIB_SINCOS_DEFAULT_F32 structure.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.38.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32Angle;
SWLIBS_2Syst_F32 pf32Output;

void main(void)
{
    // input angle = 0.5 => pi/2
    f32Angle  = FRAC32(0.5);

    // output should be:
    // pf32Output.f32Arg1 ~ sin(f32Angle) = 0x7FFF0000
    // pf32Output.f32Arg2 ~ cos(f32Angle) = 0x00000000
    GFLIB_SinCos_F32(f32Angle, &pf32Output, GFLIB_SINCOS_DEFAULT_F32);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // output should be:
    // pf32Output.f32Arg1 ~ sin(f32Angle) = 0x7FFF0000
    // pf32Output.f32Arg2 ~ cos(f32Angle) = 0x00000000
    GFLIB_SinCos(f32Angle, &pf32Output, GFLIB_SINCOS_DEFAULT_F32, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be:
    // pf32Output.f32Arg1 ~ sin(f32Angle) = 0x7FFF0000
    // pf32Output.f32Arg2 ~ cos(f32Angle) = 0x00000000
    GFLIB_SinCos(f32Angle, &pf32Output);
}
```

2.38.4 Function GFLIB_SinCos_F16

2.38.4.1 Declaration

```
void GFLIB_SinCos_F16(tFrac16 f16In, SWLIBS_2Syst_F16 *pOut,
const GFLIB_SINCOS_T_F16 *const pParam);
```

2.38.4.2 Arguments

Table 244. GFLIB_SinCos_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument is a 16-bit number that contains an angle in radians from interval [- π, π) normalized between [-1, 1).
SWLIBS_2Syst_F16 *	pOut	**output**	Pointer to the structure where the values of the sine and cosine of the input angle are stored. The function returns the sine and cosine of the input argument as a fixed point 16-bit number, normalized between [-1, 1). The *sine* of input angle is returned in first item of the structure and the *cosine* of input angle is returned in second item of the structure.
const GFLIB_SINCOS_T_F16 *const	pParam	**input**	Pointer to an array of Taylor coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SINCOS_DEFAULT_F16 symbol.

2.38.4.3 Return

void

2.38.4.4 Implementation details

The input values are scaled from [-π, π) radians to [-1, 1) in order to fit in the available fixed-point fractional range. The function uses a 7th order Taylor polynomial approximation; the default polynomial coefficients are provided in the GFLIB_SINCOS_DEFAULT_F16 structure.

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).

2.38.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16Angle;
SWLIBS_2Syst_F16 pf16Output;

void main(void)
{
    // input angle = 0.5 => pi/2
    f16Angle  = FRAC16(0.5);

    // output should be:
    // pf16Output.f16Arg1 ~ sin(f16Angle) = 0x7FFF
    // pf16Output.f16Arg2 ~ cos(f16Angle) = 0x0000
    GFLIB_SinCos_F16(f16Angle, &pf16Output, GFLIB_SINCOS_DEFAULT_F16);

    // output should be:
    // pf16Output.f16Arg1 ~ sin(f16Angle) = 0x7FFF
    // pf16Output.f16Arg2 ~ cos(f16Angle) = 0x0000
    GFLIB_SinCos(f16Angle, &pf16Output, GFLIB_SINCOS_DEFAULT_F16, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be:
    // pf16Output.f16Arg1 ~ sin(f16Angle) = 0x7FFF
    // pf16Output.f16Arg2 ~ cos(f16Angle) = 0x0000
    GFLIB_SinCos(f16Angle, &pf16Output);
}
```

2.38.5 Function GFLIB_SinCos_FLT

2.38.5.1 Declaration

```
void GFLIB_SinCos_FLT(tFloat fltIn, SWLIBS_2Syst_FLT *pOut, const
GFLIB_SINCOS_T_FLT *const pParam);
```

2.38.5.2 Arguments

Table 245. GFLIB_SinCos_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument is a single precision floating point number that contains an angle in radians from interval [- π, π].

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 245. GFLIB_SinCos_FLT arguments...*continued*

Type	Name	Direction	Description
SWLIBS_2Syst_FLT *	pOut	**output**	Pointer to the structure where the values of the sine and cosine of the input angle are stored. The function returns the sine and cosine of the input argument as a single precision floating point number. The *sine* of input angle is returned in first item of the structure and the *cosine* of input angle is returned in second item of the structure.
const GFLIB_SINCOS_T_FLT *const	pParam	**input**	Pointer to an array of approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SINCOS_DEFAULT_FLT symbol.

2.38.5.3 Return

void

2.38.5.4 Implementation details

The function uses a 7th order minimax polynomial approximation; the default polynomial coefficients are provided in the GFLIB_SINCOS_DEFAULT_FLT structure.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.

2.38.5.5 Code Example

```
#include "gflib.h"

tFloat fltAngle;
SWLIBS_2Syst_FLT pfltOutput;

void main(void)
{
    // input angle = 1.5707963 => pi/2
    fltAngle  = (tFloat)(1.5707963);

    // output should be:
    // pfltOutput.fltArg1 ~ sin(fltAngle) = 1.0f
    // pfltOutput.fltArg2 ~ cos(fltAngle) = 0f
    GFLIB_SinCos_FLT(fltAngle, &pfltOutput, GFLIB_SINCOS_DEFAULT_FLT);

    // output should be:
    // pfltOutput.fltArg1 ~ sin(fltAngle) = 1.0f
    // pfltOutput.fltArg2 ~ cos(fltAngle) = 0f
    GFLIB_SinCos(fltAngle, &pfltOutput, GFLIB_SINCOS_DEFAULT_FLT, FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // output should be:
    // pfltOutput.fltArg1 ~ sin(fltAngle) = 1.0f
    // pfltOutput.fltArg2 ~ cos(fltAngle) = 0f
    GFLIB_SinCos(fltAngle &pfltOutput);
}
```

2.39 Function GFLIB_Sqrt

This function returns the square root of input value.

2.39.1 Description

The GFLIB_Sqrt function calculates the square root of the input value.

2.39.2 Re-entrancy

The function is re-entrant.

2.39.3 Function GFLIB_Sqrt_F32

2.39.3.1 Declaration

```
tFrac32 GFLIB_Sqrt_F32(tFrac32 f32In);
```

2.39.3.2 Arguments

Table 246. GFLIB_Sqrt_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	The input value.

2.39.3.3 Return

The function returns the square root of the input value. The return value is within the [0, 1) fraction range.

2.39.3.4 Implementation details

The function uses a floating-point square root instruction with appropriate input/output conversions.

Note: *The function may raise floating-point exceptions (invalid operation, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

The valid input range is [0, 1). Negative inputs will yield undefined results. This function is implemented as inline assembly and thus is not ANSI-C compliant.

2.39.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
void main(void)
{
    // input value = 0.5
    f32In = FRAC32(0.5);

    // output should be 0x5A820000 ~ FRAC32(0.70710678)
    f32Out = GFLIB_Sqrt_F32(f32In);

    // output should be 0x5A820000 ~ FRAC32(0.70710678)
    f32Out = GFLIB_Sqrt(f32In,F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x5A820000 ~ FRAC32(0.70710678)
    f32Out = GFLIB_Sqrt(f32In);
}
```

2.39.4 Function GFLIB_Sqrt_F16

2.39.4.1 Declaration

```
tFrac16 GFLIB_Sqrt_F16(tFrac16 f16In);
```

2.39.4.2 Arguments

Table 247. GFLIB_Sqrt_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	The input value.

2.39.4.3 Return

The function returns the square root of the input value. The return value is within the [0, 1) fraction range.

2.39.4.4 Implementation details

The function uses a floating-point square root instruction with appropriate input/output conversions.

Note: *The function may raise floating-point exceptions (invalid operation, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

The valid input range is [0, 1). Negative inputs will yield undefined results. This function is implemented as inline assembly and thus is not ANSI-C compliant.

2.39.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
{
    // input value = 0.5
    f16In = FRAC16(0.5);

    // output should be 0x5A82 ~ FRAC16(0.70710678)
    f16Out = GFLIB_Sqrt_F16(f16In);

    // output should be 0x5A82 ~ FRAC16(0.70710678)
    f16Out = GFLIB_Sqrt(f16In,F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x5A82 ~ FRAC16(0.70710678)
    f16Out = GFLIB_Sqrt(f16In);
}
```

2.39.5 Function GFLIB_Sqrt_FLT

2.39.5.1 Declaration

```
tFloat GFLIB_Sqrt_FLT(tFloat fltIn);
```

2.39.5.2 Arguments

Table 248. GFLIB_Sqrt_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	The input value.

2.39.5.3 Return

The function returns the square root of the input value. The return value is in single precision floating point format.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

Valid inputs are nonnegative numbers. This function is implemented using inline assembly and thus is not ANSI-C compliant.

2.39.5.4 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;

void main(void)
{
    // input value = 0.5
    fltIn = (tFloat) 0.5;

    // output should be 0.70710678
    fltOut = GFLIB_Sqrt_FLT(fltIn);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // output should be 0.70710678
    fltOut = GFLIB_Sqrt(fltIn,FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be 0.70710678
    fltOut = GFLIB_Sqrt(fltIn);
}
```

2.40 Function GFLIB_Tan

This function implements an approximation of tangent function.

2.40.1 Description

The GFLIB_Tan function provides a computational method for calculation of the trigonometric tangent function tan(x), using the piece-wise polynomial approximation.

2.40.2 Re-entrancy

The function is re-entrant.

2.40.3 Function GFLIB_Tan_F32

2.40.3.1 Declaration

```
tFrac32 GFLIB_Tan_F32(tFrac32 f32In, const GFLIB_TAN_T_F32 *const
pParam);
```

2.40.3.2 Arguments

Table 249. GFLIB_Tan_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument is a 32-bit number that contains an angle in radians from interval [- π, π) normalized between [-1, 1).
const GFLIB_TAN_T_F32 *const	pParam	**input**	Pointer to an array of Taylor coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_TAN_DEFAULT_F32 symbol.

2.40.3.3 Return

The function returns tan(π · f32In) as a fixed point 32-bit number, normalized between [-1, 1).

2.40.3.4 Implementation details

The input values are scaled from [-π, π) radians to [-1, 1) in order to fit in the available fixed-point fractional range. Output values are saturated. The function uses a piece-wise 4th order polynomial approximation.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 94. Course of the function GFLIB_Tan_F32

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.40.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32Angle;
tFrac32 f32Output;

void main(void)
{
    // input angle = 0.25 => pi/4
    f32Angle  = FRAC32(0.25);

    // output should be 0x7FFFFFFF = 1
    f32Output = GFLIB_Tan_F32(f32Angle, GFLIB_TAN_DEFAULT_F32);

    // output should be 0x7FFFFFFF = 1
    f32Output = GFLIB_Tan(f32Angle, GFLIB_TAN_DEFAULT_F32, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x7FFFFFFF = 1
    f32Output = GFLIB_Tan(f32Angle);
}
```

2.40.4 Function GFLIB_Tan_F16

2.40.4.1 Declaration

```
tFrac16 GFLIB_Tan_F16(tFrac16 f16In, const GFLIB_TAN_T_F16 *const
pParam);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.40.4.2 Arguments

Table 250. GFLIB_Tan_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument is a 16-bit number that contains an angle in radians from interval [- π, π) normalized between [-1, 1).
const GFLIB_TAN_T_F16 *const	pParam	**input**	Pointer to an array of Taylor coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SIN_DEFAULT_F16 symbol.

2.40.4.3 Return

The function returns tan(π · f16In) as a fixed point 16-bit number, normalized between [-1, 1).

2.40.4.4 Implementation details

The input values are scaled from [-π, π) radians to [-1, 1) in order to fit in the available fixed-point fractional range. Output values are saturated. The function uses a piece-wise 4th order polynomial approximation; the default polynomial coefficients are provided in the GFLIB_TAN_DEFAULT_F16 structure.

Figure 95. Course of the function GFLIB_Tan_F16

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.40.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16Angle;
tFrac16 f16Output;

void main(void)
{
    // input angle = 0.25 => pi/4
    f16Angle  = FRAC16(0.25);

    // output should be 0x7FFF = 1
    f16Output = GFLIB_Tan_F16(f16Angle, GFLIB_TAN_DEFAULT_F16);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // output should be 0x7FFF = 1
    f16Output = GFLIB_Tan(f16Angle, GFLIB_TAN_DEFAULT_F16, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x7FFF = 1
    f16Output = GFLIB_Tan(f16Angle);
}
```

2.40.5 Function GFLIB_Tan_FLT

2.40.5.1 Declaration

```
tFloat GFLIB_Tan_FLT(tFloat fltIn, const GFLIB_TAN_T_FLT *const
pParam);
```

2.40.5.2 Arguments

Table 251. GFLIB_Tan_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument is a single precision floating point number that contains an angle in radians from interval (- π, π).
const GFLIB_TAN_T_FLT *const	pParam	**input**	Pointer to an array of approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_TAN_DEFAULT_FLT symbol.

2.40.5.3 Return

The function returns tan(fltIn) as a single precision floating point number.

2.40.5.4 Implementation details

The function uses a rational polynomial approximation. The default polynomial coefficients are provided in the GFLIB_TAN_DEFAULT_FLT structure.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 96. Course of the function GFLIB_Tan_FLT

Note: *The function may raise floating-point exceptions (invalid operation, division by zero, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).

2.40.5.5 Code Example

```
#include "gflib.h"

tFloat fltAngle;
tFloat fltOutput;

void main(void)
{
    // input angle = pi/4
    fltAngle  = (tFloat)0.78539816;

    // output should be 1
    fltOutput = GFLIB_Tan_FLT(fltAngle, GFLIB_TAN_DEFAULT_FLT);

    // output should be 1
    fltOutput = GFLIB_Tan(fltAngle, GFLIB_TAN_DEFAULT_FLT, FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be 1
    fltOutput = GFLIB_Tan(fltAngle);
}
```

2.41 Function GFLIB_UpperLimit

This function tests whether the input value is below the upper limit.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.41.1 Description

The GFLIB_UpperLimit function tests whether the input value is below the upper limit. If so, the input value will be returned. Otherwise, if the input value is above the upper limit, the upper limit will be returned.

The upper limit UpperLimit can be found in the parameters structure, supplied to the function as a pointer pParam.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.41.2 Re-entrancy

The function is re-entrant.

2.41.3 Function GFLIB_UpperLimit_F32

2.41.3.1 Declaration

```
tFrac32 GFLIB_UpperLimit_F32(tFrac32 f32In, const
GFLIB_UPPERLIMIT_T_F32 *const pParam);
```

2.41.3.2 Arguments

Table 252. GFLIB_UpperLimit_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input value.
const GFLIB_UPPERLIMIT_T_F32 *const	pParam	**input**	Pointer to the limits structure.

2.41.3.3 Return

The input value in case the input value is below the limit, or the upper limit if the input value is above the limit.

2.41.3.4 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;
GFLIB_UPPERLIMIT_T_F32 f32trMyUpperLimit = GFLIB_UPPERLIMIT_DEFAULT_F32;

void main(void)
{
    // upper limit
    f32trMyUpperLimit.f32UpperLimit = FRAC32(0.5);
    // input value = 0.75
    f32In = FRAC32(0.75);

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = GFLIB_UpperLimit_F32(f32In,&f32trMyUpperLimit);

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = GFLIB_UpperLimit(f32In,&f32trMyUpperLimit,F32);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = GFLIB_UpperLimit(f32In,&f32trMyUpperLimit);
}
```

2.41.4 Function GFLIB_UpperLimit_F16

2.41.4.1 Declaration

```
tFrac16 GFLIB_UpperLimit_F16(tFrac16 f16In, const
GFLIB_UPPERLIMIT_T_F16 *const pParam);
```

2.41.4.2 Arguments

Table 253. GFLIB_UpperLimit_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input value.
const GFLIB_UPPERLIMIT_T_F16 *const	pParam	**input**	Pointer to the limits structure.

2.41.4.3 Return

The input value in case the input value is below the limit, or the upper limit if the input value is above the limit.

2.41.4.4 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;
GFLIB_UPPERLIMIT_T_F16 f16trMyUpperLimit = GFLIB_UPPERLIMIT_DEFAULT_F16;

void main(void)
{
    // upper limit
    f16trMyUpperLimit.f16UpperLimit = FRAC16(0.5);
    // input value = 0.75
    f16In = FRAC16(0.75);

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = GFLIB_UpperLimit_F16(f16In,&f16trMyUpperLimit);

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = GFLIB_UpperLimit(f16In,&f16trMyUpperLimit,F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = GFLIB_UpperLimit(f16In,&f16trMyUpperLimit);
}
```

2.41.5 Function GFLIB_UpperLimit_FLT

2.41.5.1 Declaration

```
tFloat GFLIB_UpperLimit_FLT(tFloat fltIn, const
GFLIB_UPPERLIMIT_T_FLT *const pParam);
```

2.41.5.2 Arguments

Table 254. GFLIB_UpperLimit_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input value.
const GFLIB_UPPERLIMIT_T_FLT *const	pParam	**input**	Pointer to the limits structure.

2.41.5.3 Return

The input value in case the input value is below the limit, or the upper limit if the input value is above the limit.

Note: *The function may raise floating-point exceptions (invalid operation, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.41.5.4 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;
GFLIB_UPPERLIMIT_T_FLT flttrMyUpperLimit = GFLIB_UPPERLIMIT_DEFAULT_FLT;

void main(void)
{
    // upper limit
    flttrMyUpperLimit.fltUpperLimit = 0.5;
    // input value = 0.75
    fltIn = (tFloat) 0.75;

    // output should be 0.5
    fltOut = GFLIB_UpperLimit_FLT(fltIn,&flttrMyUpperLimit);

    // output should be 0.5
    fltOut = GFLIB_UpperLimit(fltIn,&flttrMyUpperLimit,FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be 0.5
    fltOut = GFLIB_UpperLimit(fltIn,&flttrMyUpperLimit);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
}
```

2.42 Function GFLIB_VectorLimit

This function limits the magnitude of the input vector.

2.42.1 Description

The GFLIB_VectorLimit function limits the magnitude of the input vector, keeping its direction unchanged. Limitation is performed as follows:

$$y_{out} = \begin{cases} \frac{y_{in}}{\sqrt{x_{in}^2+y_{in}^2}} \cdot L & if \quad \sqrt{x_{in}^2+y_{in}^2} > L \\ y_{in} & if \quad \sqrt{x_{in}^2+y_{in}^2} \le L \end{cases}$$

$$x_{out} = \begin{cases} \frac{x_{in}}{\sqrt{x_{in}^2+y_{in}^2}} \cdot L & if \quad \sqrt{x_{in}^2+y_{in}^2} > L \\ x_{in} & if \quad \sqrt{x_{in}^2+y_{in}^2} \le L \end{cases}$$

Equation GFLIB_VectorLimit_Eq1

Where:

- x_{in}, y_{in} and x_{out}, y_{out} are the co-ordinates of the input and output vector, respectively
- L is the maximum magnitude of the vector

The input vector co-ordinates are defined by the structure pointed to by the pIn parameter, and the output vector co-ordinates be found in the structure pointed by the pOut parameter. The maximum vector magnitude is defined in the parameters structure pointed to by the pParam function parameter.

A graphical interpretation of the function can be seen in the figure below.

Figure 97. Graphical interpretation of the GFLIB_VectorLimit function.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

If an actual limitation occurs, the function will return TRUE, otherwise the FALSE will be returned.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.42.2 Re-entrancy

The function is re-entrant.

2.42.3 Function GFLIB_VectorLimit_F32

2.42.3.1 Declaration

```
tBool GFLIB_VectorLimit_F32(SWLIBS_2Syst_F32 *const pOut, const
SWLIBS_2Syst_F32 *const pIn, const GFLIB_VECTORLIMIT_T_F32 *const
pParam);
```

2.42.3.2 Arguments

Table 255. GFLIB_VectorLimit_F32 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F32 *const	pOut	**output**	Pointer to the structure of the limited output vector.
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure of the input vector.
const GFLIB_VECTORLIMIT_T_F32 *const	pParam	**input**	Pointer to the parameters structure.

2.42.3.3 Return

The function will return TRUE if the input vector is being limited or FALSE otherwise.

2.42.3.4 Implementation details

The output vector will be computed as zero if the input vector magnitude is lower than 2^{-15}, regardless of the set maximum magnitude of the input vector. The function returns TRUE in this case.

CAUTION: *The 16 least significant bits of `pParam->f32Limit` are ignored. This means that the defined magnitude must be equal to or greater than 2^{-15}, otherwise the result is undefined.*

Note: *The function calls the AMMCLIB square root routine GFLIB_Sqrt_F32. The function uses a floating-point square root instruction.*

The function may raise floating-point exceptions (invalid operation, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.

2.42.3.5 Code Example

```
#include "gflib.h"

SWLIBS_2Syst_F32 f32pIn;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
SWLIBS_2Syst_F32 f32pOut;
GFLIB_VECTORLIMIT_T_F32 f32trMyVectorLimit = GFLIB_VECTORLIMIT_DEFAULT_F32;
tBool bLim;

void main(void)
{
    // desired magnitude of the input vector
    f32trMyVectorLimit.f32Limit = FRAC32(0.25);
    // input vector
    f32pIn.f32Arg1 = FRAC32(0.25);
    f32pIn.f32Arg2 = FRAC32(0.25);

    // output should be:
    // bLim = TRUE;
    // f32pOut.f32Arg1 = 0x16A08000 ~ FRAC32(0.17677)
    // f32pOut.f32Arg2 = 0x16A08000 ~ FRAC32(0.17677)
    bLim = GFLIB_VectorLimit_F32(&f32pOut,&f32pIn,&f32trMyVectorLimit);

    // output should be:
    // bLim = TRUE;
    // f32pOut.f32Arg1 = 0x16A08000 ~ FRAC32(0.17677)
    // f32pOut.f32Arg2 = 0x16A08000 ~ FRAC32(0.17677)
    bLim = GFLIB_VectorLimit(&f32pOut,&f32pIn,&f32trMyVectorLimit,F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be:
    // bLim = TRUE;
    // f32pOut.f32Arg1 = 0x16A08000 ~ FRAC32(0.17677)
    // f32pOut.f32Arg2 = 0x16A08000 ~ FRAC32(0.17677)
    bLim = GFLIB_VectorLimit(&f32pOut,&f32pIn,&f32trMyVectorLimit);
}
```

2.42.4 Function GFLIB_VectorLimit_F16

2.42.4.1 Declaration

```
tBool GFLIB_VectorLimit_F16(SWLIBS_2Syst_F16 *const pOut, const
SWLIBS_2Syst_F16 *const pIn, const GFLIB_VECTORLIMIT_T_F16 *const
pParam);
```

2.42.4.2 Arguments

Table 256. GFLIB_VectorLimit_F16 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *const	pOut	**output**	Pointer to the structure of the limited output vector.
const SWLIBS_2Syst_F16 *const	pIn	**input**	Pointer to the structure of the input vector.
const GFLIB_VECTORLIMIT_T_F16 *const	pParam	**input**	Pointer to the parameters structure.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.42.4.3 Return

The function will return TRUE if the input vector is being limited, or FALSE otherwise.

CAUTION: *The maximum vector magnitude in the parameters structure, the `pParam->f16Limit`, must be positive and equal to or greater than `"0"`, otherwise the result is undefined. The function does not check for the valid range of the parameter `pParam->f16Limit`.*

Note: *The function calls the AMMCLIB square root routine GFLIB_Sqrt_F16. The function uses a floating-point square root instruction.*

The function may raise floating-point exceptions (invalid operation, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.

2.42.4.4 Code Example

```
#include "gflib.h"

SWLIBS_2Syst_F16 f16pIn;
SWLIBS_2Syst_F16 f16pOut;
GFLIB_VECTORLIMIT_T_F16 f16trMyVectorLimit = GFLIB_VECTORLIMIT_DEFAULT_F16;
tBool bLim;

void main(void)
{
    // desired magnitude of the input vector
    f16trMyVectorLimit.f16Limit = FRAC16(0.25);
    // input vector
    f16pIn.f16Arg1 = FRAC16(0.25);
    f16pIn.f16Arg2 = FRAC16(0.25);

    // output should be:
    // bLim = TRUE;
    // f16pOut.f16Arg1 = 0x16A0 ~ FRAC16(0.17677)
    // f16pOut.f16Arg2 = 0x16A0 ~ FRAC16(0.17677)
    bLim = GFLIB_VectorLimit_F16(&f16pOut,&f16pIn,&f16trMyVectorLimit);

    // output should be:
    // bLim = TRUE;
    // f16pOut.f16Arg1 = 0x16A0 ~ FRAC16(0.17677)
    // f16pOut.f16Arg2 = 0x16A0 ~ FRAC16(0.17677)
    bLim = GFLIB_VectorLimit(&f16pOut,&f16pIn,&f16trMyVectorLimit,F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be:
    // bLim = TRUE;
    // f16pOut.f16Arg1 = 0x16A0 ~ FRAC16(0.17677)
    // f16pOut.f16Arg2 = 0x16A0 ~ FRAC16(0.17677)
    bLim = GFLIB_VectorLimit(&f16pOut,&f16pIn,&f16trMyVectorLimit);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.42.5 Function GFLIB_VectorLimit_FLT

2.42.5.1 Declaration

```
tBool GFLIB_VectorLimit_FLT(SWLIBS_2Syst_FLT *const pOut, const
SWLIBS_2Syst_FLT *const pIn, const GFLIB_VECTORLIMIT_T_FLT *const
pParam);
```

2.42.5.2 Arguments

Table 257. GFLIB_VectorLimit_FLT arguments

Type	Name	Direction	Description
SWLIBS_2Syst_FLT *const	pOut	**output**	Pointer to the structure of the limited output vector.
const SWLIBS_2Syst_FLT *const	pIn	**input**	Pointer to the structure of the input vector.
const GFLIB_VECTORLIMIT_T_FLT *const	pParam	**input**	Pointer to the parameters structure.

2.42.5.3 Return

The function will return TRUE if the input vector is being limited, or FALSE otherwise.

Note: *The function may raise floating-point exceptions (invalid operation, division by zero, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

The function calls the AMMCLIB square root routine GFLIB_Sqrt_FLT*.*

CAUTION: *The maximum vector magnitude in the parameters structure, the* `pParam->fltLimit`*, must be positive and equal to or greater than* `"0"`*, otherwise the result is undefined. The function does not check for the valid range of the parameter* `pParam->fltLimit`*.*

2.42.5.4 Code Example

```
#include "gflib.h"

SWLIBS_2Syst_FLT fltpIn;
SWLIBS_2Syst_FLT fltpOut;
GFLIB_VECTORLIMIT_T_FLT flttrMyVectorLimit = GFLIB_VECTORLIMIT_DEFAULT_FLT;
tBool bLim;

void main(void)
{
    // desired magnitude of the input vector
    flttrMyVectorLimit.fltLimit = (tFloat)0.25;
    // input vector
    fltpIn.fltArg1 = (tFloat)0.25;
    fltpIn.fltArg2 = (tFloat)0.25;

    // output should be:
    // bLim = TRUE;
    // fltpOut.fltArg1 = 0.17677
    // fltpOut.fltArg2 = 0.17677
    bLim = GFLIB_VectorLimit_FLT(&fltpOut,&fpltIn,&flttrMyVectorLimit);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // output should be:
    // bLim = TRUE;
    // fltpOut.fltArg1 = 0.17677
    // fltpOut.fltArg2 = 0.17677
    bLim = GFLIB_VectorLimit(&fltpOut,&fltpIn,&flttrMyVectorLimit,FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be:
    // bLim = TRUE;
    // fltpOut.fltArg1 = 0.17677
    // fltpOut.fltArg2 = 0.17677
    bLim = GFLIB_VectorLimit(&fltpOut,&fltpIn,&flttrMyVectorLimit);
}
```

2.43 Function GFLIB_VLog10_FLT

This function calculates a base-10 logarithm of absolute values of an array.

2.43.1 Declaration

```
void GFLIB_VLog10_FLT(tFloat *pInOut, tU32 u32N, const
GFLIB_VLOG10_T_FLT *const pParam);
```

2.43.2 Arguments

Table 258. GFLIB_VLog10_FLT arguments

Type	Name	Direction	Description
tFloat *	pInOut	**input, output**	Pointer to the floating-point input/output data array. Must be aligned to a double-word boundary.
tU32	u32N	**input**	Array length. Must be divisible by 8 (i.e. 8, 16, 24, ...).
const GFLIB_VLOG10_T_FLT *const	pParam	**input**	Pointer to an array of approximation coefficients.

2.43.3 Description

The function calculates a base-10 logarithm of the absolute values of the input array. Results overwrite the inputs (in-place processing). The default approximating polynomial coefficients are provided in the GFLIB_VLOG10_DEFAULT_FLT structure.

This is a vectorized variant of function GFLIB_Log10_FLT. The vectorized form provides processing speed benefit for large arrays of input values.

Note: *The function may raise floating-point exceptions (overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.43.4 Code Example

```
#include "gflib.h"

tFloat pValues[8] = {1.0F, 1.0F, 1.0F, 1.0F, 1.0F, 1.0F, 1.0F, 1.0F};

void main(void)
{
  // all outputs should be 0
  GFLIB_VLog10_FLT(pValues, 8U, GFLIB_VLOG10_DEFAULT_FLT);

  // all outputs should be 0
  GFLIB_VLog10(pValues, 8U, GFLIB_VLOG10_DEFAULT_FLT, FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // all outputs should be 0
  GFLIB_VLog10(pValues, 8U);
}
```

2.43.5 Reentrancy

The function is reentrant for a different `pInOut`.

2.44 Function GFLIB_VMin

This function finds the minimum of a vector.

2.44.1 Description

The function GFLIB_VMin finds the minimum in a vector of values and returns the index of that value. If there are several equal minimums, the index of the last one is returned.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.44.2 Re-entrancy

The function is re-entrant.

2.44.3 Function GFLIB_VMin_F32

2.44.3.1 Declaration

```
tU32 GFLIB_VMin_F32(const tFrac32 *pIn, tU32 u32N);
```

2.44.3.2 Arguments

Table 259. GFLIB_VMin_F32 arguments

Type	Name	Direction	Description
const tFrac32 *	pIn	**input**	Pointer to an array of 32-bit fixed-point signed input values.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 259. GFLIB_VMin_F32 arguments...*continued*

Type	Name	Direction	Description
tU32	u32N	**input**	Length of the input array.

2.44.3.3 Return

The index of the minimum of the input array.

2.44.3.4 Code Example

```
#include "gflib.h"

void main(void)
{
    tFrac32 pInputArray[5] = {6, 3, 1, 4, 8};
    tFrac32 f32MinValue;

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f32MinValue = pInputArray[GFLIB_VMin_F32(pInputArray, 5)];

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f32MinValue = pInputArray[GFLIB_VMin(pInputArray, 5, F32)];

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 32-bit fractional implementation is selected as default.
    f32MinValue = pInputArray[GFLIB_VMin(pInputArray, 5)];
}
```

2.44.4 Function GFLIB_VMin_F16

2.44.4.1 Declaration

```
tU16 GFLIB_VMin_F16(const tFrac16 *pIn, tU16 u16N);
```

2.44.4.2 Arguments

Table 260. GFLIB_VMin_F16 arguments

Type	Name	Direction	Description
const tFrac16 *	pIn	**input**	Pointer to an array of 16-bit fixed-point signed input values.
tU16	u16N	**input**	Length of the input array.

2.44.4.3 Return

The index of the minimum of the input array.

Note: *The library offers faster inlined version of this function for the input array lengths of 4 to 16, see GFLIB_VMin4_F16, GFLIB_VMin5_F16, etc.*

2.44.4.4 Code Example

```
#include "gflib.h"
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
void main(void)
{
    tFrac16 pInputArray[5] = {6, 3, 1, 4, 8};
    tFrac16 f16MinValue;

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    f16MinValue = pInputArray[GFLIB_VMin_F16(pInputArray, 5)];

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    f16MinValue = pInputArray[GFLIB_VMin(pInputArray, 5, F16)];

    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if 16-bit fractional implementation is selected as default.
    f16MinValue = pInputArray[GFLIB_VMin(pInputArray, 5)];
}
```

2.44.5 Function GFLIB_VMin_FLT

2.44.5.1 Declaration

```
tU32 GFLIB_VMin_FLT(const tFloat *pIn, tU32 u32N);
```

2.44.5.2 Arguments

Table 261. GFLIB_VMin_FLT arguments

Type	Name	Direction	Description
const tFloat *	pIn	**input**	Pointer to an array of single precision floating-point input values.
tU32	u32N	**input**	Length of the input array.

2.44.5.3 Return

The index of the minimum of the input array.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.44.5.4 Code Example

```
#include "gflib.h"

void main(void)
{
    tFloat pInputArray[5] = {6.0f, 3.0f, 1.0f, 4.0f, 8.0f};
    tFloat fltMinValue;

    // Alternative 1: API call with postfix
    // (only one alternative shall be used).
    fltMinValue = pInputArray[GFLIB_VMin_FLT(pInputArray, 5)];

    // Alternative 2: API call with implementation parameter
    // (only one alternative shall be used).
    fltMinValue = pInputArray[GFLIB_VMin(pInputArray, 5, FLT)];
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // Alternative 3: API call with global configuration of implementation
    // (only one alternative shall be used). This alternative is available
    // only if floating point implementation is selected as default.
    fltMinValue = pInputArray[GFLIB_VMin(pInputArray, 5)];
}
```

2.44.6 Function GFLIB_VMin4_F16

2.44.6.1 Declaration

```
INLINE tU16 GFLIB_VMin4_F16(const tFrac16 *pIn);
```

2.44.6.2 Arguments

Table 262. GFLIB_VMin4_F16 arguments

Type	Name	Direction	Description
const tFrac16 *	pIn	**input**	Pointer to an array of 16-bit fixed-point signed input values. The length of the array must be 4 elements.

2.44.6.3 Return

The function returns the index of the smallest element of the input array.

2.44.6.4 Implementation details

The function finds the minimum in a 4-element vector of values and returns the index of that value. If there are several equal minimums, the index of an arbitrary one of them is returned. This function is faster than GFLIB_VMin_F16.

2.44.6.5 Code Example

```
#include "gflib.h"

void main(void)
{
    tFrac16 pInputArray[4] = {6, 3, 1, 4};
    tFrac16 f16MinValue;

    f16MinValue = pInputArray[GFLIB_VMin4_F16(pInputArray)];
}
```

2.44.7 Function GFLIB_VMin5_F16

2.44.7.1 Declaration

```
INLINE tU16 GFLIB_VMin5_F16(const tFrac16 *pIn);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.44.7.2 Arguments

Table 263. GFLIB_VMin5_F16 arguments

Type	Name	Direction	Description
const tFrac16 *	pIn	**input**	Pointer to an array of 16-bit fixed-point signed input values. The length of the array must be 5 elements.

2.44.7.3 Return

The function returns the index of the smallest element of the input array.

2.44.7.4 Implementation details

The function finds the minimum in a 5-element vector of values and returns the index of that value. If there are several equal minimums, the index of an arbitrary one of them is returned. There must be two consecutive readable bytes in the memory following the last element of the input array. This function is faster than GFLIB_VMin_F16.

2.44.7.5 Code Example

```
#include "gflib.h"

void main(void)
{
    tFrac16 pInputArray[5] = {6, 3, 1, 4, 5};
    tFrac16 f16MinValue;

    f16MinValue = pInputArray[GFLIB_VMin5_F16(pInputArray)];
}
```

2.44.8 Function GFLIB_VMin6_F16

2.44.8.1 Declaration

```
INLINE tU16 GFLIB_VMin6_F16(const tFrac16 *pIn);
```

2.44.8.2 Arguments

Table 264. GFLIB_VMin6_F16 arguments

Type	Name	Direction	Description
const tFrac16 *	pIn	**input**	Pointer to an array of 16-bit fixed-point signed input values. The length of the array must be 6 elements.

2.44.8.3 Return

The function returns the index of the smallest element of the input array.

2.44.8.4 Implementation details

The function finds the minimum in a 6-element vector of values and returns the index of that value. If there are several equal minimums, the index of an arbitrary one of them is returned. This function is faster than GFLIB_VMin_F16.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.44.8.5 Code Example

```
#include "gflib.h"

void main(void)
{
    tFrac16 pInputArray[6] = {6, 3, 1, 4, 5, 6};
    tFrac16 f16MinValue;

    f16MinValue = pInputArray[GFLIB_VMin6_F16(pInputArray)];
}
```

2.44.9 Function GFLIB_VMin7_F16

2.44.9.1 Declaration

```
INLINE tU16 GFLIB_VMin7_F16(const tFrac16 *pIn);
```

2.44.9.2 Arguments

Table 265. GFLIB_VMin7_F16 arguments

Type	Name	Direction	Description
const tFrac16 *	pIn	**input**	Pointer to an array of 16-bit fixed-point signed input values. The length of the array must be 7 elements.

2.44.9.3 Return

The function returns the index of the smallest element of the input array.

2.44.9.4 Implementation details

The function finds the minimum in a 7-element vector of values and returns the index of that value. If there are several equal minimums, the index of an arbitrary one of them is returned. This function is faster than GFLIB_VMin_F16.

2.44.9.5 Code Example

```
#include "gflib.h"

void main(void)
{
    tFrac16 pInputArray[7] = {6, 3, 1, 4, 5, 6, 7};
    tFrac16 f16MinValue;

    f16MinValue = pInputArray[GFLIB_VMin7_F16(pInputArray)];
}
```

2.44.10 Function GFLIB_VMin8_F16

2.44.10.1 Declaration

```
INLINE tU16 GFLIB_VMin8_F16(const tFrac16 *pIn);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.44.10.2 Arguments

Table 266. GFLIB_VMin8_F16 arguments

Type	Name	Direction	Description
const tFrac16 *	pIn	**input**	Pointer to an array of 16-bit fixed-point signed input values. The length of the array must be 8 elements.

2.44.10.3 Return

The function returns the index of the smallest element of the input array.

2.44.10.4 Implementation details

The function finds the minimum in a 8-element vector of values and returns the index of that value. If there are several equal minimums, the index of an arbitrary one of them is returned. This function is faster than GFLIB_VMin_F16.

2.44.10.5 Code Example

```
#include "gflib.h"

void main(void)
{
    tFrac16 pInputArray[8] = {6, 3, 1, 4, 5, 6, 7, 8};
    tFrac16 f16MinValue;

    f16MinValue = pInputArray[GFLIB_VMin8_F16(pInputArray)];
}
```

2.44.11 Function GFLIB_VMin9_F16

2.44.11.1 Declaration

```
INLINE tU16 GFLIB_VMin9_F16(const tFrac16 *pIn);
```

2.44.11.2 Arguments

Table 267. GFLIB_VMin9_F16 arguments

Type	Name	Direction	Description
const tFrac16 *	pIn	**input**	Pointer to an array of 16-bit fixed-point signed input values. The length of the array must be 9 elements.

2.44.11.3 Return

The function returns the index of the smallest element of the input array.

2.44.11.4 Implementation details

The function finds the minimum in a 9-element vector of values and returns the index of that value. If there are several equal minimums, the index of an arbitrary one of them is returned. There must be two consecutive readable bytes in the memory following the last element of the input array. This function is faster than GFLIB_VMin_F16.

© NXP B.V. 2023. All rights reserved.

2.44.11.5 Code Example

```
#include "gflib.h"

void main(void)
{
    tFrac16 pInputArray[9] = {6, 3, 1, 4, 5, 6, 7, 8, 9};
    tFrac16 f16MinValue;

    f16MinValue = pInputArray[GFLIB_VMin9_F16(pInputArray)];
}
```

2.44.12 Function GFLIB_VMin10_F16

2.44.12.1 Declaration

```
INLINE tU16 GFLIB_VMin10_F16(const tFrac16 *pIn);
```

2.44.12.2 Arguments

Table 268. GFLIB_VMin10_F16 arguments

Type	Name	Direction	Description
const tFrac16 *	pIn	**input**	Pointer to an array of 16-bit fixed-point signed input values. The length of the array must be 10 elements.

2.44.12.3 Return

The function returns the index of the smallest element of the input array.

2.44.12.4 Implementation details

The function finds the minimum in a 10-element vector of values and returns the index of that value. If there are several equal minimums, the index of an arbitrary one of them is returned. This function is faster than GFLIB_VMin_F16.

2.44.12.5 Code Example

```
#include "gflib.h"

void main(void)
{
    tFrac16 pInputArray[10] = {6, 3, 1, 4, 5, 6, 7, 8, 9, 10};
    tFrac16 f16MinValue;

    f16MinValue = pInputArray[GFLIB_VMin10_F16(pInputArray)];
}
```

2.44.13 Function GFLIB_VMin11_F16

2.44.13.1 Declaration

```
INLINE tU16 GFLIB_VMin11_F16(const tFrac16 *pIn);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.44.13.2 Arguments

Table 269. GFLIB_VMin11_F16 arguments

Type	Name	Direction	Description
const tFrac16 *	pIn	**input**	Pointer to an array of 16-bit fixed-point signed input values. The length of the array must be 11 elements.

2.44.13.3 Return

The function returns the index of the smallest element of the input array.

2.44.13.4 Implementation details

The function finds the minimum in a 11-element vector of values and returns the index of that value. If there are several equal minimums, the index of an arbitrary one of them is returned. There must be two consecutive readable bytes in the memory following the last element of the input array. This function is faster than GFLIB_VMin_F16.

2.44.13.5 Code Example

```
#include "gflib.h"

void main(void)
{
    tFrac16 pInputArray[11] = {6, 3, 1, 4, 5, 6, 7, 8, 9, 10, 11};
    tFrac16 f16MinValue;

    f16MinValue = pInputArray[GFLIB_VMin11_F16(pInputArray)];
}
```

2.44.14 Function GFLIB_VMin12_F16

2.44.14.1 Declaration

```
INLINE tU16 GFLIB_VMin12_F16(const tFrac16 *pIn);
```

2.44.14.2 Arguments

Table 270. GFLIB_VMin12_F16 arguments

Type	Name	Direction	Description
const tFrac16 *	pIn	**input**	Pointer to an array of 16-bit fixed-point signed input values. The length of the array must be 12 elements.

2.44.14.3 Return

The function returns the index of the smallest element of the input array.

2.44.14.4 Implementation details

The function finds the minimum in a 12-element vector of values and returns the index of that value. If there are several equal minimums, the index of an arbitrary one of them is returned. This function is faster than GFLIB_VMin_F16.

© NXP B.V. 2023. All rights reserved.

2.44.14.5 Code Example

```
#include "gflib.h"

void main(void)
{
    tFrac16 pInputArray[12] = {6, 3, 1, 4, 5, 6, 7, 8, 9,
      10, 11, 12};
    tFrac16 f16MinValue;

    f16MinValue = pInputArray[GFLIB_VMin12_F16(pInputArray)];
}
```

2.44.15 Function GFLIB_VMin13_F16

2.44.15.1 Declaration

```
INLINE tU16 GFLIB_VMin13_F16(const tFrac16 *pIn);
```

2.44.15.2 Arguments

Table 271. GFLIB_VMin13_F16 arguments

Type	Name	Direction	Description
const tFrac16 *	pIn	**input**	Pointer to an array of 16-bit fixed-point signed input values. The length of the array must be 13 elements.

2.44.15.3 Return

The function returns the index of the smallest element of the input array.

2.44.15.4 Implementation details

The function finds the minimum in a 13-element vector of values and returns the index of that value. If there are several equal minimums, the index of an arbitrary one of them is returned. There must be two consecutive readable bytes in the memory following the last element of the input array. This function is faster than GFLIB_VMin_F16.

2.44.15.5 Code Example

```
#include "gflib.h"

void main(void)
{
    tFrac16 pInputArray[13] = {6, 3, 1, 4, 5, 6, 7, 8, 9,
      10, 11, 12, 13};
    tFrac16 f16MinValue;

    f16MinValue = pInputArray[GFLIB_VMin13_F16(pInputArray)];
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.44.16 Function GFLIB_VMin14_F16

2.44.16.1 Declaration

```
INLINE tU16 GFLIB_VMin14_F16(const tFrac16 *pIn);
```

2.44.16.2 Arguments

Table 272. GFLIB_VMin14_F16 arguments

Type	Name	Direction	Description
const tFrac16 *	pIn	**input**	Pointer to an array of 16-bit fixed-point signed input values. The length of the array must be 14 elements.

2.44.16.3 Return

The function returns the index of the smallest element of the input array.

2.44.16.4 Implementation details

The function finds the minimum in a 14-element vector of values and returns the index of that value. If there are several equal minimums, the index of an arbitrary one of them is returned. This function is faster than GFLIB_VMin_F16.

2.44.16.5 Code Example

```
#include "gflib.h"

void main(void)
{
    tFrac16 pInputArray[14] = {6, 3, 1, 4, 5, 6, 7, 8, 9,
      10, 11, 12, 13, 14};
    tFrac16 f16MinValue;

    f16MinValue = pInputArray[GFLIB_VMin14_F16(pInputArray)];
}
```

2.44.17 Function GFLIB_VMin15_F16

2.44.17.1 Declaration

```
INLINE tU16 GFLIB_VMin15_F16(const tFrac16 *pIn);
```

2.44.17.2 Arguments

Table 273. GFLIB_VMin15_F16 arguments

Type	Name	Direction	Description
const tFrac16 *	pIn	**input**	Pointer to an array of 16-bit fixed-point signed input values. The length of the array must be 15 elements.

2.44.17.3 Return

The function returns the index of the smallest element of the input array.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.44.17.4 Implementation details

The function finds the minimum in a 15-element vector of values and returns the index of that value. If there are several equal minimums, the index of an arbitrary one of them is returned. There must be two consecutive readable bytes in the memory following the last element of the input array. This function is faster than GFLIB_VMin_F16.

2.44.17.5 Code Example

```
#include "gflib.h"

void main(void)
{
    tFrac16 pInputArray[15] = {6, 3, 1, 4, 5, 6, 7, 8, 9,
      10, 11, 12, 13, 14, 15};
    tFrac16 f16MinValue;

    f16MinValue = pInputArray[GFLIB_VMin15_F16(pInputArray)];
}
```

2.44.18 Function GFLIB_VMin16_F16

2.44.18.1 Declaration

```
INLINE tU16 GFLIB_VMin16_F16(const tFrac16 *pIn);
```

2.44.18.2 Arguments

Table 274. GFLIB_VMin16_F16 arguments

Type	Name	Direction	Description
const tFrac16 *	pIn	**input**	Pointer to an array of 16-bit fixed-point signed input values. The length of the array must be 16 elements.

2.44.18.3 Return

The function returns the index of the smallest element of the input array.

2.44.18.4 Implementation details

The function finds the minimum in a 16-element vector of values and returns the index of that value. If there are several equal minimums, the index of an arbitrary one of them is returned. This function is faster than GFLIB_VMin_F16.

2.44.18.5 Code Example

```
#include "gflib.h"

void main(void)
{
    tFrac16 pInputArray[16] = {6, 3, 1, 4, 5, 6, 7, 8, 9,
      10, 11, 12, 13, 14, 15, 16};
    tFrac16 f16MinValue;

    f16MinValue = pInputArray[GFLIB_VMin16_F16(pInputArray)];
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
}
```

2.45 Function GMCLIB_BetaProjection

The function implements the Beta-projection transformation.

2.45.1 Description

The Beta-projection transformation is used to transform values from the three-phase (A-B-C) coordinate system to the two-phase (α-β) orthogonal coordinate system, according to the following equations:

$$\begin{bmatrix} u_\alpha \\ u_\beta \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ 0 & \frac{\sqrt{3}}{3} & -\frac{\sqrt{3}}{3} \end{bmatrix} \cdot \begin{bmatrix} u_A \\ u_B \\ u_C \end{bmatrix}$$

Equation GMCLIB_BetaProjection_Eq1

where it is assumed that the axis A (axis of the first phase) and the axis α are in the same direction. Compared to an ordinary Clarke transformation, the patent-pending Beta-projection provides superior immunity to nonlinear distortions such as DC offset, limitation, and certain higher harmonics. Beta-projection is an essential tool for the reconstruction of the angle of the induced voltage vector in a free-wheeling permanent magnet synchronous motor.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.45.2 Re-entrancy

The function is re-entrant.

2.45.3 Function GMCLIB_BetaProjection_F32

2.45.3.1 Declaration

```
void GMCLIB_BetaProjection_F32(SWLIBS_2Syst_F32 *const pOut,
const SWLIBS_3Syst_F32 *const pIn);
```

2.45.3.2 Arguments

Table 275. GMCLIB_BetaProjection_F32 arguments

Type	Name	Direction	Description
const SWLIBS_3Syst_F32 *const	pIn	**input**	Pointer to the structure containing data of the three-phase stationary system (f32A-f32B-f32C). Arguments of the structure contain fixed point 32-bit values.
SWLIBS_2Syst_F32 *const	pOut	**output**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α-β). Arguments of the structure contain fixed point 32-bit values.

Note: *The inputs and the outputs are normalized to fit in the range [-1, 1).*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.45.3.3 Code Example

```
#include "gmclib.h"

SWLIBS_3Syst_F32 f32trAbc;
SWLIBS_2Syst_F32 f32trAlBe;

void main(void)
{
    // input phase A ~ sin(45) ~ 0.707106781
    // input phase B ~ sin(45 + 120) ~ 0.258819045
    // input phase C ~ sin(45 - 120) ~ -0.965925826
    f32trAbc.f32Arg1 = FRAC32(0.707106781);
    f32trAbc.f32Arg2 = FRAC32(0.258819045);
    f32trAbc.f32Arg3 = FRAC32(-0.965925826);

    // output should be
    // f32trAlBe.f32Arg1 = 0x5A82799A ~ FRAC32(0.707106781)
    // f32trAlBe.f32Arg2 = 0x5A827999 ~ FRAC32(0.707106781)
    GMCLIB_BetaProjection_F32(&f32trAlBe,&f32trAbc);

    // output should be
    // f32trAlBe.f32Arg1 = 0x5A82799A ~ FRAC32(0.707106781)
    // f32trAlBe.f32Arg2 = 0x5A827999 ~ FRAC32(0.707106781)
    GMCLIB_BetaProjection(&f32trAlBe,&f32trAbc,F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be
    // f32trAlBe.f32Arg1 = 0x5A82799A ~ FRAC32(0.707106781)
    // f32trAlBe.f32Arg2 = 0x5A827999 ~ FRAC32(0.707106781)
    GMCLIB_BetaProjection(&f32trAlBe,&f32trAbc);
}
```

2.45.4 Function GMCLIB_BetaProjection_F16

2.45.4.1 Declaration

```
void GMCLIB_BetaProjection_F16(SWLIBS_2Syst_F16 *const pOut,
const SWLIBS_3Syst_F16 *const pIn);
```

2.45.4.2 Arguments

Table 276. GMCLIB_BetaProjection_F16 arguments

Type	Name	Direction	Description
const SWLIBS_3Syst_F16 *const	pIn	**input**	Pointer to the structure containing data of the three-phase stationary system (f16A-f16B-f16C). Arguments of the structure contain fixed point 16-bit values.
SWLIBS_2Syst_F16 *const	pOut	**output**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α-β). Arguments of the structure contain fixed point 16-bit values.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Note: *The inputs and the outputs are normalized to fit in the range [-1, 1).*

2.45.4.3 Code Example

```
#include "gmclib.h"

SWLIBS_3Syst_F16 f16trAbc;
SWLIBS_2Syst_F16 f16trAlBe;

void main(void)
{
    // input phase A ~ sin(45) ~ 0.707106781
    // input phase B ~ sin(45 + 120) ~ 0.258819045
    // input phase C ~ sin(45 - 120) ~ -0.965925826
    f16trAbc.f16Arg1 = FRAC16(0.707106781);
    f16trAbc.f16Arg2 = FRAC16(0.258819045);
    f16trAbc.f16Arg3 = FRAC16(-0.965925826);

    // output should be
    // f16trAlBe.f16Arg1 = 0x5A82 ~ FRAC16(0.707106781)
    // f16trAlBe.f16Arg2 = 0x5A83 ~ FRAC16(0.707106781)
    GMCLIB_BetaProjection_F16(&f16trAlBe,&f16trAbc);

    // output should be
    // f16trAlBe.f16Arg1 = 0x5A82 ~ FRAC16(0.707106781)
    // f16trAlBe.f16Arg2 = 0x5A83 ~ FRAC16(0.707106781)
    GMCLIB_BetaProjection(&f16trAlBe,&f16trAbc,F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be
    // f16trAlBe.f16Arg1 = 0x5A82 ~ FRAC16(0.707106781)
    // f16trAlBe.f16Arg2 = 0x5A83 ~ FRAC16(0.707106781)
    GMCLIB_BetaProjection(&f16trAlBe,&f16trAbc);
}
```

2.45.5 Function GMCLIB_BetaProjection_FLT

2.45.5.1 Declaration

```
void GMCLIB_BetaProjection_FLT(SWLIBS_2Syst_FLT *const pOut,
const SWLIBS_3Syst_FLT *const pIn);
```

2.45.5.2 Arguments

Table 277. GMCLIB_BetaProjection_FLT arguments

Type	Name	Direction	Description
const SWLIBS_3Syst_FLT *const	pIn	**input**	Pointer to the structure containing data of the three-phase stationary system (fltA-fltB-fltC). Arguments of the structure contain single precision floating point values.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 277. GMCLIB_BetaProjection_FLT arguments...*continued*

Type	Name	Direction	Description
SWLIBS_2Syst_FLT *const	pOut	**output**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α-β). Arguments of the structure contain single precision floating point values.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.45.5.3 Code Example

```
#include "gmclib.h"

SWLIBS_3Syst_FLT flttrAbc;
SWLIBS_2Syst_FLT flttrAlBe;

void main(void)
{
    // input phase A ~ sin(45) ~ 0.707106781
    // input phase B ~ sin(45 + 120) ~ 0.258819045
    // input phase C ~ sin(45 - 120) ~ -0.965925826
    flttrAbc.fltArg1 = (tFloat)0.707106781;
    flttrAbc.fltArg2 = (tFloat)0.258819045;
    flttrAbc.fltArg3 = (tFloat)-0.965925826;

    // output should be
    // flttrAlBe.fltArg1 = 0.70710677
    // flttrAlBe.fltArg2 = 0.70710671
    GMCLIB_BetaProjection_FLT(&flttrAlBe,&flttrAbc);

    // output should be
    // flttrAlBe.fltArg1 = 0.70710677
    // flttrAlBe.fltArg2 = 0.70710671
    GMCLIB_BetaProjection(&flttrAlBe,&flttrAbc,FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be
    // flttrAlBe.fltArg1 = 0.70710677
    // flttrAlBe.fltArg2 = 0.70710671
    GMCLIB_BetaProjection(&flttrAlBe,&flttrAbc);
}
```

2.46 Function GMCLIB_BetaProjection3Ph

The function implements the three-phase Beta-projection transformation.

2.46.1 Description

The three-phase Beta-projection transforms the input three-phase (A-B-C) coordinate system into an equivalent output three-phase coordinate system while eliminating certain

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

types of non-linear distortion (e.g. DC offset, 3rd harmonics, limitation). The patent-pending algorithm calculates the following equations:

$$\begin{bmatrix} u_{Aout} \\ u_{Bout} \\ u_{Cout} \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} \end{bmatrix} \cdot \begin{bmatrix} u_A \\ u_B \\ u_C \end{bmatrix}$$

Equation GMCLIB_BetaProjection3Ph_Eq1

Refer to GMCLIB_BetaProjection function for a related Beta-projection transform that produces a two-phase output.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.46.2 Re-entrancy

The function is re-entrant.

2.46.3 Function GMCLIB_BetaProjection3Ph_F32

2.46.3.1 Declaration

```
void GMCLIB_BetaProjection3Ph_F32(SWLIBS_3Syst_F32 *const pOut,
const SWLIBS_3Syst_F32 *const pIn);
```

2.46.3.2 Arguments

Table 278. GMCLIB_BetaProjection3Ph_F32 arguments

Type	Name	Direction	Description
const SWLIBS_3Syst_F32 *const	pIn	**input**	Pointer to the structure containing data of the input three-phase stationary system (f32A-f32B-f32C). Arguments of the structure contain fixed point 32-bit values.
SWLIBS_3Syst_F32 *const	pOut	**output**	Pointer to the structure containing data of the output three-phase stationary orthogonal system (f32Aout-f32Bout-f32Cout). Arguments of the structure contain fixed point 32-bit values.

Note: *The inputs and the outputs are normalized to fit in the range [-1, 1).*

2.46.3.3 Code Example

```
#include "gmclib.h"

SWLIBS_3Syst_F32 f32trAbcIn;
SWLIBS_3Syst_F32 f32trAbcOut;

void main(void)
{
    // input phase A ~ sin(45) ~ 0.707106781
    // input phase B ~ sin(45 + 120) ~ 0.258819045
    // input phase C ~ sin(45 - 120) ~ -0.965925826
    f32trAbcIn.f32Arg1 = FRAC32(0.707106781);
    f32trAbcIn.f32Arg2 = FRAC32(0.258819045);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    f32trAbcIn.f32Arg3 = FRAC32(-0.965925826);
    // add DC offset - will be eliminated by the Beta-projection
    f32trAbcIn.f32Arg1 += FRAC32(0.1);
    f32trAbcIn.f32Arg2 += FRAC32(0.1);
    f32trAbcIn.f32Arg3 += FRAC32(0.1);

    // output should be
    // f32trAbcOut.f32Arg1 = 0x5A82799A ~ FRAC32(0.707)
    // f32trAbcOut.f32Arg2 = 0x2120FB83 ~ FRAC32(0.259)
    // f32trAbcOut.f32Arg3 = 0x8B70E520 ~ FRAC32(-0.911)
    GMCLIB_BetaProjection3Ph_F32(&f32trAbcOut,&f32trAbcIn);

    // output should be
    // f32trAbcOut.f32Arg1 = 0x5A82799A ~ FRAC32(0.707)
    // f32trAbcOut.f32Arg2 = 0x2120FB83 ~ FRAC32(0.259)
    // f32trAbcOut.f32Arg3 = 0x8B70E520 ~ FRAC32(-0.911)
    GMCLIB_BetaProjection3Ph(&f32trAbcOut,&f32trAbcIn,F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be
    // f32trAbcOut.f32Arg1 = 0x5A82799A ~ FRAC32(0.707)
    // f32trAbcOut.f32Arg2 = 0x2120FB83 ~ FRAC32(0.259)
    // f32trAbcOut.f32Arg3 = 0x8B70E520 ~ FRAC32(-0.911)
    GMCLIB_BetaProjection3Ph(&f32trAbcOut,&f32trAbcIn);
}
```

2.46.4 Function GMCLIB_BetaProjection3Ph_F16

2.46.4.1 Declaration

```
void GMCLIB_BetaProjection3Ph_F16(SWLIBS_3Syst_F16 *const pOut,
const SWLIBS_3Syst_F16 *const pIn);
```

2.46.4.2 Arguments

Table 279. GMCLIB_BetaProjection3Ph_F16 arguments

Type	Name	Direction	Description
const SWLIBS_3Syst_F16 *const	pIn	**input**	Pointer to the structure containing data of the input three-phase stationary system (f16A-f16B-f16C). Arguments of the structure contain fixed point 16-bit values.
SWLIBS_3Syst_F16 *const	pOut	**output**	Pointer to the structure containing data of the output three-phase stationary orthogonal system (f16Aout-f16Bout-f16Cout). Arguments of the structure contain fixed point 16-bit values.

Note: *The inputs and the outputs are normalized to fit in the range [-1, 1).*

2.46.4.3 Code Example

```
#include "gmclib.h"

SWLIBS_3Syst_F16 f16trAbcIn;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
SWLIBS_3Syst_F16 f16trAbcOut;

void main(void)
{
    // input phase A ~ sin(45) ~ 0.707106781
    // input phase B ~ sin(45 + 120) ~ 0.258819045
    // input phase C ~ sin(45 - 120) ~ -0.965925826
    f16trAbcIn.f16Arg1 = FRAC16(0.707106781);
    f16trAbcIn.f16Arg2 = FRAC16(0.258819045);
    f16trAbcIn.f16Arg3 = FRAC16(-0.965925826);
    // add DC offset - will be eliminated by the Beta-projection
    f16trAbcIn.f16Arg1 += FRAC16(0.1);
    f16trAbcIn.f16Arg2 += FRAC16(0.1);
    f16trAbcIn.f16Arg3 += FRAC16(0.1);

    // output should be
    // f16trAbcOut.f16Arg1 = 0x5A82 ~ FRAC16(0.707)
    // f16trAbcOut.f16Arg2 = 0x2121 ~ FRAC16(0.259)
    // f16trAbcOut.f16Arg3 = 0x8B71 ~ FRAC16(-0.911)
    GMCLIB_BetaProjection3Ph_F16(&f16trAbcOut,&f16trAbcIn);

    // output should be
    // f16trAbcOut.f16Arg1 = 0x5A82 ~ FRAC16(0.707)
    // f16trAbcOut.f16Arg2 = 0x2121 ~ FRAC16(0.259)
    // f16trAbcOut.f16Arg3 = 0x8B71 ~ FRAC16(-0.911)
    GMCLIB_BetaProjection3Ph(&f16trAbcOut,&f16trAbcIn,F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be
    // f16trAbcOut.f16Arg1 = 0x5A82 ~ FRAC16(0.707)
    // f16trAbcOut.f16Arg2 = 0x2121 ~ FRAC16(0.259)
    // f16trAbcOut.f16Arg3 = 0x8B71 ~ FRAC16(-0.911)
    GMCLIB_BetaProjection3Ph(&f16trAbcOut,&f16trAbcIn);
}
```

2.46.5 Function GMCLIB_BetaProjection3Ph_FLT

2.46.5.1 Declaration

```
void GMCLIB_BetaProjection3Ph_FLT(SWLIBS_3Syst_FLT *const pOut,
const SWLIBS_3Syst_FLT *const pIn);
```

2.46.5.2 Arguments

Table 280. GMCLIB_BetaProjection3Ph_FLT arguments

Type	Name	Direction	Description
const SWLIBS_3Syst_FLT *const	pIn	**input**	Pointer to the structure containing data of the input three-phase stationary system (fltA-fltB-fltC). Arguments of the structure contain single precision floating point values.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 280. GMCLIB_BetaProjection3Ph_FLT arguments...*continued*

Type	Name	Direction	Description
SWLIBS_3Syst_FLT *const	pOut	**output**	Pointer to the structure containing data of the output three-phase stationary orthogonal system (fltAout-fltBout-fltCout). Arguments of the structure contain single precision floating point values.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.46.5.3 Code Example

```
#include "gmclib.h"

SWLIBS_3Syst_FLT flttrAbcIn;
SWLIBS_3Syst_FLT flttrAbcOut;

void main(void)
{
    // input phase A ~ sin(45) ~ 0.707106781
    // input phase B ~ sin(45 + 120) ~ 0.258819045
    // input phase C ~ sin(45 - 120) ~ -0.965925826
    flttrAbcIn.fltArg1 = (tFloat)0.707106781;
    flttrAbcIn.fltArg2 = (tFloat)0.258819045;
    flttrAbcIn.fltArg3 = (tFloat)-0.965925826;
    // add DC offset - will be eliminated by the Beta-projection
    flttrAbcIn.fltArg1 += 0.1f;
    flttrAbcIn.fltArg2 += 0.1f;
    flttrAbcIn.fltArg3 += 0.1f;

    // output should be
    // flttrAbcOut.fltArg1 = 0.707106781;
    // flttrAbcOut.fltArg2 = 0.258819045;
    // flttrAbcOut.fltArg3 = -0.965925826;
    GMCLIB_BetaProjection3Ph_FLT(&flttrAbcOut,&flttrAbcIn);

    // output should be
    // flttrAbcOut.fltArg1 = 0.707106781;
    // flttrAbcOut.fltArg2 = 0.258819045;
    // flttrAbcOut.fltArg3 = -0.965925826;
    GMCLIB_BetaProjection3Ph(&flttrAbcOut,&flttrAbcIn,FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be
    // flttrAbcOut.fltArg1 = 0.70710677;
    // flttrAbcOut.fltArg2 = 0.25881904;
    // flttrAbcOut.fltArg3 = -0.96592587;
    GMCLIB_BetaProjection3Ph(&flttrAbcOut,&flttrAbcIn);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.47 Function GMCLIB_Clark

The function implements the Clarke transformation.

2.47.1 Description

The Clarke Transformation is used to transform values from the three-phase (A-B-C) coordinate system to the two-phase (α-β) orthogonal coordinate system, according to the following equations:

$$i_\alpha = i_A$$

$$i_\beta = (i_B - i_C) \cdot \frac{1}{\sqrt{3}}$$

Equation GMCLIB_Clark_Eq1

where it is assumed that the axis A (axis of the first phase) and the axis α are in the same direction.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.47.2 Re-entrancy

The function is re-entrant for a different set of input/output pointers.

2.47.3 Function GMCLIB_Clark_F32

2.47.3.1 Declaration

```
void GMCLIB_Clark_F32(SWLIBS_2Syst_F32 *const pOut, const
SWLIBS_3Syst_F32 *const pIn);
```

2.47.3.2 Arguments

Table 281. GMCLIB_Clark_F32 arguments

Type	Name	Direction	Description
const SWLIBS_3Syst_F32 *const	pIn	**input**	Pointer to the structure containing data of the three-phase stationary system (f32A-f32B-f32C). Arguments of the structure contain fixed point 32-bit values.
SWLIBS_2Syst_F32 *const	pOut	**output**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α-β). Arguments of the structure contain fixed point 32-bit values.

Note: *The inputs and the outputs are normalized to fit in the range [-1, 1).*

2.47.3.3 Code Example

```
#include "gmclib.h"

SWLIBS_3Syst_F32 f32trAbc;
SWLIBS_2Syst_F32 f32trAlBe;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
void main(void)
{
    // input phase A ~ sin(45) ~ 0.707106781
    // input phase B ~ sin(45 + 120) ~ 0.258819045
    // input phase C ~ sin(45 - 120) ~ -0.965925826
    f32trAbc.f32Arg1 = FRAC32(0.707106781);
    f32trAbc.f32Arg2 = FRAC32(0.258819045);
    f32trAbc.f32Arg3 = FRAC32(-0.965925826);

    // output should be
    // f32trAlBe.f32Arg1 = 0x5A827999 ~ FRAC32(0.707106781)
    // f32trAlBe.f32Arg2 = 0x5A827999 ~ FRAC32(0.707106781)
    GMCLIB_Clark_F32(&f32trAlBe,&f32trAbc);

    // output should be
    // f32trAlBe.f32Arg1 = 0x5A827999 ~ FRAC32(0.707106781)
    // f32trAlBe.f32Arg2 = 0x5A827999 ~ FRAC32(0.707106781)
    GMCLIB_Clark(&f32trAlBe,&f32trAbc,F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be
    // f32trAlBe.f32Arg1 = 0x5A827999 ~ FRAC32(0.707106781)
    // f32trAlBe.f32Arg2 = 0x5A827999 ~ FRAC32(0.707106781)
    GMCLIB_Clark(&f32trAlBe,&f32trAbc);
}
```

2.47.4 Function GMCLIB_Clark_F16

2.47.4.1 Declaration

```
void GMCLIB_Clark_F16(SWLIBS_2Syst_F16 *const pOut, const
SWLIBS_3Syst_F16 *const pIn);
```

2.47.4.2 Arguments

Table 282. GMCLIB_Clark_F16 arguments

Type	Name	Direction	Description
const SWLIBS_3Syst_F16 *const	pIn	**input**	Pointer to the structure containing data of the three-phase stationary system (f16A-f16B-f16C). Arguments of the structure contain fixed point 16-bit values.
SWLIBS_2Syst_F16 *const	pOut	**output**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α-β). Arguments of the structure contain fixed point 16-bit values.

Note: *The inputs and the outputs are normalized to fit in the range [-1, 1).*

2.47.4.3 Code Example

```
#include "gmclib.h"

SWLIBS_3Syst_F16 f16trAbc;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
SWLIBS_2Syst_F16 f16trAlBe;

void main(void)
{
    // input phase A ~ sin(45) ~ 0.707106781
    // input phase B ~ sin(45 + 120) ~ 0.258819045
    // input phase C ~ sin(45 - 120) ~ -0.965925826
    f16trAbc.f16Arg1 = FRAC16(0.707106781);
    f16trAbc.f16Arg2 = FRAC16(0.258819045);
    f16trAbc.f16Arg3 = FRAC16(-0.965925826);

    // output should be
    // f16trAlBe.f16Arg1 = 0x5A82 ~ FRAC16(0.707106781)
    // f16trAlBe.f16Arg2 = 0x5A82 ~ FRAC16(0.707106781)
    GMCLIB_Clark_F16(&f16trAlBe,&f16trAbc);

    // output should be
    // f16trAlBe.f16Arg1 = 0x5A82 ~ FRAC16(0.707106781)
    // f16trAlBe.f16Arg2 = 0x5A82 ~ FRAC16(0.707106781)
    GMCLIB_Clark(&f16trAlBe,&f16trAbc,F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be
    // f16trAlBe.f16Arg1 = 0x5A82 ~ FRAC16(0.707106781)
    // f16trAlBe.f16Arg2 = 0x5A82 ~ FRAC16(0.707106781)
    GMCLIB_Clark(&f16trAlBe,&f16trAbc);
}
```

2.47.5 Function GMCLIB_Clark_FLT

2.47.5.1 Declaration

```
void GMCLIB_Clark_FLT(SWLIBS_2Syst_FLT *const pOut, const
SWLIBS_3Syst_FLT *const pIn);
```

2.47.5.2 Arguments

Table 283. GMCLIB_Clark_FLT arguments

Type	Name	Direction	Description
const SWLIBS_3Syst_FLT *const	pIn	**input**	Pointer to the structure containing data of the three-phase stationary system (fltA-fltB-fltC). Arguments of the structure contain single precision floating point values.
SWLIBS_2Syst_FLT *const	pOut	**output**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α-β). Arguments of the structure contain single precision floating point values.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.47.5.3 Code Example

```
#include "gmclib.h"

SWLIBS_3Syst_FLT flttrAbc;
SWLIBS_2Syst_FLT flttrAlBe;

void main(void)
{
    // input phase A ~ sin(45) ~ 0.707106781
    // input phase B ~ sin(45 + 120) ~ 0.258819045
    // input phase C ~ sin(45 - 120) ~ -0.965925826
    flttrAbc.fltArg1 = (tFloat)0.707106781;
    flttrAbc.fltArg2 = (tFloat)0.258819045;
    flttrAbc.fltArg3 = (tFloat)-0.965925826;

    // output should be
    // flttrAlBe.fltArg1 = 0.707106781
    // flttrAlBe.fltArg2 = 0.707106781
    GMCLIB_Clark_FLT(&flttrAlBe,&flttrAbc);

    // output should be
    // flttrAlBe.fltArg1 = 0.707106781
    // flttrAlBe.fltArg2 = 0.707106781
    GMCLIB_Clark(&flttrAlBe,&flttrAbc,FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be
    // flttrAlBe.fltArg1 = 0.707106781
    // flttrAlBe.fltArg2 = 0.707106781
    GMCLIB_Clark(&flttrAlBe,&flttrAbc);
}
```

2.48 Function GMCLIB_Clark6Ph

The function implements the direct way of the six-phase Clarke transformation.

2.48.1 Description

This function transforms the values from the six-phase machine consisting of the two sets of starconnected three-phase stator windings (f32A-f32B-f32C) and (f32D-f32E-f32F) to the two-phase (α-β) orthogonal coordinate system, according to the following equations:

$$X_\alpha = \frac{1}{2}X_A + (AlphaD \cdot X_D) + (AlphaE \cdot X_E) + (AlphaF \cdot X_F)$$

$$X_\beta = \frac{\sqrt{3}}{6}(X_B - X_C) + (BetaD \cdot X_D) + (BetaE \cdot X_E) + (BetaF \cdot X_F)$$

Equation GMCLIB_Clark6Ph_Eq1

where AlphaD, AlphaE, AlphaF, BetaD, BetaE, and BetaF are the transformation coefficients calculated by GMCLIB_Clark6PhInit function based on the shift between the

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

two windings in the six-phase machine. Also it is assumed that the axis A (axis of the first phase) and the axis α are in the same direction.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.48.2 Re-entrancy

The function is re-entrant for a different set of input/output pointers.

2.48.3 Function GMCLIB_Clark6Ph_F32

2.48.3.1 Declaration

```
void GMCLIB_Clark6Ph_F32(SWLIBS_2Syst_F32 *const pOut, const
SWLIBS_6Syst_F32 *const pIn, const GMCLIB_CLARK_6PH_T_F32 *const
pParam);
```

2.48.3.2 Arguments

Table 284. GMCLIB_Clark6Ph_F32 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F32 *const	pOut	**output**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α-β). Arguments of the structure contain fixed point 32-bit values.
const SWLIBS_6Syst_F32 *const	pIn	**input**	Pointer to the structure containing data of the six-phase machine consisting of two sets of starconnected three-phase stator windings f32A-f32B-f32C and f32D-f32E-f32F. Arguments of the structure contain fixed point 32-bit values.
const GMCLIB_CLARK_6PH_T_F32 *const	pParam	**input**	Pointer to the structure containing the transformation coefficient calculated by GMCLIB_Clark6PhInit function. Arguments of the structure contain fixed point 32-bit values.

Note: *The inputs and the outputs are normalized to fit in the range [-1, 1).*

2.48.3.3 Code Example

```
#include "gmclib.h"

GMCLIB_CLARK_6PH_T_F32 pParam;

void main(void)
{
  tFrac32 f32Shift;

  // Initialize GMCLIB_Clark6Ph transformation parameters

  // input shift -30° (-pi/6) normalized into [-1, 1) interval
  f32Shift = (tFrac32)-357913941;

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_Clark6PhInit_F32(&pParam, f32Shift);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_Clark6PhInit(&pParam, f32Shift,F32);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_Clark6PhInit(&pParam, f32Shift);

  while(1);
}

// Periodical function or interrupt
void loop(void)
{
  SWLIBS_6Syst_F32 pIn_6Syst;
  SWLIBS_2Syst_F32 pOut_2Syst;

  // Read the A/D, etc.
  // (...)

  // Calculate alpha-beta values:

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_Clark6Ph_F32(&pOut_2Syst, &pIn_6Syst, &pParam);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_Clark6Ph(&pOut_2Syst, &pIn_6Syst, &pParam, F32);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_Clark6Ph(&pOut_2Syst, &pIn_6Syst, &pParam);

  // Park transform, PI controllers, new PWM values calculation, etc.
  // (...)
}
```

2.48.4 Function GMCLIB_Clark6Ph_F16

2.48.4.1 Declaration

```
void GMCLIB_Clark6Ph_F16(SWLIBS_2Syst_F16 *const pOut, const
SWLIBS_6Syst_F16 *const pIn, const GMCLIB_CLARK_6PH_T_F16 *const
pParam);
```

2.48.4.2 Arguments

Table 285. GMCLIB_Clark6Ph_F16 arguments

Type	Name	Direction	Description
const SWLIBS_6Syst_F16 *const	pIn	**input**	Pointer to the structure containing data of the six-phase machine consisting of two sets of starconnected three-phase stator windings f16A-f16B-f16C and f16D-f16E-f16F. Arguments of the structure contain fixed point 16-bit values.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 285. GMCLIB_Clark6Ph_F16 arguments...*continued*

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *const	pOut	**output**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α-β). Arguments of the structure contain fixed point 16-bit values.
const GMCLIB_CLARK_6PH_T_F16 *const	pParam	**input**	Pointer to the structure containing the transformation coefficient calculated by GMCLIB_Clark6PhInit function. Arguments of the structure contain fixed point 16-bit values.

Note: *The inputs and the outputs are normalized to fit in the range [-1, 1).*

2.48.4.3 Code Example

```
#include "gmclib.h"

GMCLIB_CLARK_6PH_T_F16 pParam;

void main(void)
{
  tFrac16 f16Shift;

  // Initialize GMCLIB_Clark6Ph transformation parameters

  // input shift -30° (-pi/6) normalized into [-1, 1) interval
  f16Shift = (tFrac16)-5461;

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_Clark6PhInit_F16(&pParam, f16Shift);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_Clark6PhInit(&pParam, f16Shift,F16);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 16-bit fractional implementation is selected as default.
  GMCLIB_Clark6PhInit(&pParam, f16Shift);

  while(1);
}

// Periodical function or interrupt
void loop(void)
{
  SWLIBS_6Syst_F16 pIn_6Syst;
  SWLIBS_2Syst_F16 pOut_2Syst;

  // Read the A/D, etc.
  // (...)

  // Calculate alpha-beta values:

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_Clark6Ph_F16(&pOut_2Syst, &pIn_6Syst, &pParam);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_Clark6Ph(&pOut_2Syst, &pIn_6Syst, &pParam, F16);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 16-bit fractional implementation is selected as default.
  GMCLIB_Clark6Ph(&pOut_2Syst, &pIn_6Syst, &pParam);

  // Park transform, PI controllers, new PWM values calculation, etc.
  // (...)
}
```

2.48.5 Function GMCLIB_Clark6Ph_FLT

2.48.5.1 Declaration

```
void GMCLIB_Clark6Ph_FLT(SWLIBS_2Syst_FLT *const pOut, const
SWLIBS_6Syst_FLT *const pIn, const GMCLIB_CLARK_6PH_T_FLT *const
pParam);
```

2.48.5.2 Arguments

Table 286. GMCLIB_Clark6Ph_FLT arguments

Type	Name	Direction	Description
const SWLIBS_6Syst_FLT *const	pIn	**input**	Pointer to the structure containing data of the six-phase machine consisting of two sets of starconnected three-phase stator windings fltA-fltB-fltC and fltD-fltE-fltF. Arguments of the structure contain single precision floating point values.
SWLIBS_2Syst_FLT *const	pOut	**output**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α-β). Arguments of the structure contain single precision floating point values.
const GMCLIB_CLARK_6PH_T_FLT *const	pParam	**input**	Pointer to the structure containing the transformation coefficient calculated by GMCLIB_Clark6PhInit function. Arguments of the structure contain single precision floating point values.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.48.5.3 Code Example

```
#include "gmclib.h"

GMCLIB_CLARK_6PH_T_FLT pParam;

void main(void)
{
  tFloat fltShift;

  // Initialize GMCLIB_Clark6Ph transformation parameters
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // input shift -30° (-pi/6)
  fltShift = -FLOAT_PI_DIVBY_6;

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_Clark6PhInit_FLT(&pParam, fltShift);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_Clark6PhInit(&pParam, fltShift,FLT);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_Clark6PhInit(&pParam, fltShift);

  while(1);
}

// Periodical function or interrupt
void loop(void)
{
  SWLIBS_6Syst_FLT pIn_6Syst;
  SWLIBS_2Syst_FLT pOut_2Syst;

  // Read the A/D, etc.
  // (...)

  // Calculate alpha-beta values:

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_Clark6Ph_FLT(&pOut_2Syst, &pIn_6Syst, &pParam);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_Clark6Ph(&pOut_2Syst, &pIn_6Syst, &pParam, FLT);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_Clark6Ph(&pOut_2Syst, &pIn_6Syst, &pParam);

  // Park transform, PI controllers, new PWM values calculation, etc.
  // (...)
}
```

2.48.6 Function GMCLIB_Clark6PhInit

The function initializes the parameters structure required for six phase Clarke transformation.

2.48.6.1 Description

GMCLIB_Clark6PhInit calculates the transformation coefficients required for GMCLIB_Clark6Ph function from the shift between two sets of star-connected three phase stator windings, according to the following equations:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$AlphaD = \frac{2}{6} \cdot \cos(\theta)$$
$$AlphaE = \frac{2}{6} \cdot \cos\left(\theta - \frac{2}{3}\pi\right)$$
$$AlphaF = \frac{2}{6} \cdot \cos\left(\theta + \frac{2}{3}\pi\right)$$
$$BetaD = -\frac{2}{6} \cdot \sin(\theta)$$
$$BetaE = -\frac{2}{6} \cdot \sin\left(\theta - \frac{2}{3}\pi\right)$$
$$BetaF = -\frac{2}{6} \cdot \sin\left(\theta + \frac{2}{3}\pi\right)$$

Equation GMCLIB_Clark6PhInit_Eq1

where θ is the shift between the two windings of the six-phase machine.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.48.6.2 Re-entrancy

The function is re-entrant for a different `pParam`.

2.48.6.3 Function GMCLIB_Clark6PhInit_F32

2.48.6.3.1 Declaration

```
void GMCLIB_Clark6PhInit_F32(GMCLIB_CLARK_6PH_T_F32 *const
pParam, tFrac32 f32WindingShift);
```

2.48.6.3.2 Arguments

Table 287. GMCLIB_Clark6PhInit_F32 arguments

Type	Name	Direction	Description
tFrac32	f32WindingShift	**input**	Input argument is a 32-bit number that contains an angle in radians from interval [-π, π) normalized between [-1, 1). This angle represents the shift between two sets of star-connected three phase stator windings.
GMCLIB_CLARK_6PH_T_F32 *const	pParam	**output**	Pointer to the structure containing the transformation coefficient required for GMCLIB_Clark6Ph function. Arguments of the structure contain fixed point 32-bit values.

2.48.6.3.3 Code Example

```
#include "gmclib.h"

GMCLIB_CLARK_6PH_T_F32 pParam;

void main(void)
{
  tFrac32 f32Shift;

  // Initialize GMCLIB_Clark6Ph transformation parameters

  // input shift -30° (-pi/6) normalized into [-1, 1) interval
  f32Shift = (tFrac32)-357913941;;
```


© NXP B.V. 2023. All rights reserved.

```
  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_Clark6PhInit_F32(&pParam, f32Shift);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_Clark6PhInit(&pParam, f32Shift,F32);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_Clark6PhInit(&pParam, f32Shift);

  while(1);
}

// Periodical function or interrupt
void loop(void)
{
  SWLIBS_6Syst_F32 pIn_6Syst;
  SWLIBS_2Syst_F32 pOut_2Syst;

  // Read the A/D, etc.
  // (...)

  // Calculate alpha-beta values:

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_Clark6Ph_F32(&pOut_2Syst, &pIn_6Syst, &pParam);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_Clark6Ph(&pOut_2Syst, &pIn_6Syst, &pParam, F32);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_Clark6Ph(&pOut_2Syst, &pIn_6Syst, &pParam);

  // Park transform, PI controllers, new PWM values calculation, etc.
  // (...)
}
```

2.48.6.4 Function GMCLIB_Clark6PhInit_F16

2.48.6.4.1 Declaration

```
void GMCLIB_Clark6PhInit_F16(GMCLIB_CLARK_6PH_T_F16 *const
pParam, tFrac16 f16WindingShift);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.48.6.4.2 Arguments

Table 288. GMCLIB_Clark6PhInit_F16 arguments

Type	Name	Direction	Description
tFrac16	f16WindingShift	**input**	Input argument is a 16-bit number that contains an angle in radians from interval [-π, π) normalized between [-1, 1). This angle represents the shift between two sets of star-connected three phase stator windings.
GMCLIB_CLARK_6PH_T_F16 *const	pParam	**output**	Pointer to the structure containing the transformation coefficient required for GMCLIB_Clark6Ph function. Arguments of the structure contain fixed point 16-bit values.

2.48.6.4.3 Code Example

```
#include "gmclib.h"

GMCLIB_CLARK_6PH_T_F16 pParam;

void main(void)
{
  tFrac16 f16Shift;

  // Initialize GMCLIB_Clark6Ph transformation parameters

  // input shift -30° (-pi/6) normalized into [-1, 1) interval
  f16Shift = (tFrac16)-5461;

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_Clark6PhInit_F16(&pParam, f16Shift);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_Clark6PhInit(&pParam, f16Shift,F16);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 16-bit fractional implementation is selected as default.
  GMCLIB_Clark6PhInit(&pParam, f16Shift);

  while(1);
}

// Periodical function or interrupt
void loop(void)
{
  SWLIBS_6Syst_F16 pIn_6Syst;
  SWLIBS_2Syst_F16 pOut_2Syst;

  // Read the A/D, etc.
  // (...)

  // Calculate alpha-beta values:

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_Clark6Ph_F16(&pOut_2Syst, &pIn_6Syst, &pParam);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_Clark6Ph(&pOut_2Syst, &pIn_6Syst, &pParam, F16);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 16-bit fractional implementation is selected as default.
  GMCLIB_Clark6Ph(&pOut_2Syst, &pIn_6Syst, &pParam);

  // Park transform, PI controllers, new PWM values calculation, etc.
  // (...)
}
```

2.48.6.5 Function GMCLIB_Clark6PhInit_FLT

2.48.6.5.1 Declaration

```
void GMCLIB_Clark6PhInit_FLT(GMCLIB_CLARK_6PH_T_FLT *const
pParam, tFloat fltWindingShift);
```

2.48.6.5.2 Arguments

Table 289. GMCLIB_Clark6PhInit_FLT arguments

Type	Name	Direction	Description
tFloat	fltWindingShift	**input**	Input argument is a single precision floating point number that contains an angle in radians from interval [- π, π]. This angle represents the shift between two sets of star-connected three phase stator windings.
GMCLIB_CLARK_6PH_T_FLT *const	pParam	**output**	Pointer to the structure containing the transformation coefficient required for GMCLIB_Clark6Ph function. Arguments of the structure contain single precision floating point values.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.48.6.5.3 Code Example

```
#include "gmclib.h"

GMCLIB_CLARK_6PH_T_FLT pParam;

void main(void)
{
  tFloat fltShift;

  // Initialize GMCLIB_Clark6Ph transformation parameters

  // input shift -30° (-pi/6)
  fltShift = -FLOAT_PI_DIVBY_6;

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_Clark6PhInit_FLT(&pParam, fltShift);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_Clark6PhInit(&pParam, fltShift,FLT);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_Clark6PhInit(&pParam, fltShift);

  while(1);
}

// Periodical function or interrupt
void loop(void)
{
  SWLIBS_6Syst_FLT pIn_6Syst;
  SWLIBS_2Syst_FLT pOut_2Syst;

  // Read the A/D, etc.
  // (...)

  // Calculate alpha-beta values:

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_Clark6Ph_FLT(&pOut_2Syst, &pIn_6Syst, &pParam);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_Clark6Ph(&pOut_2Syst, &pIn_6Syst, &pParam, FLT);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_Clark6Ph(&pOut_2Syst, &pIn_6Syst, &pParam);

  // Park transform, PI controllers, new PWM values calculation, etc.
  // (...)
}
```

2.48.6.6 Function GMCLIB_ClarkInv6PhInit_F16

2.48.6.6.1 Declaration

```
void GMCLIB_ClarkInv6PhInit_F16(GMCLIB_CLARKINV_6PH_T_F16 *const
pParam, tFrac16 f16WindingShift);
```

2.48.6.6.2 Arguments

Table 290. GMCLIB_ClarkInv6PhInit_F16 arguments

Type	Name	Direction	Description
tFrac16	f16WindingShift	**input**	Input argument is a 16-bit number that contains an angle in radians from interval [-π, π) normalized between [-1, 1). This angle represents the shift between two sets of star-connected three phase stator windings.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 290. GMCLIB_ClarkInv6PhInit_F16 arguments...*continued*

Type	Name	Direction	Description
GMCLIB_CLARKINV_6PH_T_F16 *const	pParam	**output**	Pointer to the structure containing the transformation coefficient required for GMCLIB_ClarkInv6Ph function. Arguments of the structure contain fixed point 16-bit values.

2.48.6.6.3 Code Example

```
#include "gmclib.h"

GMCLIB_CLARKINV_6PH_T_F16 pParam;

void main(void)
{
  tFrac16 f16Shift;

  // Initialize GMCLIB_ClarkInv6Ph transformation parameters

  // input shift -30° (-pi/6) normalized into [-1, 1) interval
  f16Shift = FRAC16(-1.0/6.0);

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6PhInit_F16(&pParam, f16Shift);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6PhInit(&pParam, f16Shift,F16);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 16-bit fractional implementation is selected as default.
  GMCLIB_ClarkInv6PhInit(&pParam, f16Shift);

  while(1);
}

// Periodical function or interrupt
void loop(void)
{
  SWLIBS_6Syst_F16 pOutPwm;
  SWLIBS_2Syst_F16 pIn;

  // Read the A/D, Transform from 6-phase to d-q, PI controllers, Inverse Park transform etc.
  // (...)

  // Calculate alpha-beta values:

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6Ph_F16(&pOut, &pIn, &pParam);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6Ph(&pOut, &pIn, &pParam, F16);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 16-bit fractional implementation is selected as default.
  GMCLIB_ClarkInv6Ph(&pOut, &pIn, &pParam);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.48.6.7 Function GMCLIB_ClarkInv6PhInit_FLT

2.48.6.7.1 Declaration

```
void GMCLIB_ClarkInv6PhInit_FLT(GMCLIB_CLARKINV_6PH_T_FLT *const
pParam, tFloat fltWindingShift);
```

2.48.6.7.2 Arguments

Table 291. GMCLIB_ClarkInv6PhInit_FLT arguments

Type	Name	Direction	Description
tFloat	fltWindingShift	**input**	Input argument is a single precision floating-point number that contains an angle in radians from interval [- π, π). This angle represents the shift between two sets of star-connected three phase stator windings.
GMCLIB_CLARKINV_6PH_T_FLT *const	pParam	**output**	Pointer to the structure containing the transformation coefficient required for GMCLIB_ClarkInv6Ph function. Arguments of the structure contain single precision floating-point values.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.48.6.7.3 Code Example

```
#include "gmclib.h"

GMCLIB_CLARKINV_6PH_T_F16 pParam;

void main(void)
{
  tFrac32 fltShift;

  // Initialize GMCLIB_ClarkInv6Ph transformation parameters

  // input shift -30° (-pi/6)
  fltShift = -FLOAT_PI_DIVBY_6;

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6PhInit_FLT(&pParam, fltShift);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6PhInit(&pParam, fltShift,FLT);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_ClarkInv6PhInit(&pParam, fltShift);

  while(1);
}

// Periodical function or interrupt
void loop(void)
{
  SWLIBS_6Syst_FLT pOutPwm;
  SWLIBS_2Syst_FLT pIn;

  // Read the A/D, Transform from 6-phase to d-q, PI controllers, Inverse Park transform etc.
  // (...)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // Calculate alpha-beta values:

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6Ph_FLT(&pOut, &pIn, &pParam);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6Ph(&pOut, &pIn, &pParam, FLT);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if single precision floating point implementation selected as default
  GMCLIB_ClarkInv6Ph(&pOut, &pIn, &pParam);
}
```

2.48.6.8 Function GMCLIB_SvmStd6PhInit_F16

2.48.6.8.1 Declaration

```
void GMCLIB_SvmStd6PhInit_F16(GMCLIB_SVMSTD_6PH_T_F16 *const
pParam, tFrac16 f16WindingShift);
```

2.48.6.8.2 Arguments

Table 292. GMCLIB_SvmStd6PhInit_F16 arguments

Type	Name	Direction	Description
tFrac16	f16WindingShift	**input**	Input argument is a 16-bit number that contains an angle in radians from interval [-π, π) normalized between [-1, 1). This angle represents the shift between two sets of star-connected three phase stator windings.
GMCLIB_SVMSTD_6PH_T_F16 *const	pParam	**output**	Pointer to the structure containing the transformation coefficient required for GMCLIB_SvmStd6Ph function. Arguments of the structure contain fixed point 16-bit values.

2.48.6.8.3 Code Example

```
#include "gmclib.h"

GMCLIB_SVMSTD_6PH_T_F16 pParam;

void main(void)
{
  tFrac16 f16Shift;

  // Initialize GMCLIB_SvmStd6Ph transformation parameters

  // input shift -30° (-pi/6) normalized into [-1, 1) interval
  f16Shift = FRAC16(-1.0/6.0);

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_SvmStd6PhInit_F16(&pParam, f16Shift);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_SvmStd6PhInit(&pParam, f16Shift,F16);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 16-bit fractional implementation is selected as default.
  GMCLIB_SvmStd6PhInit(&pParam, f16Shift);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  while(1);
}

// Periodical function or interrupt
void loop(void)
{
  SWLIBS_6Syst_F16 pOutPwm;
  SWLIBS_2Syst_U16 pOutSectors;
  SWLIBS_2Syst_F16 pIn;

  // Read the A/D, Transform from 6-phase to d-q, PI controllers, Inverse Park transform etc.
  // (...)

  // Calculate alpha-beta values:

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_SvmStd6Ph_F16(&pOutPwm, &pOutSectors, &pIn, &pParam);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_SvmStd6Ph(&pOutPwm, &pOutSectors, &pIn, &pParam, F16);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 16-bit fractional implementation is selected as default.
  GMCLIB_SvmStd6Ph(&pOutPwm, &pOutSectors, &pIn, &pParam);
}
```

2.49 Function GMCLIB_ClarkInv

The function implements the inverse Clarke transformation.

2.49.1 Description

The GMCLIB_ClarkInv function calculates the Inverse Clarke transformation, which is used to transform values from the two-phase (α-β) orthogonal coordinate system to the three-phase (A-B-C) coordinate system, according to these equations:

$$i_A = i_\alpha$$

$$i_B = -\frac{1}{2} \cdot i_\alpha + \frac{\sqrt{3}}{2} \cdot i_\beta$$

$$i_C = -\frac{1}{2} \cdot i_\alpha - \frac{\sqrt{3}}{2} \cdot i_\beta$$

Equation GMCLIB_ClarkInv_Eq1

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.49.2 Re-entrancy

The function is re-entrant for a different set of input/output pointers.

2.49.3 Function GMCLIB_ClarkInv_F32

2.49.3.1 Declaration

```
void GMCLIB_ClarkInv_F32(SWLIBS_3Syst_F32 *const pOut, const
SWLIBS_2Syst_F32 *const pIn);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.49.3.2 Arguments

Table 293. GMCLIB_ClarkInv_F32 arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α-β). Arguments of the structure contain fixed point 32-bit values.
SWLIBS_3Syst_F32 *const	pOut	**output**	Pointer to the structure containing data of the three-phase stationary system (f32A-f32B-f32C). Arguments of the structure contain fixed point 32-bit values.

Note: *The inputs and the outputs are normalized to fit in the range [-1, 1).*

2.49.3.3 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_F32 f32trAlBe;
SWLIBS_3Syst_F32 f32trAbc;

void main(void)
{
    // input phase alpha ~ sin(45) ~ 0.707106781
    // input phase beta ~ cos(45) ~ 0.707106781
    f32trAlBe.f32Arg1 = FRAC32(0.707106781);
    f32trAlBe.f32Arg2 = FRAC32(0.707106781);

    // output should be
    // f32trAbc.f32Arg1 = 0x5A827999 ~ FRAC32(0.707106781)
    // f32trAbc.f32Arg2 = 0x2120FB83 ~ FRAC32(0.258819045)
    // f32trAbc.f32Arg3 = 0x845C8AE5 ~ FRAC32(-0.965925826)
    GMCLIB_ClarkInv_F32(&f32trAbc,&f32trAlBe);

    // output should be
    // f32trAbc.f32Arg1 = 0x5A827999 ~ FRAC32(0.707106781)
    // f32trAbc.f32Arg2 = 0x2120FB83 ~ FRAC32(0.258819045)
    // f32trAbc.f32Arg3 = 0x845C8AE5 ~ FRAC32(-0.965925826)
    GMCLIB_ClarkInv(&f32trAbc,&f32trAlBe,F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be
    // f32trAbc.f32Arg1 = 0x5A827999 ~ FRAC32(0.707106781)
    // f32trAbc.f32Arg2 = 0x2120FB83 ~ FRAC32(0.258819045)
    // f32trAbc.f32Arg3 = 0x845C8AE5 ~ FRAC32(-0.965925826)
    GMCLIB_ClarkInv(&f32trAbc,&f32trAlBe);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.49.4 Function GMCLIB_ClarkInv_F16

2.49.4.1 Declaration

```
void GMCLIB_ClarkInv_F16(SWLIBS_3Syst_F16 *const pOut, const
SWLIBS_2Syst_F16 *const pIn);
```

2.49.4.2 Arguments

Table 294. GMCLIB_ClarkInv_F16 arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_F16 *const	pIn	**input**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α-β). Arguments of the structure contain fixed point 16-bit values.
SWLIBS_3Syst_F16 *const	pOut	**output**	Pointer to the structure containing data of the three-phase stationary system (f16A-f16B-f16C). Arguments of the structure contain fixed point 16-bit values.

Note: *The inputs and the outputs are normalized to fit in the range [-1, 1).*

2.49.4.3 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_F16 f16trAlBe;
SWLIBS_3Syst_F16 f16trAbc;

void main(void)
{
    // input phase alpha ~ sin(45) ~ 0.707106781
    // input phase beta ~ cos(45) ~ 0.707106781
    f16trAlBe.f16Arg1 = FRAC16(0.707106781);
    f16trAlBe.f16Arg2 = FRAC16(0.707106781);

    // output should be
    // f16trAbc.f16Arg1 = 0x5A82 ~ FRAC16(0.707106781)
    // f16trAbc.f16Arg2 = 0x2120 ~ FRAC16(0.258819045)
    // f16trAbc.f16Arg3 = 0x845C ~ FRAC16(-0.965925826)
    GMCLIB_ClarkInv_F16(&f16trAbc,&f16trAlBe);

    // output should be
    // f16trAbc.f16Arg1 = 0x5A82 ~ FRAC16(0.707106781)
    // f16trAbc.f16Arg2 = 0x2120 ~ FRAC16(0.258819045)
    // f16trAbc.f16Arg3 = 0x845C ~ FRAC16(-0.965925826)
    GMCLIB_ClarkInv(&f16trAbc,&f16trAlBe,F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be
    // f16trAbc.f16Arg1 = 0x5A82 ~ FRAC16(0.707106781)
    // f16trAbc.f16Arg2 = 0x2120 ~ FRAC16(0.258819045)
    // f16trAbc.f16Arg3 = 0x845C ~ FRAC16(-0.965925826)
    GMCLIB_ClarkInv(&f16trAbc,&f16trAlBe);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
}
```

2.49.5 Function GMCLIB_ClarkInv_FLT

2.49.5.1 Declaration

```
void GMCLIB_ClarkInv_FLT(SWLIBS_3Syst_FLT *const pOut, const
SWLIBS_2Syst_FLT *const pIn);
```

2.49.5.2 Arguments

Table 295. GMCLIB_ClarkInv_FLT arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_FLT *const	pIn	**input**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α-β). Arguments of the structure contain single precision floating point values.
SWLIBS_3Syst_FLT *const	pOut	**output**	Pointer to the structure containing data of the three-phase stationary system (fltA-fltB-fltC). Arguments of the structure contain single precision floating point values.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.49.5.3 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_FLT flttrAlBe;
SWLIBS_3Syst_FLT flttrAbc;

void main(void)
{
    // input phase alpha ~ sin(45) ~ 0.707106781
    // input phase beta ~ cos(45) ~ 0.707106781
    flttrAlBe.fltArg1 = (tFloat)0.707106781;
    flttrAlBe.fltArg2 = (tFloat)0.707106781;

    // output should be
    // flttrAbc.fltArg1 = 0.707106781
    // flttrAbc.fltArg2 = 0.258819045
    // flttrAbc.fltArg3 = -0.965925826
    GMCLIB_ClarkInv_FLT(&flttrAbc,&flttrAlBe);

    // output should be
    // flttrAbc.fltArg1 = 0.707106781
    // flttrAbc.fltArg2 = 0.258819045
    // flttrAbc.fltArg3 = -0.965925826
    GMCLIB_ClarkInv(&flttrAbc,&flttrAlBe,FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // output should be
    // flttrAbc.fltArg1 ~ phA = 0.707106781
    // flttrAbc.fltArg2 ~ phB = 0.258819045
    // flttrAbc.fltArg3 ~ phC = -0.965925826
    GMCLIB_ClarkInv(&flttrAbc,&flttrAlBe);
}
```

2.50 Function GMCLIB_ClarkInv6Ph

The function implements the six-phase inverse Clarke transformation.

2.50.1 Description

The GMCLIB_ClarkInv6Ph function calculates the six-phase Inverse Clarke transformation, which is used to transform values from the two-phase (α-β) orthogonal coordinate system to the six-phase system consisting of the two sets of starconnected three-phase stator windings (f32A-f32B-f32C) and (f32D-f32E-f32F), according to these equations:

$$X_A = X_\alpha \qquad X_D = (AlphaD \cdot X_\alpha) + (BetaD \cdot X_\beta)$$

$$X_B = \frac{1}{2}\left(-X_\alpha + \sqrt{3}\,U_\beta\right) \qquad X_E = (AlphaE \cdot X_\alpha) + (BetaE \cdot X_\beta)$$

$$X_C = \frac{1}{2}\left(-X_\alpha - \sqrt{3}\,U_\beta\right) \qquad X_F = (AlphaF \cdot X_\alpha) + (BetaF \cdot X_\beta)$$

Equation GMCLIB_ClarkInv6Ph_Eq1

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.50.2 Re-entrancy

The function is re-entrant for a different set of input/output pointers.

2.50.3 Function GMCLIB_ClarkInv6Ph_F32

2.50.3.1 Declaration

```
void GMCLIB_ClarkInv6Ph_F32(SWLIBS_6Syst_F32 *const pOut, const
SWLIBS_2Syst_F32 *const pIn, const GMCLIB_CLARKINV_6PH_T_F32
*const pParam);
```

2.50.3.2 Arguments

Table 296. GMCLIB_ClarkInv6Ph_F32 arguments

Type	Name	Direction	Description
SWLIBS_6Syst_F32 *const	pOut	**output**	Pointer to the structure containing calculated duty-cycle ratios of the six-phase machine consisting of two sets of starconnected three-phase stator windings f32A-f32B-f32C and f32D-f32E-f32F. Arguments of the structure contain fixed point 32-bit values.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 296. GMCLIB_ClarkInv6Ph_F32 arguments...*continued*

Type	Name	Direction	Description
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector. Arguments of the structure contain fixed point 32-bit values.
const GMCLIB_CLARKINV_6PH_T_F32 *const	pParam	**output**	Pointer to the structure containing the transformation coefficient calculated by GMCLIB_ClarkInv6PhInit function. Arguments of the structure contain fixed point 32-bit values.

2.50.3.3 Code Example

```
#include "gmclib.h"

GMCLIB_CLARKINV_6PH_T_F32 pParam;

void main(void)
{
  tFrac32 f32Shift;

  // Initialize GMCLIB_ClarkInv6Ph transformation parameters

  // input shift -30° (-pi/6) normalized into [-1, 1) interval
  f32Shift = FRAC32(-1.0/6.0);

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6PhInit_F32(&pParam, f32Shift);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6PhInit(&pParam, f32Shift,F32);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_ClarkInv6PhInit(&pParam, f32Shift);

  while(1);
}

// Periodical function or interrupt
void loop(void)
{
  SWLIBS_6Syst_F32 pOut;
  SWLIBS_2Syst_F32 pIn;

  // Read the A/D, Transform from 6-phase to d-q, PI controllers, Inverse Park transform etc.
  // (...)

  // Calculate alpha-beta values:

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6Ph_F32(&pOut, &pIn, &pParam);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6Ph(&pOut, &pIn, &pParam, F32);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_ClarkInv6Ph(&pOut, &pIn, &pParam);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.50.4 Function GMCLIB_ClarkInv6Ph_F16

2.50.4.1 Declaration

```
void GMCLIB_ClarkInv6Ph_F16(SWLIBS_6Syst_F16 *const pOut, const
SWLIBS_2Syst_F16 *const pIn, const GMCLIB_CLARKINV_6PH_T_F16
*const pParam);
```

2.50.4.2 Arguments

Table 297. GMCLIB_ClarkInv6Ph_F16 arguments

Type	Name	Direction	Description
SWLIBS_6Syst_F16 *const	pOut	**output**	Pointer to the structure containing calculated duty-cycle ratios of the six-phase machine consisting of two sets of starconnected three-phase stator windings f16A-f16B-f16C and f16D-f16E-f16F. Arguments of the structure contain fixed point 16-bit values.
const SWLIBS_2Syst_F16 *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector. Arguments of the structure contain fixed point 16-bit values.
const GMCLIB_CLARKINV_6PH_T_F16 *const	pParam	**output**	Pointer to the structure containing the transformation coefficient calculated by GMCLIB_ClarkInv6PhInit function. Arguments of the structure contain fixed point 16-bit values.

2.50.4.3 Code Example

```
#include "gmclib.h"

GMCLIB_CLARKINV_6PH_T_F16 pParam;

void main(void)
{
  tFrac16 f16Shift;

  // Initialize GMCLIB_ClarkInv6Ph transformation parameters

  // input shift -30° (-pi/6) normalized into [-1, 1) interval
  f16Shift = FRAC16(-1.0/6.0);

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6PhInit_F16(&pParam, f16Shift);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6PhInit(&pParam, f16Shift,F16);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 16-bit fractional implementation is selected as default.
  GMCLIB_ClarkInv6PhInit(&pParam, f16Shift);

  while(1);
}

// Periodical function or interrupt
void loop(void)
{
  SWLIBS_6Syst_F16 pOutPwm;
  SWLIBS_2Syst_F16 pIn;

  // Read the A/D, Transform from 6-phase to d-q, PI controllers, Inverse Park transform etc.
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // (...)

  // Calculate alpha-beta values:

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6Ph_F16(&pOut, &pIn, &pParam);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6Ph(&pOut, &pIn, &pParam, F16);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 16-bit fractional implementation is selected as default.
  GMCLIB_ClarkInv6Ph(&pOut, &pIn, &pParam);
}
```

2.50.5 Function GMCLIB_ClarkInv6Ph_FLT

2.50.5.1 Declaration

```
void GMCLIB_ClarkInv6Ph_FLT(SWLIBS_6Syst_FLT *const pOut, const
SWLIBS_2Syst_FLT *const pIn, const GMCLIB_CLARKINV_6PH_T_FLT
*const pParam);
```

2.50.5.2 Arguments

Table 298. GMCLIB_ClarkInv6Ph_FLT arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_FLT *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector. Arguments of the structure contain single precision floating-point values.
const GMCLIB_CLARKINV_6PH_T_FLT *const	pParam	**output**	Pointer to the structure containing the transformation coefficient calculated by GMCLIB_ClarkInv6PhInit function. Arguments of the structure contain single precision floating-point values.

2.50.5.3 Implementation details

pOut Pointer to the structure containing calculated duty-cycle ratios of the six-phase machine consisting of two sets of starconnected three-phase stator windings fltA-fltB-fltC and fltD-fltE-fltF. Arguments of the structure contain single precision floating-point values.

The function presumes that the input voltages are normalized to fit the range <-1; 1>. In a typical motor control application, this function is preceded by the function GMCLIB_ElimDcBusRip_FLT which ensures that the voltages are correctly normalized.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.50.5.4 Code Example

```
#include "gmclib.h"

GMCLIB_CLARKINV_6PH_T_F16 pParam;

void main(void)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
{
  tFrac32 fltShift;

  // Initialize GMCLIB_ClarkInv6Ph transformation parameters

  // input shift -30° (-pi/6)
  fltShift = -FLOAT_PI_DIVBY_6;

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6PhInit_FLT(&pParam, fltShift);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6PhInit(&pParam, fltShift,FLT);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_ClarkInv6PhInit(&pParam, fltShift);

  while(1);
}

// Periodical function or interrupt
void loop(void)
{
  SWLIBS_6Syst_FLT pOutPwm;
  SWLIBS_2Syst_U32 pOutSectors;
  SWLIBS_2Syst_FLT pIn;

  // Read the A/D, Transform from 6-phase to d-q, PI controllers, Inverse Park transform etc.
  // (...)

  // Calculate alpha-beta values:

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6Ph_FLT(&pOut, &pIn, &pParam);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6Ph(&pOut, &pIn, &pParam, FLT);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if single precision floating point implementation selected as default
  GMCLIB_ClarkInv6Ph(&pOut, &pIn, &pParam);
}
```

2.50.6 Function GMCLIB_ClarkInv6PhInit

The function initializes the parameters structure required for inverse six phase Clarke transformation.

2.50.6.1 Description

GMCLIB_ClarkInv6PhInit calculates the transformation coefficients required for GMCLIB_ClarkInv6Ph function from the shift between two sets of star-connected three phase stator windings, according to the following equations:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$AlphaD = \cos(-\theta)$$
$$AlphaE = \cos\left(-\theta + \frac{2}{3}\pi\right)$$
$$AlphaF = \cos\left(-\theta - \frac{2}{3}\pi\right)$$
$$BetaD = \sin(-\theta)$$
$$BetaE = \sin\left(-\theta + \frac{2}{3}\pi\right)$$
$$BetaF = \sin\left(-\theta - \frac{2}{3}\pi\right)$$

Equation GMCLIB_ClarkInv6PhInit_Eq1

where θ is the shift between the two windings of the six-phase machine.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.50.6.2 Re-entrancy

The function is re-entrant for a different `pParam`.

2.50.6.3 Function GMCLIB_ClarkInv6PhInit_F32

2.50.6.3.1 Declaration

```
void GMCLIB_ClarkInv6PhInit_F32(GMCLIB_CLARKINV_6PH_T_F32 *const
pParam, tFrac32 f32WindingShift);
```

2.50.6.3.2 Arguments

Table 299. GMCLIB_ClarkInv6PhInit_F32 arguments

Type	Name	Direction	Description
tFrac32	f32WindingShift	**input**	Input argument is a 32-bit number that contains an angle in radians from interval [-π, π) normalized between [-1, 1). This angle represents the shift between two sets of star-connected three phase stator windings.
GMCLIB_CLARKINV_6PH_T_F32 *const	pParam	**output**	Pointer to the structure containing the transformation coefficient required for GMCLIB_ClarkInv6Ph function. Arguments of the structure contain fixed point 32-bit values.

2.50.6.3.3 Code Example

```
#include "gmclib.h"

GMCLIB_CLARKINV_6PH_T_F32 pParam;

void main(void)
{
  tFrac32 f32Shift;

  // Initialize GMCLIB_ClarkInv6Ph transformation parameters

  // input shift -30° (-pi/6) normalized into [-1, 1) interval
  f32Shift = FRAC32(-1.0/6.0);

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6PhInit_F32(&pParam, f32Shift);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6PhInit(&pParam, f32Shift,F32);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_ClarkInv6PhInit(&pParam, f32Shift);

  while(1);
}

// Periodical function or interrupt
void loop(void)
{
  SWLIBS_6Syst_F32 pOut;
  SWLIBS_2Syst_F32 pIn;

  // Read the A/D, Transform from 6-phase to d-q, PI controllers, Inverse Park transform etc.
  // (...)

  // Calculate alpha-beta values:

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6Ph_F32(&pOut, &pIn, &pParam);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_ClarkInv6Ph(&pOut, &pIn, &pParam, F32);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_ClarkInv6Ph(&pOut, &pIn, &pParam);
}
```

2.51 Function GMCLIB_DecouplingPMSM

This function calculates the cross-coupling voltages to eliminate the dq axis coupling causing on-linearity of the field oriented control.

2.51.1 Description

The quadrature phase model of a PMSM motor, in a synchronous reference frame, is very popular for field oriented control structures because both controllable quantities, current and voltage, are DC values. This allows employing only simple controllers to force the machine currents into the defined states.

The voltage equations of this model can be obtained by transforming the motor three phase voltage equations into a quadrature phase rotational frame, which is aligned and rotates synchronously with the rotor. Such a transformation, after some mathematical corrections, yields the following set of equations, describing the quadrature phase model of a PMSM motor, in a synchronous reference frame:

$$\begin{bmatrix} u_d \\ u_q \end{bmatrix} = R_S \cdot \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} L_d & 0 \\ 0 & L_q \end{bmatrix} \frac{d}{dt} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \omega_e \begin{bmatrix} -L_q & 0 \\ 0 & L_d \end{bmatrix} \begin{bmatrix} i_q \\ i_d \end{bmatrix} + \omega_e \psi_{pm} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Equation GMCLIB_DecouplingPMSM_Eq1

It can be seen that GMCLIB_DecouplingPMSM_Eq1 represents a non-linear cross dependent system. The linear voltage components cover the model of the phase winding, which is simplified to a resistance in series with inductance (R-L circuit). The cross-coupling components represent the mutual coupling between the two phases of the

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

quadrature phase model, and the back-EMF component (visible only in q-axis voltage) represents the generated back-EMF voltage caused by rotor rotation.

In order to achieve dynamic torque, speed and positional control, the non-linear and back-EMF components from GMCLIB_DecouplingPMSM_Eq1 must be compensated for. This will result in a fully decoupled flux and torque control of the machine and simplifies the PMSM motor model into two independent R-L circuit models as follows:

$$u_d = R_S i_d + L_d \frac{di_d}{dt}$$
$$u_q = R_S i_q + L_q \frac{di_q}{dt}$$

Equation GMCLIB_DecouplingPMSM_Eq2

Such a simplification of the PMSM model also greatly simplifies the design of both the d-q current controllers.

Therefore, it is advantageous to compensate for the cross-coupling terms in GMCLIB_DecouplingPMSM_Eq1, using the feed-forward voltages u_dq_comp given from GMCLIB_DecouplingPMSM_Eq1 as follows:

$$u_{d_{comp}} = -\omega_e L_q i_q$$
$$u_{q_{comp}} = \omega_e L_d i_d$$

Equation GMCLIB_DecouplingPMSM_Eq3

The feed-forward voltages u_dq_comp are added to the voltages generated by the current controllers u_dq, which cover the R-L model. The resulting voltages represent the direct u_{q_dec} and quadrature u_{q_decq} components of the decoupled voltage vector that is to be applied on the motor terminals (using a pulse width modulator). The back-EMF voltage component is already considered to be compensated by an external function.

The function GMCLIB_DecouplingPMSM calculates the cross-coupling voltages u_dq_comp and adds these to the input u_dq voltage vector. Because the back-EMF voltage component is considered compensated, this component is equal to zero. Therefore, calculations performed by GMCLIB_DecouplingPMSM are derived from these two equations:

$$u_{d_{dec}} = u_d + u_{d_{comp}}$$
$$u_{q_{dec}} = u_q + u_{q_{comp}}$$

Equation GMCLIB_DecouplingPMSM_Eq4

where u_dq is the voltage vector calculated by the controllers (with the already compensated back-EMF component), u_dq_comp is the feed-forward compensating voltage vector described in GMCLIB_DecouplingPMSM_Eq3, and u_dq_dec is the resulting decoupled voltage vector to be applied on the motor terminals.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.51.2 Re-entrancy

The function is re-entrant.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.51.3 Function GMCLIB_DecouplingPMSM_F32

2.51.3.1 Declaration

```
void GMCLIB_DecouplingPMSM_F32(SWLIBS_2Syst_F32 *const pUdqDec,
const SWLIBS_2Syst_F32 *const pUdq, const SWLIBS_2Syst_F32 *const
pIdq, tFrac32 f32AngularVel, const GMCLIB_DECOUPLINGPMSM_T_F32
*const pParam);
```

2.51.3.2 Arguments

Table 300. GMCLIB_DecouplingPMSM_F32 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F32 *const	pUdqDec	**output**	Pointer to the structure containing direct (u_{df_dec}) and quadrature (u_{qf_dec}) components of the decoupled stator voltage vector to be applied on the motor terminals.
const SWLIBS_2Syst_F32 *const	pUdq	**input**	Pointer to the structure containing direct (u_{df}) and quadrature (u_{qf}) components of the stator voltage vector generated by the current controllers.
const SWLIBS_2Syst_F32 *const	pIdq	**input**	Pointer to the structure containing direct (i_{df}) and quadrature (i_{qf}) components of the stator current vector measured on the motor terminals.
tFrac32	f32AngularVel	**input**	Rotor angular velocity in rad/sec, referred to as (ω_{ef}) in the detailed section of the documentation.
const GMCLIB_DECOUPLINGPMSM_T_F32 *const	pParam	**input**	Pointer to the structure containing k_{df} and k_{qf} coefficients (see the detailed section of the documentation) and scale parameters (k_{d_shift}) and (k_{q_shift}).

2.51.3.3 Implementation details

Substituting GMCLIB_DecouplingPMSM_eq3 into GMCLIB_DecouplingPMSM_eq4, and normalizing GMCLIB_DecouplingPMSM_eq4, results in the following set of equations:

$$udf_{dec} \cdot U_{max} = u_{df} \cdot U_{max} - \omega_{ef} \cdot \Omega_{max} \cdot L_q \cdot i_{qf} \cdot I_{max}$$
$$uqf_{dec} \cdot U_{max} = u_{qf} \cdot U_{max} + \omega_{ef} \cdot \Omega_{max} \cdot L_d \cdot i_{df} \cdot I_{max}$$

Equation GMCLIB_DecouplingPMSM_F32_Eq1

where subscript f denotes the fractional representation of the respective quantity, and U_{max}, I_{max}, Ω_{max} are the maximal values (scale values) for the voltage, current and angular velocity respectively.

Real quantities are converted to the fractional range [-1, 1) using the following equations:

$$udf_{dec} = \frac{ud_{dec}}{U_{max}}, \quad uqf_{dec} = \frac{uq_{dec}}{U_{max}}, \quad u_{df} = \frac{u_d}{U_{max}}, \quad u_{qf} = \frac{u_q}{U_{max}},$$
$$i_{df} = \frac{i_d}{I_{max}}, \quad i_{qf} = \frac{i_q}{I_{max}}, \quad \omega_{ef} = \frac{\omega_e}{\Omega_{max}}$$

Equation GMCLIB_DecouplingPMSM_F32_Eq2

Further, rearranging GMCLIB_DecouplingPMSM_F32_Eq1 results in:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$udf_{dec} = u_{df} - \omega_{ef} \cdot i_{qf} \cdot \frac{\Omega_{max} \cdot L_q \cdot I_{max}}{U_{max}} = u_{df} - \omega_{ef} \cdot i_{qf} \cdot k_q$$

$$uqf_{dec} = u_{qf} + \omega_{ef} \cdot i_{df} \cdot \frac{\Omega_{max} \cdot L_d \cdot I_{max}}{U_{max}} = u_{qf} + \omega_{ef} \cdot i_{df} \cdot k_d$$

Equation GMCLIB_DecouplingPMSM_F32_Eq3

where k_d and k_q are coefficients calculated as:

$$k_d = L_d \cdot \Omega_{max} \cdot \frac{I_{max}}{U_{max}}$$

$$k_q = L_q \cdot \Omega_{max} \cdot \frac{I_{max}}{U_{max}}$$

Equation GMCLIB_DecouplingPMSM_F32_Eq4

Because function GMCLIB_DecouplingPMSM_F32 is implemented using the fractional arithmetic, both the k_d and k_q coefficients also have to be scaled to fit into the fractional range [-1, 1). For that purpose, two additional scaling coefficients are defined as:

$$k_{d_shift} = ceil\left(\frac{log(k_d)}{log(2)}\right)$$

$$k_{q_shift} = ceil\left(\frac{log(k_q)}{log(2)}\right)$$

Equation GMCLIB_DecouplingPMSM_F32_Eq5

Using scaling coefficients GMCLIB_DecouplingPMSM_F32_Eq5, the fractional representation of coefficients k_d and k_q from GMCLIB_DecouplingPMSM_F32_Eq4 are derived as follows:

$$k_{df} = k_d \cdot 2^{-k_{d_shift}}$$

$$k_{qf} = k_q \cdot 2^{-k_{q_shift}}$$

Equation GMCLIB_DecouplingPMSM_F32_Eq6

Substituting GMCLIB_DecouplingPMSM_F32_Eq4 - GMCLIB_DecouplingPMSM_F32_Eq6 into GMCLIB_DecouplingPMSM_F32_Eq3 results in the final form of the equation set, actually implemented in the GMCLIB_DecouplingPMSM_F32 function:

$$udf_{dec} = u_{df} - \omega_{ef} \cdot i_{qf} \cdot k_{qf} \cdot 2^{k_{q_shift}}$$

$$uqf_{dec} = u_{qf} + \omega_{ef} \cdot i_{df} \cdot k_{df} \cdot 2^{k_{d_shift}}$$

Equation GMCLIB_DecouplingPMSM_F32_Eq7

Scaling of both equations into the fractional range is done using a multiplication by $2^{k_d_shift}$, $2^{k_q_shift}$, respectively. Therefore, it is implemented as a simple left shift with overflow protection.

Note: *All parameters can be reset during declaration using the* GMCLIB_DECOUPLINGPMSM_DEFAULT_F32 *macro.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.51.3.4 Code Example

```
#include "gmclib.h"

#define L_D      (50.0e-3)   // Ld inductance = 50mH
#define L_Q      (100.0e-3)  // Lq inductance = 100mH
#define U_MAX    (50.0)      // scale for voltage = 50V
#define I_MAX    (10.0)      // scale for current = 10A
#define W_MAX    (2000.0)    // scale for angular velocity = 2000rad/sec

GMCLIB_DECOUPLINGPMSM_T_F32 f32trDec = GMCLIB_DECOUPLINGPMSM_DEFAULT_F32;
SWLIBS_2Syst_F32 f32trUDQ;
SWLIBS_2Syst_F32 f32trIDQ;
SWLIBS_2Syst_F32 f32trUDecDQ;
tFrac32 f32We;

void main(void)
{
  // input values - scaling coefficients of given decoupling algorithm
  f32trDec.f32Kd = FRAC32(0.625);
  f32trDec.s16KdShift = (tS16)6;
  f32trDec.f32Kq = FRAC32(0.625);
  f32trDec.s16KqShift = (tS16)5;
  // d quantity of input voltage vector 5[V]
  f32trUDQ.f32Arg1 = FRAC32(5.0/U_MAX);
  // q quantity of input voltage vector 10[V]
  f32trUDQ.f32Arg2 = FRAC32(10.0/U_MAX);
  // d quantity of measured current vector 6[A]
  f32trIDQ.f32Arg1 = FRAC32(6.0/I_MAX);
  // q quantity of measured current vector 4[A]
  f32trIDQ.f32Arg2 = FRAC32(4.0/I_MAX);
  // rotor angular velocity
  f32We = FRAC32(100.0/W_MAX);

  // output should be
  // f32trUDecDQ.f32Arg1 ~ 0xA6666666 ~ FRAC32(-0.7)*50V ~=-35[V]
  // f32trUDecDQ.f32Arg2 ~ 0x66666666 ~ FRAC32(0.8)*50V ~=40[V]
  GMCLIB_DecouplingPMSM_F32(&f32trUDecDQ,&f32trUDQ,&f32trIDQ,f32We,
                            &f32trDec);

  // output should be
  // f32trUDecDQ.f32Arg1 ~ 0xA6666666 ~ FRAC32(-0.7)*50V ~=-35[V]
  // f32trUDecDQ.f32Arg2 ~ 0x66666666 ~ FRAC32(0.8)*50V ~=40[V]
  GMCLIB_DecouplingPMSM(&f32trUDecDQ,&f32trUDQ,&f32trIDQ,f32We,&f32trDec,
                        F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be
  // f32trUDecDQ.f32Arg1 ~ 0xA6666666 ~ FRAC32(-0.7)*50V ~=-35[V]
  // f32trUDecDQ.f32Arg2 ~ 0x66666666 ~ FRAC32(0.8)*50V ~=40[V]
  GMCLIB_DecouplingPMSM(&f32trUDecDQ,&f32trUDQ,&f32trIDQ,f32We,&f32trDec);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.51.4 Function GMCLIB_DecouplingPMSM_F16

2.51.4.1 Declaration

```
void GMCLIB_DecouplingPMSM_F16(SWLIBS_2Syst_F16 *const pUdqDec,
const SWLIBS_2Syst_F16 *const pUdq, const SWLIBS_2Syst_F16 *const
pIdq, tFrac16 f16AngularVel, const GMCLIB_DECOUPLINGPMSM_T_F16
*const pParam);
```

2.51.4.2 Arguments

Table 301. GMCLIB_DecouplingPMSM_F16 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *const	pUdqDec	**output**	Pointer to the structure containing direct (u_{df_dec}) and quadrature (u_{qf_dec}) components of the decoupled stator voltage vector to be applied on the motor terminals.
const SWLIBS_2Syst_F16 *const	pUdq	**input**	Pointer to the structure containing direct (u_{df}) and quadrature (u_{qf}) components of the stator voltage vector generated by the current controllers.
const SWLIBS_2Syst_F16 *const	pIdq	**input**	Pointer to the structure containing direct (i_{df}) and quadrature (i_{qf}) components of the stator current vector measured on the motor terminals.
tFrac16	f16AngularVel	**input**	Rotor angular velocity in rad/sec, referred to as ($omega_{ef}$) in the detailed section of the documentation.
const GMCLIB_DECOUPLINGPMSM_T_F16 *const	pParam	**input**	Pointer to the structure containing k_{df} and k_{qf} coefficients (see the detailed section of the documentation) and scale parameters (k_{d_shift}) and (k_{q_shift}).

2.51.4.3 Implementation details

Substituting GMCLIB_DecouplingPMSM_eq3 into GMCLIB_DecouplingPMSM_eq4, and normalizing GMCLIB_DecouplingPMSM_eq4, results in the following set of equations:

$$udf_{dec} \cdot U_{max} = u_{df} \cdot U_{max} - \omega_{ef} \cdot \Omega_{max} \cdot L_q \cdot i_{qf} \cdot I_{max}$$
$$uqf_{dec} \cdot U_{max} = u_{qf} \cdot U_{max} + \omega_{ef} \cdot \Omega_{max} \cdot L_d \cdot i_{df} \cdot I_{max}$$

Equation GMCLIB_DecouplingPMSM_F16_Eq1

where subscript f denotes the fractional representation of the respective quantity, and U_{max}, I_{max}, Ω_{max} are the maximal values (scale values) for the voltage, current and angular velocity respectively.

Real quantities are converted to the fractional range [-1, 1) using the following equations:

$$udf_{dec} = \frac{ud_{dec}}{U_{max}}, \quad uqf_{dec} = \frac{uq_{dec}}{U_{max}}, \quad u_{df} = \frac{u_d}{U_{max}}, \quad u_{qf} = \frac{u_q}{U_{max}},$$
$$i_{df} = \frac{i_d}{I_{max}}, \quad i_{qf} = \frac{i_q}{I_{max}}, \quad \omega_{ef} = \frac{\omega_e}{\Omega_{max}}$$

Equation GMCLIB_DecouplingPMSM_F16_Eq2

Further, rearranging GMCLIB_DecouplingPMSM_F16_Eq1 results in:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$udf_{dec} = u_{df} - \omega_{ef} \cdot i_{qf} \cdot \frac{\Omega_{max} \cdot L_q \cdot I_{max}}{U_{max}} = u_{df} - \omega_{ef} \cdot i_{qf} \cdot k_q$$

$$uqf_{dec} = u_{qf} + \omega_{ef} \cdot i_{df} \cdot \frac{\Omega_{max} \cdot L_d \cdot I_{max}}{U_{max}} = u_{qf} + \omega_{ef} \cdot i_{df} \cdot k_d$$

Equation GMCLIB_DecouplingPMSM_F16_Eq3

where k_d and k_q are coefficients calculated as:

$$k_d = L_d \cdot \Omega_{max} \cdot \frac{I_{max}}{U_{max}}$$

$$k_q = L_q \cdot \Omega_{max} \cdot \frac{I_{max}}{U_{max}}$$

Equation GMCLIB_DecouplingPMSM_F16_Eq4

Because function GMCLIB_DecouplingPMSM_F16 is implemented using the fractional arithmetic, both the k_d and k_q coefficients also have to be scaled to fit into the fractional range [-1, 1). For that purpose, two additional scaling coefficients are defined as:

$$k_{d_shift} = ceil\left(\frac{log(k_d)}{log(2)}\right)$$

$$k_{q_shift} = ceil\left(\frac{log(k_q)}{log(2)}\right)$$

Equation GMCLIB_DecouplingPMSM_F16_Eq5

Using scaling coefficients GMCLIB_DecouplingPMSM_F16_Eq5, the fractional representation of coefficients k_d and k_q from GMCLIB_DecouplingPMSM_F16_Eq4 are derived as follows:

$$k_{df} = k_d \cdot 2^{-k_{d_shift}}$$

$$k_{qf} = k_q \cdot 2^{-k_{q_shift}}$$

Equation GMCLIB_DecouplingPMSM_F16_Eq6

Substituting GMCLIB_DecouplingPMSM_F16_Eq4 - GMCLIB_DecouplingPMSM_F16_Eq6 into GMCLIB_DecouplingPMSM_F16_Eq3 results in the final form of the equation set, actually implemented in the GMCLIB_DecouplingPMSM_F16 function:

$$udf_{dec} = u_{df} - \omega_{ef} \cdot i_{qf} \cdot k_{qf} \cdot 2^{k_{q_shift}}$$

$$uqf_{dec} = u_{qf} + \omega_{ef} \cdot i_{df} \cdot k_{df} \cdot 2^{k_{d_shift}}$$

Equation GMCLIB_DecouplingPMSM_F16_Eq7

Scaling of both equations into the fractional range is done using a multiplication by $2^{k_d_shift}$, $2^{k_q_shift}$, respectively. Therefore, it is implemented as a simple left shift with overflow protection.

Note: *All parameters can be reset during declaration using the* GMCLIB_DECOUPLINGPMSM_DEFAULT_F16 *macro.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.51.4.4 Code Example

```
#include "gmclib.h"

#define L_D      (50.0e-3)   // Ld inductance = 50mH
#define L_Q      (100.0e-3)  // Lq inductance = 100mH
#define U_MAX    (50.0)      // scale for voltage = 50V
#define I_MAX    (10.0)      // scale for current = 10A
#define W_MAX    (2000.0)    // scale for angular velocity = 2000rad/sec

GMCLIB_DECOUPLINGPMSM_T_F16 f16trDec = GMCLIB_DECOUPLINGPMSM_DEFAULT_F16;
SWLIBS_2Syst_F16 f16trUDQ;
SWLIBS_2Syst_F16 f16trIDQ;
SWLIBS_2Syst_F16 f16trUDecDQ;
tFrac16 f16We;

void main(void)
{
  // input values - scaling coefficients of given decoupling algorithm
  f16trDec.f16Kd = FRAC16(0.625);
  f16trDec.s16KdShift = (tS16)6;
  f16trDec.f16Kq = FRAC16(0.625);
  f16trDec.s16KqShift = (tS16)5;
  // d quantity of input voltage vector 5[V]
  f16trUDQ.f16Arg1 = FRAC16(5.0/U_MAX);
  // q quantity of input voltage vector 10[V]
  f16trUDQ.f16Arg2 = FRAC16(10.0/U_MAX);
  // d quantity of measured current vector 6[A]
  f16trIDQ.f16Arg1 = FRAC16(6.0/I_MAX);
  // q quantity of measured current vector 4[A]
  f16trIDQ.f16Arg2 = FRAC16(4.0/I_MAX);
  // rotor angular velocity
  f16We = FRAC16(100.0/W_MAX);

  // output should be
  // f16trUDecDQ.f16Arg1 ~ 0xA666 ~ FRAC16(-0.7)*50V ~=-35[V]
  // f16trUDecDQ.f16Arg2 ~ 0x6666 ~ FRAC16(0.8)*50V ~=40[V]
  GMCLIB_DecouplingPMSM_F16(&f16trUDecDQ,&f16trUDQ,&f16trIDQ,f16We,
                            &f16trDec);

  // output should be
  // f16trUDecDQ.f16Arg1 ~ 0xA666 ~ FRAC16(-0.7)*50V ~=-35[V]
  // f16trUDecDQ.f16Arg2 ~ 0x6666 ~ FRAC16(0.8)*50V ~=40[V]
  GMCLIB_DecouplingPMSM(&f16trUDecDQ,&f16trUDQ,&f16trIDQ,f16We,&f16trDec,
                        F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be
  // f16trUDecDQ.f16Arg1 ~ 0xA666 ~ FRAC16(-0.7)*50V ~=-35[V]
  // f16trUDecDQ.f16Arg2 ~ 0x6666 ~ FRAC16(0.8)*50V ~=40[V]
  GMCLIB_DecouplingPMSM(&f16trUDecDQ,&f16trUDQ,&f16trIDQ,f16We,&f16trDec);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.51.5 Function GMCLIB_DecouplingPMSM_FLT

2.51.5.1 Declaration

```
void GMCLIB_DecouplingPMSM_FLT(SWLIBS_2Syst_FLT *const pUdqDec,
const SWLIBS_2Syst_FLT *const pUdq, const SWLIBS_2Syst_FLT *const
pIdq, tFloat fltAngularVel, const GMCLIB_DECOUPLINGPMSM_T_FLT
*const pParam);
```

2.51.5.2 Arguments

Table 302. GMCLIB_DecouplingPMSM_FLT arguments

Type	Name	Direction	Description
SWLIBS_2Syst_FLT *const	pUdqDec	**output**	Pointer to the structure containing direct (u_{df_dec}) and quadrature (u_{qf_dec}) components of the decoupled stator voltage vector to be applied on the motor terminals.
const SWLIBS_2Syst_FLT *const	pUdq	**input**	Pointer to the structure containing direct (u_{df}) and quadrature (u_{qf}) components of the stator voltage vector generated by the current controllers.
const SWLIBS_2Syst_FLT *const	pIdq	**input**	Pointer to the structure containing direct (i_{df}) and quadrature (i_{qf}) components of the stator current vector measured on the motor terminals.
tFloat	fltAngularVel	**input**	Rotor angular velocity in rad/sec, referred to as ($omega_{ef}$) in the detailed section of the documentation.
const GMCLIB_DECOUPLINGPMSM_T_FLT *const	pParam	**input**	Pointer to the structure containing L_D and L_Q coefficients (see the detailed section of the documentation).

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

All parameters can be reset during declaration using the GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT *macro. All inputs and parameters contain single precision floating point data type values.*

2.51.5.3 Code Example

```
#include "gmclib.h"

GMCLIB_DECOUPLINGPMSM_T_FLT flttrDec = GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT;
SWLIBS_2Syst_FLT flttrUDQ;
SWLIBS_2Syst_FLT flttrIDQ;
SWLIBS_2Syst_FLT flttrUDecDQ;
tFloat fltWe;

void main(void)
{
  // input values
  flttrDec.fltLD = (tFloat) 50e-3; // LD inductance = 50mH
  flttrDec.fltLQ = (tFloat) 100e-3; // LQ inductance = 100mH
  // D quantity of input voltage vector 5[V]
  flttrUDQ.fltArg1 = (tFloat) 5.0;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // Q quantity of input voltage vector 10[V]
  flttrUDQ.fltArg2 = (tFloat) 10.0;
  // D quantity of measured current vector 6[A]
  flttrIDQ.fltArg1 = (tFloat) 6.0;
  // Q quantity of measured current vector 4[A]
  flttrIDQ.fltArg2 = (tFloat) 4.0;
  fltWe = (tFloat) 100.0; // rotor angular velocity

  // output should be
  // flttrUDecDQ.fltArg1 ~= -35[V]
  // flttrUDecDQ.fltArg2 ~= 40[V]
  GMCLIB_DecouplingPMSM_FLT(&flttrUDecDQ,&flttrUDQ,&flttrIDQ,fltWe,
                            &flttrDec);

  // output should be
  // flttrUDecDQ.fltArg1 ~= -35[V]
  // flttrUDecDQ.fltArg2 ~= 40[V]
  GMCLIB_DecouplingPMSM(&flttrUDecDQ,&flttrUDQ,&flttrIDQ,fltWe,&flttrDec,
                        FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output should be
  // flttrUDecDQ.fltArg1 ~= -35[V]
  // flttrUDecDQ.fltArg2 ~= 40[V]
  GMCLIB_DecouplingPMSM(&flttrUDecDQ,&flttrUDQ,&flttrIDQ,fltWe,&flttrDec);
}
```

2.52 Function GMCLIB_DoubleSwitching_F16

This function calculates the PWM edges and ADC triggers for Double-switching technique used in single-shunt current reconstruction applications.

2.52.1 Declaration

```
void
GMCLIB_DoubleSwitching_F16(GMCLIB_DOUBLESWITCHING_3PH_EDGES_T_F16
*const pOutEdges, GMCLIB_DOUBLESWITCHING_TRIGGERS_T_F16 *const
pOutTriggers, const SWLIBS_3Syst_F16 *const pInPwm, tU16
u16Sector, const GMCLIB_DOUBLESWITCHING_T_F16 *const pParam);
```

2.52.2 Arguments

Table 303. GMCLIB_DoubleSwitching_F16 arguments

Type	Name	Direction	Description
const SWLIBS_3Syst_F16 *const	pInPwm	**input**	Pointer to the structure containing three-phase PWM duty cycles calculated by the function GMCLIB_SvmStd_F16 or another PWM generating function (see GMCLIB for more options). The arguments of the structure contain fixed-point 16-bit values and must be in the fractional range [0, 1).

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 303. GMCLIB_DoubleSwitching_F16 arguments...*continued*

Type	Name	Direction	Description
tU16	u16Sector	**input**	Unsigned 16-bit integer value indicating the SVM sector of the alpha/beta plane in which the voltage vector is placed.
const GMCLIB_DOUBLESWITCHING_T_F16 *const	pParam	**input**	Pointer to the structure containing parameters for the double switching algorithm.
GMCLIB_DOUBLESWITCHING_3PH_EDGES_T_F16 *const	pOutEdges	**output**	Pointer to the structure of the PWM rising/falling edges of each of the PWM channels corresponding to the three-phase voltages. Each PWM channel is defined by two rising and two falling edges (see picture GMCLIB_DoubleSwitching_F16_fig4). The position of each edge is determined by a 16-bit fixed-point number in the fractional range [0, 1). This range corresponds to one PWM period.
GMCLIB_DOUBLESWITCHING_TRIGGERS_T_F16 *const	pOutTriggers	**output**	Pointer to the structure of four ADC triggers for the triggering unit, stored as 16-bit unsigned integers in the range [0, 32767]. This range corresponds to one PWM period.

2.52.3 Description

Double-switching technique is used to reconstruct the phase currents from DC-link current sensor in three-phase AC applications.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 98. DC-link current sensor.

Following figure shows how the DC-link current is linked to the phase currents. It can be seen that in most of the time, at least two phase currents are available on the DC-link sensor. Considering symmetrical 3-phase system, the third current can be calculated out of two.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 99. Dependency of the DC-link current on phase currents.

Unfortunately, there are casese when two or all three edges of the PWM signals are overlapping or simply too close to each other, so the "window" of a phase current is too short to be sampled by ADC. To "re-open" the window again, the double-switching feature is used.

For double-switching, signals are calculated based on the 3-phase reference system, sinusoidal, space vector modulated (SVM) or modulated with other techniques.

Figure 100. Overlapping edges of the PWM signals.

The key idea is to split one of the PWM patterns in half and shift the two halves apart. This way, the outer edges would no longer overlap with another PWM, opening the sampling window for the second current.

In order to get the right split in the PWM pattern, minimal sampling pulse (for ADC sampling), minimal zero pulse (to allow MOSFETs to fully switch on and off) and deadtime (to consider additional delay) need to be included. Unfortunately, the drawback is another active vector created in the middle. To minimize the impact, all three PWM signals are split in half, delivering two short active vectors and one passive (null) vector to the motor (load). This way, the negative impact is less significant and the performance is better.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Normally, the application uses complementary pairs of MOSFETs, leading to a need for deadtime insertion (which is usually done automatically by a dedicated timer or modulator). Nevertheless, the edges of the double-switched pattern have to be calculated, considering all the delays required by the application (deadtime, minimal sampling time of the ADC - more specifically, time between two samples). When inserting the null vector, full switch-off and switch-on sequence of MOSFETs should be considered as the minimal zero pulse.

Figure 101. Split PWM patterns.

In order to use this algorithm, the following parameters must be defined as a fraction of the PWM period normalized in [0,1) interval.

Figure 102. Split PWM patterns.

$$deadtime_f = deadtime * f_{PWM}$$

$$minSamplingTime_f = minSamplingTime * f_{PWM}$$

$$sampleNhold_f = sampleNhold * f_{PWM}$$

$$minZeroPulse_f = minZeroPulse * f_{PWM}$$

Equation GMCLIB_DoubleSwitching_F16_Eq1

where f_{PWM} is PWM frequency in Hz, minZeroPulse determines the minimal width of pulse in seconds inserted in the middle of the PWM period including at least one deadtime, sampleNhold is the minimal "sample&hold" time in seconds of the ADC and minSamplingTime is a sum of "sample&hold" time and "successive approximation" time in seconds. MinSamplingTime should also include delays such as ADC pre-trigger delay or delays between two subsequent triggers caused by the triggering unit.

Since the algorithm inserts some pulses into the PWM pattern, it is clear that there will be certain loss of duty cycle involved. Amplitude of the input signal must be limited by following equation:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$A_{MAX} = 100 - 100 \cdot \left(4 \cdot deadtime_f + minSamplingPulse_f + minZeroPulse_f\right) \ [\%]$$

Equation GMCLIB_DoubleSwitching_F16_Eq2

Note: *In certain duty cycles, there is a risk, that the last ADC trigger will cause the ADC conversion time to end in the next period of the double-switching PWM. In such cases, it is recommended to set the triggers only for every odd period of the PWM. In case the application requires ADC sampling each PWM period, then it is possible to trigger the ADC only with the first two triggers or to reduce the duty cycle by minSamplingPulse in order to avoid ADC sampling colision.*

2.52.4 Code Example

```
#include "gmclib.h"

void main(void)
{
    SWLIBS_3Syst_F16 inPwm;
    tU16 u16Sector;
    GMCLIB_DOUBLESWITCHING_T_F16 params;
    GMCLIB_DOUBLESWITCHING_3PH_EDGES_T_F16 outEdges;
    GMCLIB_DOUBLESWITCHING_TRIGGERS_T_F16 outTriggers;

    inPwm.f16Arg1 = FRAC16(0.5);
    inPwm.f16Arg2 = FRAC16(0.75);
    inPwm.f16Arg3 = FRAC16(0.25);
    u16Sector = 2u;
    //Example for 20kHz PWM frequency, 2.5us minSamplingPulse, 2us minZeroPulse
    //and 1us sampleNhold
    params.f16MinZeroPulse     = FRAC16(0.04);
    params.f16MinSamplingPulse = FRAC16(0.05);
    params.f16SampleNhold      = FRAC16(0.02);

     GMCLIB_DoubleSwitching_F16(&outEdges,&outTriggers,&inPwm,u16Sector,&params);

     GMCLIB_DoubleSwitching(&outEdges,&outTriggers,&inPwm,u16Sector,&params,F16);

    // ##########################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##########################################################

    GMCLIB_DoubleSwitching(&outEdges,&outTriggers,&inPwm,u16Sector,&params);
}
```

2.53 Function GMCLIB_ElimDcBusRip

This function implements the DC Bus voltage ripple elimination.

2.53.1 Description

The GMCLIB_ElimDcBusRip function provides a computational method for the recalculation of the direct (α) and quadrature (β) components of the required stator voltage vector, so as to compensate for voltage ripples on the DC bus of the power stage.

Considering a cascaded type structure of the control system in a standard motor control application, the required voltage vector to be applied on motor terminals is generated

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

by a set of controllers (usually P, PI or PID) only with knowledge of the maximal value of the DC bus voltage. The amplitude and phase of the required voltage vector are then used by the pulse width modulator (PWM) for generation of appropriate duty-cycles for the power inverter switches. The amplitude of the generated phase voltage (averaged across one switching period) does not only depend on the actual on/off times of the given phase switches and the maximal value of the DC bus voltage. The actual amplitude of the phase voltage is also directly affected by the actual value of the available DC bus voltage. Therefore, any variations in amplitude of the actual DC bus voltage must be accounted for by modifying the amplitude of the required voltage so that the output phase voltage remains unaffected.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.53.2 Re-entrancy

The function is re-entrant.

2.53.3 Function GMCLIB_ElimDcBusRip_F32

2.53.3.1 Declaration

```
void GMCLIB_ElimDcBusRip_F32(SWLIBS_2Syst_F32 *const pOut, const
SWLIBS_2Syst_F32 *const pIn, const GMCLIB_ELIMDCBUSRIP_T_F32
*const pParam);
```

2.53.3.2 Arguments

Table 304. GMCLIB_ElimDcBusRip_F32 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F32 *const	pOut	**output**	Pointer to the structure with direct (α) and quadrature (β) components of the required stator voltage vector re-calculated so as to compensate for voltage ripples on the DC bus.
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure with direct (α) and quadrature (β) components of the required stator voltage vector before compensation of voltage ripples on the DC bus.
const GMCLIB_ELIMDCBUSRIP_T_F32 *const	pParam	**input**	Pointer to the parameters structure.

2.53.3.3 Return

Function returns no value.

2.53.3.4 Implementation details

For better understanding, let's consider the following two simple examples:

Example 1:

- amplitude of the required phase voltage U_{reg}=50[V]
- maximal amplitude of the DC bus voltage $U_{DC_BUS_MAX}$=100[V]
- actual amplitude of the DC bus voltage $U_{DC_BUS_ACTUAL}$=100[V]

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

- voltage to be applied to the PWM modulator to generate U_{reg}=50[V] on the inverter phase output:

$$U_{req_new} = \frac{U_{req} U_{DC_BUS_MAX}}{U_{DC_BUS_ACTUAL}} = 50V$$

Equation GMCLIB_ElimDcBusRip_F32_Eq1

Example 2:

- amplitude of the required phase voltage U_{reg}=50[V]
- maximal amplitude of the DC bus voltage $U_{DC_BUS_MAX}$=100[V]
- actual amplitude of the DC bus voltage $U_{DC_BUS_ACTUAL}$=90[V]
- voltage to be applied to the PWM modulator to generate U_{reg}=50[V] on the inverter phase output:

$$U_{req_new} = \frac{U_{req} U_{DC_BUS_MAX}}{U_{DC_BUS_ACTUAL}} = 55.5V$$

Equation GMCLIB_ElimDcBusRip_F32_Eq2

The imperfections of the DC bus voltage are compensated for by the modification of amplitudes of the direct- α and the quadrature- β components of the stator reference voltage vector. The following formulas are used:

$$u_\alpha^* = \begin{cases} \frac{f32ModIndex \cdot u_\alpha}{f32ArgDCBusMsr/2} & if \quad abs(f32ModIndex \cdot u_\alpha) < \frac{f32ArgDCBusMsr}{2} \\ sign(u_\alpha) & otherwise \end{cases}$$

$$u_\beta^* = \begin{cases} \frac{f32ModIndex \cdot u_\beta}{f32ArgDCBusMsr/2} & if \quad abs(f32ModIndex \cdot u_\beta) < \frac{f32ArgDCBusMsr}{2} \\ sign(u_\beta) & otherwise \end{cases}$$

Equation GMCLIB_ElimDcBusRip_F32_Eq3

where: f32ModIndex is the inverse modulation index, f32ArgDcBusMsr is the measured DC bus voltage, the u_α and u_β are the input voltages, and the u_α* and u_β* are the output duty-cycle ratios.

The f32ModIndex and f32ArgDcBusMsr are supplied to the function within the parameters structure through its members. The u_α, u_β correspond respectively to the `f32Arg1` and `f32Arg2` members of the input structure, and the u_α* and u_β* respectively to the `f32Arg1` and `f32Arg2` members of the output structure.

The inverse modulation index (see the parameters structure `pParam`, the `f32ModIndex` member) must be equal to or greater than zero. Besides this restriction, the f32ModIndex must be set to a valid value resulting from the use of Space Vector Modulation techniques.

Table 305. Inverse modulation index values for various SVM techniques

SVM technique	Inverse modulation index (f32ModIndex)
#GMCLIB_SvmU7n	sqrt(3)/2, use FRAC32(0.8660254037844386)
#GMCLIB_PwmIct	1, use FRAC32(1)
#GMCLIB_SvmStd	sqrt(3)/2, use FRAC32(0.8660254037844386)
#GMCLIB_SvmU0n	sqrt(3)/2, use FRAC32(0.8660254037844386)
#GMCLIB_SvmSci	sqrt(3)/2, use FRAC32(0.8660254037844386)

The function explicitly avoids the case of division by zero. If f32ArgDcBusMsr equals zero, the outputs will be saturated.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Note: *Both the inverse modulation index `pIn->f32ModIndex` and the measured DC bus voltage `pIn->f32DcBusMsr` must be equal to or greater than 0, otherwise the results are undefined.*

2.53.3.5 Code Example

```
#include "gmclib.h"

#define U_MAX   (36.0) // voltage scale
SWLIBS_2Syst_F32 f32AB;
SWLIBS_2Syst_F32 f32OutAB;
GMCLIB_ELIMDCBUSRIP_T_F32 f32trMyElimDcBusRip =
  GMCLIB_ELIMDCBUSRIP_DEFAULT_F32;

void main(void)
{
  // inverse modulation coefficient for standard space vector modulation
  f32trMyElimDcBusRip.f32ModIndex = FRAC32(0.866025403784439);
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  f32AB.f32Arg1 = FRAC32(12.99/U_MAX);
  // beta component of input voltage vector = 7.5[V]
  f32AB.f32Arg2 = FRAC32(7.5/U_MAX);
  // value of the measured DC bus voltage 17[V]
  f32trMyElimDcBusRip.f32ArgDcBusMsr = FRAC32(17.0/U_MAX);

  // output alpha component of the output vector should be
  // f32OutAB.f32Arg1 = (12.99/36)*0.8660/(17.0/36/2) =
  //                    1.3235 -> FRAC32(1.0) ~ 0x7FFFFFFF
  // output beta component of the output vector should be
  // f32OutAB.f32Arg2 = (7.5/36)*0.8660/(17.0/36/2) =
  //                    0.7641 -> FRAC32(0.7641) ~ 0x61CF8000
  GMCLIB_ElimDcBusRip_F32(&f32OutAB,&f32AB,&f32trMyElimDcBusRip);

  // output alpha component of the output vector should be
  // f32OutAB.f32Arg1 = (12.99/36)*0.8660/(17.0/36/2) =
  //                    1.3235 -> FRAC32(1.0) ~ 0x7FFFFFFF
  // output beta component of the output vector should be
  // f32OutAB.f32Arg2 = (7.5/36)*0.8660/(17.0/36/2) =
  //                    0.7641 -> FRAC32(0.7641) ~ 0x61CF8000
  GMCLIB_ElimDcBusRip(&f32OutAB,&f32AB,&f32trMyElimDcBusRip,F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output alpha component of the output vector should be
  // f32OutAB.f32Arg1 = (12.99/36)*0.8660/(17.0/36/2) =
  //                    1.3235 -> FRAC32(1.0) ~ 0x7FFFFFFF
  // output beta component of the output vector should be
  // f32OutAB.f32Arg2 = (7.5/36)*0.8660/(17.0/36/2) =
  //                    0.7641 -> FRAC32(0.7641) ~ 0x61CF8000
  GMCLIB_ElimDcBusRip(&f32OutAB,&f32AB,&f32trMyElimDcBusRip);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.53.4 Function GMCLIB_ElimDcBusRip_F16

2.53.4.1 Declaration

```
void GMCLIB_ElimDcBusRip_F16(SWLIBS_2Syst_F16 *const pOut, const
SWLIBS_2Syst_F16 *const pIn, const GMCLIB_ELIMDCBUSRIP_T_F16
*const pParam);
```

2.53.4.2 Arguments

Table 306. GMCLIB_ElimDcBusRip_F16 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *const	pOut	**output**	Pointer to the structure with direct (α) and quadrature (β) components of the required stator voltage vector re-calculated so as to compensate for voltage ripples on the DC bus.
const SWLIBS_2Syst_F16 *const	pIn	**input**	Pointer to the structure with direct (α) and quadrature (β) components of the required stator voltage vector before compensation of voltage ripples on the DC bus.
const GMCLIB_ELIMDCBUSRIP_T_F16 *const	pParam	**input**	Pointer to the parameters structure.

2.53.4.3 Return

Function returns no value.

2.53.4.4 Implementation details

For better understanding, let's consider the following two simple examples:

Example 1:

- amplitude of the required phase voltage U_{reg}=50[V]
- maximal amplitude of the DC bus voltage $U_{DC_BUS_MAX}$=100[V]
- actual amplitude of the DC bus voltage $U_{DC_BUS_ACTUAL}$=100[V]
- voltage to be applied to the PWM modulator to generate U_{reg}=50[V] on the inverter phase output:

$$U_{req_new} = \frac{U_{req} U_{DC_BUS_MAX}}{U_{DC_BUS_ACTUAL}} = 50V$$

Equation GMCLIB_ElimDcBusRip_F16_Eq1

- amplitude of the required phase voltage U_{reg}=50[V]
- maximal amplitude of the DC bus voltage $U_{DC_BUS_MAX}$=100[V]
- actual amplitude of the DC bus voltage $U_{DC_BUS_ACTUAL}$=90[V]

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

- voltage to be applied to the PWM modulator to generate U_{reg}=50[V] on the inverter phase output:

$$U_{req_new} = \frac{U_{req} U_{DC_BUS_MAX}}{U_{DC_BUS_ACTUAL}} = 55.5V$$

Equation GMCLIB_ElimDcBusRip_F16_Eq2

$$u_{\alpha}^{*} = \begin{cases} \frac{f16ModIndex \cdot u_{\alpha}}{f16ArgDCBusMsr/2} & if \quad abs(f16ModIndex \cdot u_{\alpha}) < \frac{f16ArgDCBusMsr}{2} \\ sign(u_{\alpha}) & otherwise \end{cases}$$

$$u_{\beta}^{*} = \begin{cases} \frac{f16ModIndex \cdot u_{\beta}}{f16ArgDCBusMsr/2} & if \quad abs(f16ModIndex \cdot u_{\beta}) < \frac{f16ArgDCBusMsr}{2} \\ sign(u_{\beta}) & otherwise \end{cases}$$

Equation GMCLIB_ElimDcBusRip_F16_Eq3

where: f16ModIndex is the inverse modulation index, f16ArgDcBusMsr is the measured DC bus voltage, the u_{α} and u_{β} are the input voltages, and the u_{α}* and u_{β}* are the output duty-cycle ratios. The f16ModIndex and f16ArgDcBusMsr are supplied to the function within the parameters structure through its members. The u_{α}, u_{β} correspond respectively to the `f16Arg1` and `f16Arg2` members of the input structure, and the u_{α}* and u_{β}* respectively to the `f16Arg1` and `f16Arg2` members of the output structure. The inverse modulation index (see the parameters structure `pParam`, the `f16ModIndex` member) must be equal to or greater than zero. Besides this restriction, the f16ModIndex must be set to a valid value resulting from the use of Space Vector Modulation techniques.

Table 307. Inverse modulation index values for various SVM techniques

SVM technique	Inverse modulation index (f16ModIndex)
#GMCLIB_SvmU7n	sqrt(3)/2, use FRAC16(0.8660254037844386)
#GMCLIB_PwmIct	1, use FRAC16(1)
#GMCLIB_SvmStd	sqrt(3)/2, use FRAC16(0.8660254037844386)
#GMCLIB_SvmU0n	sqrt(3)/2, use FRAC16(0.8660254037844386)
#GMCLIB_SvmSci	sqrt(3)/2, use FRAC16(0.8660254037844386)

The function explicitly avoids the case of division by zero. If f16ArgDcBusMsr equals zero, the outputs will be saturated.

Note: *Both the inverse modulation index `pIn->f16ModIndex` and the measured DC bus voltage `pIn->f16DcBusMsr` must be equal to or greater than 0, otherwise the results are undefined.*

2.53.4.5 Code Example

```
#include "gmclib.h"

#define U_MAX   (36.0) // voltage scale
SWLIBS_2Syst_F16 f16AB;
SWLIBS_2Syst_F16 f16OutAB;
GMCLIB_ELIMDCBUSRIP_T_F16 f16trMyElimDcBusRip =
  GMCLIB_ELIMDCBUSRIP_DEFAULT_F16;

void main(void)
{
  // inverse modulation coefficient for standard space vector modulation
  f16trMyElimDcBusRip.f16ModIndex = FRAC16(0.866025403784439);
  // Input voltage vector 15V @ angle 30deg
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // alpha component of input voltage vector = 12.99[V]
  f16AB.f16Arg1 = FRAC16(12.99/U_MAX);
  // beta component of input voltage vector = 7.5[V]
  f16AB.f16Arg2 = FRAC16(7.5/U_MAX);
  // value of the measured DC bus voltage 17[V]
  f16trMyElimDcBusRip.f16ArgDcBusMsr = FRAC16(17.0/U_MAX);

  // output alpha component of the output vector should be
  // f16OutAB.f16Arg1 = (12.99/36)*0.8660/(17.0/36/2) =
  //                     1.3235 -> FRAC16(1.0) ~ 0x7FFF
  // output beta component of the output vector should be
  // f16OutAB.f16Arg2 = (7.5/36)*0.8660/(17.0/36/2) =
  //                    0.7641 -> FRAC16(0.7641) ~ 0x61CF
  GMCLIB_ElimDcBusRip_F16(&f16OutAB,&f16AB,&f16trMyElimDcBusRip);

  // output alpha component of the output vector should be
  // f16OutAB.f16Arg1 = (12.99/36)*0.8660/(17.0/36/2) =
  //                    1.3235 -> FRAC16(1.0) ~ 0x7FFF
  // output beta component of the output vector should be
  // f16OutAB.f16Arg2 = (7.5/36)*0.8660/(17.0/36/2) =
  //                    0.7641 -> FRAC16(0.7641) ~ 0x61CF
  GMCLIB_ElimDcBusRip(&f16OutAB,&f16AB,&f16trMyElimDcBusRip,F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output alpha component of the output vector should be
  // f16OutAB.f16Arg1 = (12.99/36)*0.8660/(17.0/36/2) =
  //                    1.3235 -> FRAC16(1.0) ~ 0x7FFF
  // output beta component of the output vector should be
  // f16OutAB.f16Arg2 = (7.5/36)*0.8660/(17.0/36/2) =
  //                    0.7641 -> FRAC16(0.7641) ~ 0x61CF
  GMCLIB_ElimDcBusRip(&f16OutAB,&f16AB,&f16trMyElimDcBusRip);
}
```

2.53.5 Function GMCLIB_ElimDcBusRip_FLT

2.53.5.1 Declaration

```
void GMCLIB_ElimDcBusRip_FLT(SWLIBS_2Syst_FLT *const pOut, const
SWLIBS_2Syst_FLT *const pIn, const GMCLIB_ELIMDCBUSRIP_T_FLT
*const pParam);
```

2.53.5.2 Arguments

Table 308. GMCLIB_ElimDcBusRip_FLT arguments

Type	Name	Direction	Description
SWLIBS_2Syst_FLT *const	pOut	**output**	Pointer to the structure with direct (α) and quadrature (β) components of the required stator voltage vector re-calculated so as to compensate for voltage ripples on the DC bus. Outputs are normalized to the range [-1; 1].
const SWLIBS_2Syst_FLT *const	pIn	**input**	Pointer to the structure with direct (α) and quadrature (β) components of the required stator voltage vector before compensation of voltage ripples on the DC bus.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 308. GMCLIB_ElimDcBusRip_FLT arguments...*continued*

Type	Name	Direction	Description
const GMCLIB_ELIMDCBUSRIP_T_FLT *const	pParam	**input**	Pointer to the parameters structure.

2.53.5.3 Return

Function returns no value.

2.53.5.4 Implementation details

Outputs are normalized to the range [-1; 1] and can be used as inputs to the function GMCLIB_SvmStd_FLT. If the required stator voltage exceeds the capabilities of the DC bus, the output is saturated to +/-1.

For better understanding, let's consider the following two simple examples:

Example 1:

- amplitude of the required phase voltage U_{reg}=50[V]
- maximal amplitude of the DC bus voltage $U_{DC_BUS_MAX}$=100[V]
- actual amplitude of the DC bus voltage $U_{DC_BUS_ACTUAL}$=100[V]
- voltage to be applied to the PWM modulator to generate U_{reg}=50[V] on the inverter phase output:

$$U_{req_new} = \frac{U_{req} U_{DC_BUS_MAX}}{U_{DC_BUS_ACTUAL}} = 50V$$

Equation GMCLIB_ElimDcBusRip_FLT_Eq1

Example 2:

- amplitude of the required phase voltage U_{reg}=50[V]
- maximal amplitude of the DC bus voltage $U_{DC_BUS_MAX}$=100[V]
- actual amplitude of the DC bus voltage $U_{DC_BUS_ACTUAL}$=90[V]
- voltage to be applied to the PWM modulator to generate U_{reg}=50[V] on the inverter phase output:

$$U_{req_new} = \frac{U_{req} U_{DC_BUS_MAX}}{U_{DC_BUS_ACTUAL}} = 55.5V$$

Equation GMCLIB_ElimDcBusRip_FLT_Eq2

The imperfections of the DC bus voltage are compensated for by the modification of amplitudes of the direct- α and the quadrature- β components of the stator reference voltage vector. The following formulas are used:

$$u_{\alpha}^{*} = \begin{cases} \frac{fltModIndex \cdot u_{\alpha}}{fltArgDCBusMsr/2} & if \quad abs(fltModIndex \cdot u_{\alpha}) < \frac{fltArgDCBusMsr}{2} \\ sign(u_{\alpha}) & otherwise \end{cases}$$

$$u_{\beta}^{*} = \begin{cases} \frac{fltModIndex \cdot u_{\beta}}{fltArgDCBusMsr/2} & if \quad abs(fltModIndex \cdot u_{\beta}) < \frac{fltArgDCBusMsr}{2} \\ sign(u_{\beta}) & otherwise \end{cases}$$

Equation GMCLIB_ElimDcBusRip_FLT_Eq3

where fltModIndex is the inverse modulation index, fltArgDcBusMsr is the measured DC bus voltage, the u_{α} and u_{β} are the input voltages, and the u_{α}* and u_{β}* are the output duty-cycle ratios.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The fltModIndex and fltArgDcBusMsr are supplied to the function within the parameters structure through its members. The u_α, u_β correspond respectively to the `fltArg1` and `fltArg2` members of the input structure `pIn`, and the u_α* and u_β* respectively to the `fltArg1` and `fltArg2` members of the output structure `pOut`.

The inverse modulation index (see the parameters structure `pParam`, the `fltModIndex` member) must be equal to or greater than zero. Besides this restriction, the fltModIndex must be set to a valid value resulting from the use of Space Vector Modulation techniques.

Table 309. Inverse modulation index values for various SVM techniques

SVM technique	Inverse modulation index (fltModIndex)
#GMCLIB_SvmU7n	sqrt(3)/2 = 0.8660254037844386
#GMCLIB_PwmIct	1
#GMCLIB_SvmStd	sqrt(3)/2 = 0.8660254037844386
#GMCLIB_SvmU0n	sqrt(3)/2 = 0.8660254037844386
#GMCLIB_SvmSci	sqrt(3)/2 = 0.8660254037844386

Note: *The function may raise floating-point exceptions (invalid operation, division by zero, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

Both the inverse modulation index `pIn->fltModIndex` and the measured DC bus voltage `pIn->fltDcBusMsr` must be equal to or greater than 0, otherwise the results are undefined.

2.53.5.5 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_FLT fltAB;
SWLIBS_2Syst_FLT fltOutAB;
GMCLIB_ELIMDCBUSRIP_T_FLT flttrMyElimDcBusRip =
  GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT;

void main(void)
{
  // inverse modulation coefficient for standard space vector modulation
  trMyElimDcBusRip.fltModIndex = 0.866025404;
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  fltAB.fltArg1 = 12.99038106;
  // beta component of input voltage vector = 7.5[V]
  fltAB.fltArg2 = 7.5;
  // value of the measured DC bus voltage 17V
  flttrMyElimDcBusRip.fltArgDcBusMsr = 17;

  // output alpha component of the output vector should be
  // fltOutAB.fltArg1 = 1
  // output beta component of the output vector should be
  // fltOutAB.fltArg2 = 0.764140062
  GMCLIB_ElimDcBusRip_FLT(&fltOutAB,&fltAB,&flttrMyElimDcBusRip);

  // output alpha component of the output vector should be
  // fltOutAB.fltArg1 = 1
  // output beta component of the output vector should be
  // fltOutAB.fltArg2 = 0.764140062
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  GMCLIB_ElimDcBusRip(&fltOutAB,&fltAB,&flttrMyElimDcBusRip,FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output alpha component of the output vector should be
  // fltOutAB.fltArg1 = 1
  // output beta component of the output vector should be
  // fltOutAB.fltArg2 = 0.764140062
  GMCLIB_ElimDcBusRip(&fltOutAB,&fltAB,&flttrMyElimDcBusRip);
}
```

2.54 Function GMCLIB_Park

This function implements the calculation of Park transformation.

2.54.1 Description

The GMCLIB_Park function calculates the Park Transformation, which transforms values (flux, voltage, current) from the two-phase (α-β) stationary orthogonal coordinate system to the two-phase (d-q) rotational orthogonal coordinate system, according to these equations:

$$d = \cos(\theta_e) \cdot \alpha + \sin(\theta_e) \cdot \beta$$
$$q = -\sin(\theta_e) \cdot \alpha + \cos(\theta_e) \cdot \beta$$

Equation GMCLIB_Park_Eq1

where θ_e represents the electrical position of the rotor flux.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.54.2 Re-entrancy

The function is re-entrant for a different set of input/output pointers.

2.54.3 Function GMCLIB_Park_F32

2.54.3.1 Declaration

```
void GMCLIB_Park_F32(SWLIBS_2Syst_F32 *pOut, const
SWLIBS_2Syst_F32 *const pInAngle, const SWLIBS_2Syst_F32 *const
pIn);
```

2.54.3.2 Arguments

Table 310. GMCLIB_Park_F32 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F32 *	pOut	**input, output**	Pointer to the structure containing data of the two-phase rotational orthogonal system (d-q).
const SWLIBS_2Syst_F32 *const	pInAngle	**input**	Pointer to the structure where the values of the sine and cosine of the rotor position are stored.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 310. GMCLIB_Park_F32 arguments...*continued*

Type	Name	Direction	Description
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α-β).

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

The inputs and the outputs are normalized to fit in the range [-1, 1).

2.54.3.3 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_F32 tr32Angle;
SWLIBS_2Syst_F32 tr32AlBe;
SWLIBS_2Syst_F32 tr32Dq;

void main(void)
{
  // input angle sin(60) = 0.866025403
  // input angle cos(60) = 0.5
  tr32Angle.f32Arg1  = FRAC32(0.866025403);
  tr32Angle.f32Arg2  = FRAC32(0.5);

  // input alpha = 0.123
  // input beta = 0.654
  tr32AlBe.f32Arg1  = FRAC32(0.123);
  tr32AlBe.f32Arg2  = FRAC32(0.654);

  // output should be
  // tr32Dq.f32Arg1 ~ d = 0x505E6455
  // tr32Dq.f32Arg2 ~ q = 0x1C38ABDC
  GMCLIB_Park_F32(&tr32Dq,&tr32Angle,&tr32AlBe);

  // output should be
  // tr32Dq.f32Arg1 ~ d = 0x505E6455
  // tr32Dq.f32Arg2 ~ q = 0x1C38ABDC
  GMCLIB_Park(&tr32Dq,&tr32Angle,&tr32AlBe,F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be
  // tr32Dq.f32Arg1 ~ d = 0x505E6455
  // tr32Dq.f32Arg2 ~ q = 0x1C38ABDC
  GMCLIB_Park(&tr32Dq,&tr32Angle,&tr32AlBe);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.54.4 Function GMCLIB_Park_F16

2.54.4.1 Declaration

```
void GMCLIB_Park_F16(SWLIBS_2Syst_F16 *pOut, const
SWLIBS_2Syst_F16 *const pInAngle, const SWLIBS_2Syst_F16 *const
pIn);
```

2.54.4.2 Arguments

Table 311. GMCLIB_Park_F16 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *	pOut	**input, output**	Pointer to the structure containing data of the two-phase rotational orthogonal system (d-q).
const SWLIBS_2Syst_F16 *const	pInAngle	**input**	Pointer to the structure where the values of the sine and cosine of the rotor position are stored.
const SWLIBS_2Syst_F16 *const	pIn	**input**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α-β).

Note: *This function is implemented using inline assembly and is therefore not ANSI-C compliant.*

The inputs and the outputs are normalized to fit in the range [-1, 1).

2.54.4.3 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_F16 tr16Angle;
SWLIBS_2Syst_F16 tr16AlBe;
SWLIBS_2Syst_F16 tr16Dq;

void main(void)
{
  // input angle sin(60) = 0.866025403
  // input angle cos(60) = 0.5
  tr16Angle.f16Arg1  = FRAC16(0.866025403);
  tr16Angle.f16Arg2  = FRAC16(0.5);

  // input alpha = 0.123
  // input beta = 0.654
  tr16AlBe.f16Arg1  = FRAC16(0.123);
  tr16AlBe.f16Arg2  = FRAC16(0.654);

  // output should be
  // tr16Dq.f16Arg1 ~ d = 0x505E
  // tr16Dq.f16Arg2 ~ q = 0x1C38
  GMCLIB_Park_F16(&tr16Dq,&tr16Angle,&tr16AlBe);

  // output should be
  // tr16Dq.f16Arg1 ~ d = 0x505E
  // tr16Dq.f16Arg2 ~ q = 0x1C38
  GMCLIB_Park(&tr16Dq,&tr16Angle,&tr16AlBe,F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // ##########################################################

  // output should be
  // tr16Dq.f16Arg1 ~ d = 0x505E
  // tr16Dq.f16Arg2 ~ q = 0x1C38
  GMCLIB_Park(&tr16Dq,&tr16Angle,&tr16AlBe);
}
```

2.54.5 Function GMCLIB_Park_FLT

2.54.5.1 Declaration

```
void GMCLIB_Park_FLT(SWLIBS_2Syst_FLT *pOut, const
SWLIBS_2Syst_FLT *const pInAngle, const SWLIBS_2Syst_FLT *const
pIn);
```

2.54.5.2 Arguments

Table 312. GMCLIB_Park_FLT arguments

Type	Name	Direction	Description
SWLIBS_2Syst_FLT *	pOut	**input, output**	Pointer to the structure containing data of the two-phase rotational orthogonal system (d-q).
const SWLIBS_2Syst_FLT *const	pInAngle	**input**	Pointer to the structure where the values of the sine and cosine of the rotor position are stored.
const SWLIBS_2Syst_FLT *const	pIn	**input**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α-β).

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.54.5.3 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_FLT trfltAngle;
SWLIBS_2Syst_FLT trfltAlBe;
SWLIBS_2Syst_FLT trfltDq;

void main(void)
{
  // input angle sin(60) = 0.866025403
  // input angle cos(60) = 0.5
  trfltAngle.fltArg1  = 0.866025403;
  trfltAngle.fltArg2  = 0.5;

  // input alpha = 0.123
  // input beta = 0.654
  trfltAlBe.fltArg1  = 0.123;
  trfltAlBe.fltArg2  = 0.654;

  // output should be:
  // trfltDq.fltArg1 ~ d = 0.627880613
  // trfltDq.fltArg2 ~ q = 0.220479472
  GMCLIB_Park_FLT(&trfltDq,&trfltAngle,&trfltAlBe);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // output should be:
  // trfltDq.fltArg1 ~ d = 0.627880613
  // trfltDq.fltArg2 ~ q = 0.220479472
  GMCLIB_Park(&trfltDq,&trfltAngle,&trfltAlBe,FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output should be:
  // trfltDq.fltArg1 ~ d = 0.627880613
  // trfltDq.fltArg2 ~ q = 0.220479472
  GMCLIB_Park(&trfltDq,&trfltAngle,&trfltAlBe);
}
```

2.55 Function GMCLIB_ParkInv

This function implements the inverse Park transformation.

2.55.1 Description

The GMCLIB_ParkInv function calculates the Inverse Park Transformation, which transforms quantities (flux, voltage, current) from the two-phase (d-q) rotational orthogonal coordinate system to the two-phase (α-β) stationary orthogonal coordinate system, according to these equations:

$$\alpha = \cos(\theta_e) \cdot d - \sin(\theta_e) \cdot q$$
$$\beta = \sin(\theta_e) \cdot d + \cos(\theta_e) \cdot q$$

Equation GMCLIB_ParkInv_Eq1

where θ_e represents the electrical position of the rotor flux.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.55.2 Re-entrancy

The function is re-entrant for a different set of input/output pointers.

2.55.3 Function GMCLIB_ParkInv_F32

2.55.3.1 Declaration

```
void GMCLIB_ParkInv_F32(SWLIBS_2Syst_F32 *const pOut, const
SWLIBS_2Syst_F32 *const pInAngle, const SWLIBS_2Syst_F32 *const
pIn);
```

2.55.3.2 Arguments

Table 313. GMCLIB_ParkInv_F32 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F32 *const	pOut	**input, output**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α-β).

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 313. GMCLIB_ParkInv_F32 arguments...*continued*

Type	Name	Direction	Description
const SWLIBS_2Syst_F32 *const	pInAngle	**input**	Pointer to the structure where the values of the sine and cosine of the rotor position are stored.
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure containing data of the two-phase rotational orthogonal system (d-q).

Note: *The inputs and the outputs are normalized to fit in the range [-1, 1).*

2.55.3.3 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_F32 tr32Angle;
SWLIBS_2Syst_F32 tr32Dq;
SWLIBS_2Syst_F32 tr32AlBe;

void main(void)
{
  // input angle sin(60) = 0.866025403
  // input angle cos(60) = 0.5
  tr32Angle.f32Arg1  = FRAC32(0.866025403);
  tr32Angle.f32Arg2  = FRAC32(0.5);

  // input d  = 0.123
  // input q  = 0.654
  tr32Dq.f32Arg1  = FRAC32(0.123);
  tr32Dq.f32Arg2  = FRAC32(0.654);

  // output should be
  // tr32AlBe.f32Arg1 ~ alpha = 0xBF601273
  // tr32AlBe.f32Arg2 ~ beta  = 0x377D9EE4
  GMCLIB_ParkInv_F32(&tr32AlBe,&tr32Angle,&tr32Dq);

  // output should be
  // tr32AlBe.f32Arg1 ~ alpha = 0xBF601273
  // tr32AlBe.f32Arg2 ~ beta  = 0x377D9EE4
  GMCLIB_ParkInv(&tr32AlBe,&tr32Angle,&tr32Dq,F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be
  // tr32AlBe.f32Arg1 ~ alpha = 0xBF601273
  // tr32AlBe.f32Arg2 ~ beta  = 0x377D9EE4
  GMCLIB_ParkInv(&tr32AlBe,&tr32Angle,&tr32Dq);
}
```

2.55.4 Function GMCLIB_ParkInv_F16

2.55.4.1 Declaration

```
void GMCLIB_ParkInv_F16(SWLIBS_2Syst_F16 *const pOut, const
SWLIBS_2Syst_F16 *const pInAngle, const SWLIBS_2Syst_F16 *const
pIn);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.55.4.2 Arguments

Table 314. GMCLIB_ParkInv_F16 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *const	pOut	**input, output**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α-β).
const SWLIBS_2Syst_F16 *const	pInAngle	**input**	Pointer to the structure where the values of the sine and cosine of the rotor position are stored.
const SWLIBS_2Syst_F16 *const	pIn	**input**	Pointer to the structure containing data of the two-phase rotational orthogonal system (d-q).

Note: *The inputs and the outputs are normalized to fit in the range [-1, 1).*

2.55.4.3 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_F16 tr16Angle;
SWLIBS_2Syst_F16 tr16Dq;
SWLIBS_2Syst_F16 tr16AlBe;

void main(void)
{
  // input angle sin(60) = 0.866025403
  // input angle cos(60) = 0.5
  tr16Angle.f16Arg1  = FRAC16(0.866025403);
  tr16Angle.f16Arg2  = FRAC16(0.5);

  // input d  = 0.123
  // input q  = 0.654
  tr16Dq.f16Arg1  = FRAC16(0.123);
  tr16Dq.f16Arg2  = FRAC16(0.654);

  // output should be
  // tr16AlBe.f16Arg1 ~ alpha = 0xBF61
  // tr16AlBe.f16Arg2 ~ beta  = 0x377C
  GMCLIB_ParkInv_F16(&tr16AlBe,&tr16Angle,&tr16Dq);

  // output should be
  // tr16AlBe.f16Arg1 ~ alpha = 0xBF61
  // tr16AlBe.f16Arg2 ~ beta  = 0x377C
  GMCLIB_ParkInv(&tr16AlBe,&tr16Angle,&tr16Dq,F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be
  // tr16AlBe.f16Arg1 ~ alpha = 0xBF61
  // tr16AlBe.f16Arg2 ~ beta  = 0x377C
  GMCLIB_ParkInv(&tr16AlBe,&tr16Angle,&tr16Dq);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.55.5 Function GMCLIB_ParkInv_FLT

2.55.5.1 Declaration

```
void GMCLIB_ParkInv_FLT(SWLIBS_2Syst_FLT *const pOut, const
SWLIBS_2Syst_FLT *const pInAngle, const SWLIBS_2Syst_FLT *const
pIn);
```

2.55.5.2 Arguments

Table 315. GMCLIB_ParkInv_FLT arguments

Type	Name	Direction	Description
SWLIBS_2Syst_FLT *const	pOut	**input, output**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α-β).
const SWLIBS_2Syst_FLT *const	pInAngle	**input**	Pointer to the structure where the values of the sine and cosine of the rotor position are stored.
const SWLIBS_2Syst_FLT *const	pIn	**input**	Pointer to the structure containing data of the two-phase rotational orthogonal system (d-q).

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

The inputs and the outputs are in single precision floating point data format.

2.55.5.3 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_FLT trfltAngle;
SWLIBS_2Syst_FLT trfltDq;
SWLIBS_2Syst_FLT trfltAlBe;

void main(void)
{
  // input angle sin(60) = 0.866025403
  // input angle cos(60) = 0.5
  trfltAngle.fltArg1  = 0.866025403;
  trfltAngle.fltArg2  = 0.5;

  // input d  = 0.123
  // input q  = 0.654
  trfltDq.fltArg1  = 0.123;
  trfltDq.fltArg2  = 0.654;

  // output should be:
  // trfltAlBe.fltArg1 ~ alpha = -0.495119387
  // trfltAlBe.fltArg2 ~ beta  = 0.433521139
  GMCLIB_ParkInv_FLT(&trfltAlBe,&trfltAngle,&trfltDq);

  // output should be:
  // trfltAlBe.fltArg1 ~ alpha = -0.495119387
  // trfltAlBe.fltArg2 ~ beta  = 0.433521139
  GMCLIB_ParkInv(&trfltAlBe,&trfltAngle,&trfltDq,FLT);

  // ##############################################################
  // Available only if single precision floating point
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // implementation selected as default
  // ##########################################################

  // output should be:
  // trfltAlBe.fltArg1 ~ alpha = -0.495119387
  // trfltAlBe.fltArg2 ~ beta  = 0.433521139
  GMCLIB_ParkInv(&trfltAlBe,&trfltAngle,&trfltDq);
}
```

2.56 Function GMCLIB_PwmIct

This function calculates appropriate duty-cycle ratios which are needed for generating the given stator reference voltage vector using the general sinusoidal modulation technique.

2.56.1 Description

The GMCLIB_PwmIct function calculates the appropriate duty-cycle ratios needed to generate the given stator reference voltage vector with the help of the conventional inverse Clarke transform. Refer to the description of GMLIB_SvmStd for explanation of the stator voltage sectors.

The following equations are used to calculate the output duty-cycle ratios in the range 0 < pwm < 1

$$pwm_A = 0.5 + \frac{u_\alpha}{2}$$

$$pwm_B = 0.5 + \frac{-u_\alpha + \sqrt{3}\,u_\beta}{4}$$

$$pwm_C = 0.5 + \frac{-u_\alpha - \sqrt{3}\,u_\beta}{4}$$

Equation GMCLIB_PwmIct_Eq1

The following figures show the input and output waveforms.

Figure 103. Input waveforms

© NXP B.V. 2023. All rights reserved.

Figure 104. Output waveforms

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.56.2 Re-entrancy

The function is re-entrant for a different set of input/output pointers.

2.56.3 Function GMCLIB_PwmIct_F32

2.56.3.1 Declaration

```
tU32 GMCLIB_PwmIct_F32(SWLIBS_3Syst_F32 *pOut, const
SWLIBS_2Syst_F32 *const pIn);
```

2.56.3.2 Arguments

Table 316. GMCLIB_PwmIct_F32 arguments

Type	Name	Direction	Description
SWLIBS_3Syst_F32 *	pOut	**input, output**	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector.

2.56.3.3 Return

The function returns a 32-bit integer value representing the sector which contains the stator reference vector U_s.

2.56.3.4 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_F32 inVoltage;
SWLIBS_3Syst_F32 pwmABC;
tU32 sector;

void main(void)
{
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  // beta component of input voltage vector  = 7.5[V]
  inVoltage.f32Arg1 = FRAC32(12.99/U_MAX);
  inVoltage.f32Arg2 = FRAC32(7.5/U_MAX);

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  sector = GMCLIB_PwmIct_F32(&pwmABC,&inVoltage);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  sector = GMCLIB_PwmIct(&pwmABC,&inVoltage,F32);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  sector = GMCLIB_PwmIct(&pwmABC,&inVoltage);

  // Expected output pwm duty-cycles pwmABC:
  // pwmABC.f32Arg1 = 0x776C8B43 ~ FRAC32(0.9330)
  // pwmABC.f32Arg2 = 0x40003546 ~ FRAC32(0.5000)
  // pwmABC.f32Arg3 = 0x08933F77 ~ FRAC32(0.0670)
  // sector = 1
}
```

2.56.4 Function GMCLIB_PwmIct_F16

2.56.4.1 Declaration

```
tU16 GMCLIB_PwmIct_F16(SWLIBS_3Syst_F16 *pOut, const
SWLIBS_2Syst_F16 *const pIn);
```

2.56.4.2 Arguments

Table 317. GMCLIB_PwmIct_F16 arguments

Type	Name	Direction	Description
SWLIBS_3Syst_F16 *	pOut	**input, output**	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.
const SWLIBS_2Syst_F16 *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector.

2.56.4.3 Return

The function returns a 16-bit integer value representing the sector which contains the stator reference vector U_s.

2.56.4.4 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_F16 inVoltage;
SWLIBS_3Syst_F16 pwmABC;
tU16 sector;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
void main(void)
{
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  // beta component of input voltage vector  = 7.5[V]
  inVoltage.f16Arg1 = FRAC16(12.99/U_MAX);
  inVoltage.f16Arg2 = FRAC16(7.5/U_MAX);

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  sector = GMCLIB_PwmIct_F16(&pwmABC,&inVoltage);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  sector = GMCLIB_PwmIct(&pwmABC,&inVoltage,F16);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  sector = GMCLIB_PwmIct(&pwmABC,&inVoltage);

  // Expected output pwm duty-cycles pwmABC:
  // pwmABC.f16Arg1 = 0x776C ~ FRAC16(0.9330)
  // pwmABC.f16Arg2 = 0x4000 ~ FRAC16(0.5000)
  // pwmABC.f16Arg3 = 0x0894 ~ FRAC16(0.0670)
  // sector = 1
}
```

2.56.5 Function GMCLIB_PwmIct_FLT

2.56.5.1 Declaration

```
tU32 GMCLIB_PwmIct_FLT(SWLIBS_3Syst_FLT *pOut, const
SWLIBS_2Syst_FLT *const pIn);
```

2.56.5.2 Arguments

Table 318. GMCLIB_PwmIct_FLT arguments

Type	Name	Direction	Description
SWLIBS_3Syst_FLT *	pOut	**input, output**	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.
const SWLIBS_2Syst_FLT *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector.

2.56.5.3 Return

The function returns a 32-bit integer value representing the sector which contains the stator reference vector U_s.

2.56.5.4 Implementation details

The function presumes that the input voltages are normalized to fit the range <-1; 1>. In a typical motor control application, this function is preceded by the function GMCLIB_ElimDcBusRip_FLT which ensures that the voltages are correctly normalized.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.56.5.5 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_FLT inVoltage;
SWLIBS_3Syst_FLT pwmABC;
tU32 sector;

void main(void)
{
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  // beta component of input voltage vector  = 7.5[V]
  inVoltage.fltArg1 = (tFloat)(12.99/U_MAX);
  inVoltage.fltArg2 = (tFloat)(7.5/U_MAX);

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  sector = GMCLIB_PwmIct_FLT(&pwmABC,&inVoltage);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  sector = GMCLIB_PwmIct(&pwmABC,&inVoltage,FLT);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if single precision floating point implementation is selected
  // as default.
  sector = GMCLIB_PwmIct(&pwmABC,&inVoltage);

  // Expected output pwm duty-cycles pwmABC:
  // pwmABC.fltArg1 = 0.93299997
  // pwmABC.fltArg2 = 0.50000632
  // pwmABC.fltArg3 = 0.066993654
  // sector = 1
}
```

2.57 Function GMCLIB_SpreadPWM

This function outputs a spreading sequence for the pulse-width modulator (PWM) period counter.

2.57.1 Declaration

```
tU16 GMCLIB_SpreadPWM(GMCLIB_SPREADPWM_T *const pParam);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.57.2 Arguments

Table 319. GMCLIB_SpreadPWM arguments

Type	Name	Direction	Description
GMCLIB_SPREADPWM_T *const	pParam	**input, output**	Pointer to the state and parameters structure.

2.57.3 Return

The function returns 16-bit unsigned integer corresponding to a new modulus of the PWM period counter.

2.57.4 Description

This function outputs a spreading sequence for the pulse-width modulator (PWM) period counter. Spreading the PWM spectrum reduces electromagnetic interference (EMI). The following figure demonstrates the effect of the PWM spectrum spreading on the power spectral density of the 20 kHz PWM carrier and higher harmonics. The distinct peaks in the spectrum are significantly reduced. Note that the actual emitted spectrum depends on various physical factors (power stage and motor parameters, shielding etc.).

Figure 105. Power spectral density of a PWM signal with higher harmonics

It is necessary to set the parameters and initialize state variables by calling GMCLIB_SpreadPWMInit before the first call of GMCLIB_SpreadPWM. The effect of the spectrum spreading increases with higher frequency deviation *u8FreqDev*. The spreading sequence contains a semi-periodic component, the frequency of which is determined by the modulation frequency *u32SpreadFreq*. Lowering this value produces flatter spectrum, however, the modulation frequency must be kept above the passband of the FOC control loop to avoid undesirable disturbances of the control algorithm. Setting *u32SpreadFreq* higher than one half of *u32PWMFreq* removes the semi-periodic component and turns the spreading sequence into white noise. This setting also speeds up the execution by approx. 43% and sets constant execution time.

Higher *u32SpreadFreq* may be also needed to satisfy the requirements of an EMC standard such as CISPR. The average speed of change of the PWM frequency is given by the following equation:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$\Delta f / \Delta t = \frac{2 \cdot u8FreqDev}{100} \cdot u32PWMFreq \cdot u32SpreadFreq \text{ [Hz/s]}$$

Equation GMCLIB_SpreadPWM_Eq1

To achieve effective spreading, the PWM frequency (and higher harmonics) must sweep through the bandpass of the receiver filter bandwidth (RBW) faster than

$$(\Delta f / \Delta t)_{min} = RBW^2 \text{ [Hz/s]}$$

Equation GMCLIB_SpreadPWM_Eq2

For example, *u32SpreadFreq* higher than 1013 Hz with 25% PWM frequency deviation will ensure that the 8th harmonic of the 20 kHz PWM carrier will be sweeping faster than 81 MHz/s, which is the minimum dictated by the bandpass impulse response time of the 9 kHz receiver filter in the 150 kHz - 30 MHz band defined by CISPR 16-1-1.

The unsigned 16-bit integer returned by the function should be written into the PWM modulus register in each PWM period to make sure that the spreading sequence achieves the desired spectral properties. If the PWM duty cycle depends on the PWM period, the threshold values for each phase need to be updated according to the new PWM period (see code example below).

Note that changing the period of the PWM cycle also affects the available amount of computational time. The upper bound of the PWM frequency (and thus the shortest period) needs to be selected such that all necessary processing is finished before the next PWM reload interrupt.

The function also produces a 32-bit pseudorandom white noise output in the element *u32Rnd* of the structure pointed to by *pParam*. All bits of *u32Rnd* are randomized, the sequence repeats with a period of 2^{32}. This pseudorandom sequence is not suitable for security or Monte Carlo simulation purposes. The sequence is seeded with a constant value in GMCLIB_SpreadPWMInit, the seed can be changed at any time by writing a non-zero value to *u32Rnd*.

Note: *The input pointer must contain a valid address otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.57.5 Code Example

```
#include "gmclib.h"
GMCLIB_SPREADPWM_T pParam;
void main(void)
{
  // Initialize the internal state
  GMCLIB_SpreadPWMInit(20000, 4000, 1000, 25, &pParam);
}

// Important: The PWM modulus must be updated
// every PWM period otherwise the modulation
// frequency will intrude the control loop
// passband.
void SetDutyCycle(SWLIBS_3Syst_F16 *f16pwm)
{
  tU16 u16Modulus;

  // get new modulus
  u16Modulus = GMCLIB_SpreadPWM(&pParam);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // set new PWM modulus for all phases (tU16 value)
  <PWM_A_MODULUS_REG> = u16Modulus;
  <PWM_B_MODULUS_REG> = u16Modulus;
  <PWM_C_MODULUS_REG> = u16Modulus;

  // update PWM duty cycle thresholds for all phases (tU16 value)
  <PWM_A_THRESHOLD_REG> = (tU16)MLIB_Mul_F16(f16pwm->f16Arg1, (tFrac16)u16Modulus);
  <PWM_B_THRESHOLD_REG> = (tU16)MLIB_Mul_F16(f16pwm->f16Arg2, (tFrac16)u16Modulus);
  <PWM_C_THRESHOLD_REG> = (tU16)MLIB_Mul_F16(f16pwm->f16Arg3, (tFrac16)u16Modulus);
}
```

2.57.6 Re-entrancy

The function is re-entrant for a different `pParam`.

2.57.7 Function GMCLIB_SpreadPWMInit

This function initializes the spreading sequence generator.

2.57.7.1 Declaration

```
void GMCLIB_SpreadPWMInit(tU32 u32PWMFreq, tU16 u16Mod, tU32
u32SpreadFreq, tU8 u8FreqDev, GMCLIB_SPREADPWM_T *const pParam);
```

2.57.7.2 Arguments

Table 320. GMCLIB_SpreadPWMInit arguments

Type	Name	Direction	Description
tU32	u32PWMFreq	**input**	PWM center frequency [Hz].
tU16	u16Mod	**input**	Nominal period of the PWM counter (modulus) corresponding to the PWM center frequency.
tU32	u32SpreadFreq	**input**	Modulation frequency of the spreading signal [Hz]. This frequency should be set above the passband of the FOC current controller to avoid disturbing the control loop with the spreading signal.
tU8	u8FreqDev	**input**	Frequency deviation of the spread spectrum in percents of PWM center frequency [%]. The range of the frequency deviation is limited to max. ±25%.
GMCLIB_SPREADPWM_T *const	pParam	**input, output**	Pointer to the state and parameters structure.

2.57.7.3 Description

This function initializes the spreading sequence generator and sets the properties of the generated sequence. It is necessary to call GMCLIB_SpreadPWMInit before the first use of GMCLIB_SpreadPWM.

2.57.7.4 Code Example

```
#include "gmclib.h"
GMCLIB_SPREADPWM_T pParam;
void main(void)
{
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  tU16 u16Modulus = 2000;

  // Initialize the internal state
  GMCLIB_SpreadPWMInit(20000, 2000, 1000, 25, &pParam);

  // u16Modulus = 2000
  u16Modulus = GMCLIB_SpreadPWM(&pParam);
  // u16Modulus = random value near 2000
  u16Modulus = GMCLIB_SpreadPWM(&pParam);
  // u16Modulus = another random value near 2000
}
```

2.57.7.5 Re-entrancy

The function is re-entrant for a different *pParam*.

2.58 Function GMCLIB_SvmSci

This function calculates appropriate duty-cycle ratios, which are needed for generating the given stator reference voltage vector using the General Sinusoidal Modulation with injection of the third harmonic.

2.58.1 Description

The GMCLIB_SvmSci function calculates the appropriate duty-cycle ratios, needed for generating the given stator reference voltage vector with the help of the sinusoidal modulation with Sine Cap Injection algorithm.

Finding the sector in which the reference stator voltage vector U_S resides is similar to that discused in GMCLIB_SvmStd.

The balanced 3-Phase duty-cycle ratios may be calculated based on Sine Cap Injection algorithm in the following stages:

1. The calculation of the basic duty-cycle ratios using the Inverse Clarke Transformation.

$$u_a = u_\alpha$$

$$u_b = \frac{-u_\alpha + \sqrt{3} \cdot u_\beta}{2}$$

$$u_c = \frac{-u_\alpha - \sqrt{3} \cdot u_\beta}{2}$$

Equation GMCLIB_SvmSci_Eq1

2. An amplitude of the basic duty-cycle ratios u_a, u_b and u_c calculated by GMCLIB_SvmSci_Eq1 is in the range [-1, 1]. The basic duty-cycle ratios are then multiplied by the coeficient 2/√3.

$$u'_a = \frac{2}{\sqrt{3}} u_a$$

$$u'_b = \frac{2}{\sqrt{3}} u_b$$

$$u'_c = \frac{2}{\sqrt{3}} u_c$$

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Equation GMCLIB_SvmSci_Eq2

3. If the values of variables u'_a, u'_b and u'_c exceed the unity, they are stored in an auxiliary variable u_0. This variable is called the Sine Cap Voltage variable. The procedure to obtain this can be mathematically defined by a following series of formulas:

$$u_0 = \begin{cases} 1 - u'_a & if\ u'_a > 1 \\ -1 - u'_a & if\ u'_a < -1 \\ 0 & otherwise \end{cases}$$
$$u_0 = \begin{cases} 1 - u'_b & if\ u'_b > 1 \\ -1 - u'_b & if\ u'_b < -1 \\ 0 & otherwise \end{cases}$$
$$u_0 = \begin{cases} 1 - u'_c & if\ u'_c > 1 \\ -1 - u'_c & if\ u'_c < -1 \\ 0 & otherwise \end{cases}$$

Equation GMCLIB_SvmSci_Eq3

4. Due to the 120° voltage phase shift, distinguishing for balanced three-phase system, only one phase contributes to the building of Sine-Cap Voltage u_0 at each time point.

Finally the duty-cycle ratios are then calculated bu following equations:

$$pwm_A = \frac{1}{2}(u_0 + u'_a + 1)$$
$$pwm_B = \frac{1}{2}(u_0 + u'_b + 1)$$
$$pwm_C = \frac{1}{2}(u_0 + u'_c + 1)$$

Equation GMCLIB_SvmSci_Eq4

Figure 106 shows calculated waveforms of the duty-cycle ratios using the sinusoidal modulation with Sine Cap Injection algorithm.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 106. Sinusoidal Modulation with Injection of the Third Harmonic

Note: *To provide an accurate calculation of the duty-cycle ratios, the stator reference voltage vector given by direct-a and quadrature-b components can not have magnitude exceeding the unity circle.*

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.58.2 Re-entrancy

The function is re-entrant for a different set of input/output pointers.

2.58.3 Function GMCLIB_SvmSci_F32

2.58.3.1 Declaration

```
tU32 GMCLIB_SvmSci_F32(SWLIBS_3Syst_F32 *pOut, const
SWLIBS_2Syst_F32 *const pIn);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.58.3.2 Arguments

Table 321. GMCLIB_SvmSci_F32 arguments

Type	Name	Direction	Description
SWLIBS_3Syst_F32 *	pOut	**input, output**	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector.

2.58.3.3 Return

The function returns a 32-bit value in integer format, representing the actual space sector which contains the stator reference vector U_s.

2.58.3.4 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_F32 tr32InVoltage;
SWLIBS_3Syst_F32 tr32PwmABC;
tU32             u32SvmSector;

void main(void)
{
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  // beta component of input voltage vector  = 7.5[V]
  tr32InVoltage.f32Arg1 = FRAC32(12.99/U_MAX);
  tr32InVoltage.f32Arg2 = FRAC32(7.5/U_MAX);

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = 0x7FFF FFFF = FRAC32(1.0000000... )
  // pwmb dutycycle = 0x4000 B88D = FRAC32(0.5000220... )
  // pwmc dutycycle = 0x0000 B88D = FRAC32(0.0000220... )
  // svmSector      = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmSci_F32(&tr32PwmABC,&tr32InVoltage);

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = 0x7FFF FFFF = FRAC32(1.0000000... )
  // pwmb dutycycle = 0x4000 B88D = FRAC32(0.5000220... )
  // pwmc dutycycle = 0x0000 B88D = FRAC32(0.0000220... )
  // svmSector      = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmSci(&tr32PwmABC,&tr32InVoltage,F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = 0x7FFF FFFF = FRAC32(1.0000000... )
  // pwmb dutycycle = 0x4000 B88D = FRAC32(0.5000220... )
  // pwmc dutycycle = 0x0000 B88D = FRAC32(0.0000220... )
  // svmSector      = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmSci(&tr32PwmABC,&tr32InVoltage);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.58.4 Function GMCLIB_SvmSci_F16

2.58.4.1 Declaration

```
tU16 GMCLIB_SvmSci_F16(SWLIBS_3Syst_F16 *pOut, const
SWLIBS_2Syst_F16 *const pIn);
```

2.58.4.2 Arguments

Table 322. GMCLIB_SvmSci_F16 arguments

Type	Name	Direction	Description
SWLIBS_3Syst_F16 *	pOut	**input, output**	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.
const SWLIBS_2Syst_F16 *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector.

2.58.4.3 Return

The function returns a 16-bit value in integer format, representing the actual space sector which contains the stator reference vector U_s.

2.58.4.4 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_F16 tr16InVoltage;
SWLIBS_3Syst_F16 tr16PwmABC;
tU16             u16SvmSector;

void main(void)
{
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  // beta component of input voltage vector  = 7.5[V]
  tr16InVoltage.f16Arg1 = FRAC16(12.99/U_MAX);
  tr16InVoltage.f16Arg2 = FRAC16(7.5/U_MAX);

  // output pwm dutycycles stored in structure referenced by tr16PwmABC
  // pwmA dutycycle = 0x7FFF = FRAC32(1.000... )
  // pwmb dutycycle = 0x4000 = FRAC32(0.500... )
  // pwmc dutycycle = 0x0000 = FRAC32(0.000... )
  // svmSector      = 0x1 [sector]
  u16SvmSector = GMCLIB_SvmSci_F16(&tr16PwmABC,&tr16InVoltage);

  // output pwm dutycycles stored in structure referenced by tr16PwmABC
  // pwmA dutycycle = 0x7FFF = FRAC32(1.000... )
  // pwmb dutycycle = 0x4000 = FRAC32(0.500... )
  // pwmc dutycycle = 0x0000 = FRAC32(0.000... )
  // svmSector      = 0x1 [sector]
  u16SvmSector = GMCLIB_SvmSci(&tr16PwmABC,&tr16InVoltage,F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // output pwm dutycycles stored in structure referenced by tr16PwmABC
  // pwmA dutycycle = 0x7FFF = FRAC32(1.000... )
  // pwmb dutycycle = 0x4000 = FRAC32(0.500... )
  // pwmc dutycycle = 0x0000 = FRAC32(0.000... )
  // svmSector      = 0x1 [sector]
  u16SvmSector = GMCLIB_SvmSci(&tr16PwmABC,&tr16InVoltage);
}
```

2.58.5 Function GMCLIB_SvmSci_FLT

2.58.5.1 Declaration

```
tU32 GMCLIB_SvmSci_FLT(SWLIBS_3Syst_FLT *pOut, const
SWLIBS_2Syst_FLT *const pIn);
```

2.58.5.2 Arguments

Table 323. GMCLIB_SvmSci_FLT arguments

Type	Name	Direction	Description
SWLIBS_3Syst_FLT *	pOut	**input, output**	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.
const SWLIBS_2Syst_FLT *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector.

2.58.5.3 Return

The function returns a 32-bit value in integer format, representing the actual space sector which contains the stator reference vector U_s.

2.58.5.4 Implementation details

The function presumes that the input voltages are normalized to fit the range <-1; 1>. In a typical motor control application, this function is preceded by the function GMCLIB_ElimDcBusRip_FLT which ensures that the voltages are correctly normalized.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.58.5.5 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_FLT tInVoltage;
SWLIBS_3Syst_FLT tPwmABC;
tU32             u32SvmSector;

void main(void)
{
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  // beta component of input voltage vector  = 7.5[V]
  tInVoltage.fltArg1 = (tFloat)(12.99/U_MAX);
  tInVoltage.fltArg2 = (tFloat)(7.5/U_MAX);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = (tFloat)(1.0000000)
  // pwmb dutycycle = (tFloat)(0.5000220)
  // pwmc dutycycle = (tFloat)(0.0000220)
  // u32SvmSector   = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmSci_FLT(&tPwmABC,&tInVoltage);

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = (tFloat)(1.0000000)
  // pwmb dutycycle = (tFloat)(0.5000220)
  // pwmc dutycycle = (tFloat)(0.0000220)
  // u32SvmSector   = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmSci(&tPwmABC,&tInVoltage,FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = (tFloat)(1.0000000)
  // pwmb dutycycle = (tFloat)(0.5000220)
  // pwmc dutycycle = (tFloat)(0.0000220)
  // u32SvmSector   = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmSci(&tPwmABC,&tInVoltage);
}
```

2.59 Function GMCLIB_SvmStd

This function calculates the duty-cycle ratios using the Standard Space Vector Modulation technique.

2.59.1 Description

The GMCLIB_SvmStd function for calculating duty-cycle ratios is widely-used in the modern electric drive. This function calculates appropriate duty-cycle ratios, which are needed for generating the given stator reference voltage vector using a special Space Vector Modulation technique, termed Standard Space Vector Modulation. The basic principle of the Standard Space Vector Modulation Technique can be explained with the help of the power stage diagram in Figure 107 .

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 107. Power stage schematic diagram

Top and bottom switches work in a complementary mode; i.e., if the top switch, S_{At}, is ON, then the corresponding bottom switch, S_{Ab}, is OFF, and vice versa. Considering that value 1 is assigned to the ON state of the top switch, and value 0 is assigned to the ON state of the bottom switch, the switching vector, $[a, b, c]^T$ can be defined. Creating such a vector allows a numerical definition of all possible switching states. In a three-phase power stage configuration (as shown in Figure 107), eight possible switching states (detailed in Figure 108) are feasible.

Figure 108. Basic space vectors

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

These states, together with the resulting instantaneous output line-to-line and phase voltages, are listed in Table 324 .

Table 324. Switching patterns

a	b	c	U_a	U_b	U_c	U_{AB}	U_{BC}	U_{CA}	Vector
0	0	0	0	0	0	0	0	0	O_{000}
1	0	0	$2/3*U_{DCBus}$	$-1/3*U_{DCBus}$	$-1/3*U_{DCBus}$	U_{DCBus}	0	$-U_{DCBus}$	U_0
1	1	0	$1/3*U_{DCBus}$	$1/3*U_{DCBus}$	$-2/3*U_{DCBus}$	0	U_{DCBus}	$-U_{DCBus}$	U_{60}
0	1	0	$-1/3*U_{DCBus}$	$2/3*U_{DCBus}$	$-1/3*U_{DCBus}$	$-U_{DCBus}$	U_{DCBus}	0	U_{120}
0	1	1	$-2/3*U_{DCBus}$	$1/3*U_{DCBus}$	$1/3*U_{DCBus}$	$-U_{DCBus}$	0	U_{DCBus}	U_{180}
0	0	1	$-1/3*U_{DCBus}$	$-1/3*U_{DCBus}$	$2/3*U_{DCBus}$	0	$-U_{DCBus}$	U_{DCBus}	U_{240}
1	0	1	$1/3*U_{DCBus}$	$-2/3*U_{DCBus}$	$1/3*U_{DCBus}$	U_{DCBus}	$-U_{DCBus}$	0	U_{300}
1	1	1	0	0	0	0	0	0	O_{111}

The quantities of the direct-u_α and the quadrature-u_β components of the two-phase orthogonal coordinate system, describing the three-phase stator voltages, are expressed by the Clarke Transformation.

$$U_\alpha = \frac{2}{3} \cdot \left(U_a - \frac{U_b}{2} - \frac{U_c}{2} \right)$$

$$U_\beta = \frac{2}{3} \cdot \left(0 + \frac{\sqrt{3}\, U_b}{2} - \frac{\sqrt{3}\, U_c}{2} \right)$$

Equation GMCLIB_SvmStd_Eq1

The three-phase stator voltages, U_a, U_b, and U_c, are transformed using the Clarke Transformation into the U_α and the U_β components of the two-phase orthogonal coordinate system. The transformation results are listed in Table 325 .

Table 325. Switching patterns and space vectors

a	b	c	u_α	u_β	Vector
0	0	0	0	0	O_{000}
1	0	0	$2/3*U_{DCBus}$	0	U_0
1	1	0	$1/3*U_{DCBus}$	$1/\sqrt{(3)}*U_{DCBus}$	U_{60}
0	1	0	$-1/3*U_{DCBus}$	$1/\sqrt{(3)}*U_{DCBus}$	U_{120}
0	1	1	$-2/3*U_{DCBus}$	0	U_{180}
0	0	1	$-1/3*U_{DCBus}$	$-1/\sqrt{(3)}*U_{DCBus}$	U_{240}
1	0	1	$1/3*U_{DCBus}$	$-1/\sqrt{(3)}*U_{DCBus}$	U_{300}
1	1	1	0	0	O_{111}

Figure 108 graphically depicts some feasible basic switching states (vectors). It is clear that there are six non-zero vectors U_0, U_{60}, U_{120}, U_{180}, U_{240}, U_{300}, and two zero vectors O_{111}, O_{000}, usable for switching. Therefore, the principle of the Standard Space Vector Modulation resides in applying appropriate switching states for a certain time and thus generating a voltage vector identical to the reference one.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 109. Projection of reference voltage vector in sector I

Referring to that principle, an objective of the Standard Space Vector Modulation is an approximation of the reference stator voltage vector U_S with an appropriate combination of the switching patterns composed of basic space vectors. The graphical explanation of this objective is shown in Figure 109 and Figure 110 .

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 110. Detail of the voltage vector projection in sector I

The stator reference voltage vector U_S is phase-advanced by 30° from the axis-α and thus might be generated with an appropriate combination of the adjacent basic switching states U_0 and U_{60}. These figures also indicate the resultant U_α and U_β components for space vectors U_0 and U_{60}

In this case, the reference stator voltage vector U_S is located in Sector I and, as previously mentioned, can be generated with the appropriate duty-cycle ratios of the basic switching states U_{60} and U_0. The principal equations concerning this vector location are:

$$T = T_{60} + T_0 + T_{null}$$

$$U_S = \frac{T_{60}}{T} \cdot U_{60} + \frac{T_0}{T} \cdot U_0$$

Equation GMCLIB_SvmStd_Eq2

where T_{60} and T_0 are the respective duty-cycle ratios for which the basic space vectors U_{60} and U_0 should be applied within the time period T. T_{null} is the course of time for which the null vectors O_{000} and O_{111} are applied. Those duty-cycle ratios can be calculated using equations:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$u_\beta = \frac{T_{60}}{T} \cdot |U_{60}| \cdot \sin^{-1} 60°$$

$$u_a = \frac{T_0}{T} \cdot |U_0| + \frac{u_\beta}{\tan^{-1} 60°}$$

Equation GMCLIB_SvmStd_Eq3

Considering that the normalized magnitudes of the basic space vectors are $|U_{60}| = |U_0| = 2/\sqrt{(3)}$ and by substitution of the trigonometric expressions sin(60°) and tan(60°) by their quantities $2/\sqrt{(3)}$ and $\sqrt{(3)}$, respectively, equations GMCLIB_SvmStd_Eq3 can be rearranged for the unknown duty-cycle ratios T_{60}/T and T_0/T:

$$\frac{T_{60}}{T} = u_\beta$$

$$\frac{T_0}{T} = \frac{1}{2}\left(\sqrt{3} \cdot u_\alpha - u_\beta\right)$$

Equation GMCLIB_SvmStd_Eq4

Figure 111. Projection of the reference voltage vector in sector II

Sector II is depicted in Figure 111 . In this particular case, the reference stator voltage vector U_S is generated by the appropriate duty-cycle ratios of the basic switching states U_{60} and U_{120} . The basic equations describing this sector are:

$$T = T_{120} + T_{60} + T_{null}$$

$$U_S = \frac{T_{120}}{T} \cdot U_{120} + \frac{T_{60}}{T} \cdot U_{60}$$

Equation GMCLIB_SvmStd_Eq5

where T_{120} and T_{60} are the respective duty-cycle ratios for which the basic space vectors U_{120} and U_{60} should be applied within the time period T. These resultant duty-cycle ratios

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

are formed from the auxiliary components termed A and B. The graphical representation of the auxiliary components is shown in Figure 112 .

Figure 112. Detail of the voltage vector projection in sector II

The equations describing those auxiliary time-duration components are:

$$\frac{\sin^{-1}30^{\circ}}{\sin^{-1}120^{\circ}} = \frac{A}{u_a}$$

$$\frac{\sin^{-1}60^{\circ}}{\sin^{-1}120^{\circ}} = \frac{B}{u_a}$$

Equation GMCLIB_SvmStd_Eq6

Equations GMCLIB_SvmStd_Eq11 have been formed using the sine rule. These equations can be rearranged for the calculation of the auxiliary time-duration components A and B. This is done simply by substitution of the trigonometric terms sin(30°), sin(120°) and sin(60°) by their numerical representations 1/2, √(3)/2 and 1/√(3), respectively.

$$A = \frac{1}{\sqrt{3}} \cdot u_\beta$$

$$B = u_\alpha$$

Equation GMCLIB_SvmStd_Eq7

The resultant duty-cycle ratios, T_{120}/T and T_{60}/T, are then expressed in terms of the auxiliary time-duration components defined by equations GMCLIB_SvmStd_Eq13, as follows:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$\frac{T_{120}}{T} \cdot |U_{120}| = A - B$$
$$\frac{T_{60}}{T} \cdot |U_{60}| = A + B$$

Equation GMCLIB_SvmStd_Eq8

With the help of these equations, and also considering the normalized magnitudes of the basic space vectors to be $|U_{120}| = |U_{60}| = 2/\sqrt{(3)}$, the equations expressed for the unknown duty-cycle ratios of basic space vectors T_{120}/T and T_{60}/T can be written:

$$\frac{T_{120}}{T} = \frac{1}{2}\left(u_\beta - \sqrt{3} \cdot u_\alpha\right)$$
$$\frac{T_{60}}{T} = \frac{1}{2}\left(u_\beta + \sqrt{3} \cdot u_\alpha\right)$$

Equation GMCLIB_SvmStd_Eq9

The duty-cycle ratios in remaining sectors can be derived using the same approach. The resulting equations will be similar to those derived for Sector I and Sector II.

To depict duty-cycle ratios of the basic space vectors for all sectors, we define:

- Three auxiliary variables:

$$X = u_\beta$$
$$Y = \frac{1}{2}\left(u_\beta + \sqrt{3} \cdot u_\alpha\right)$$
$$Z = \frac{1}{2}\left(u_\beta - \sqrt{3} \cdot u_\alpha\right)$$

Equation GMCLIB_SvmStd_Eq10

Two expressions t_1 and t_2 generally represent duty-cycle ratios of the basic space vectors in the respective sector; e.g., for the first sector, t_1 and t_2 represent duty-cycle ratios of the basic space vectors U_{60} and U_0; for the second sector, t_1 and t_2 represent duty-cycle ratios of the basic space vectors U_{120} and U_{60}, etc.

For each sector, the expressions t_1 and t_2, in terms of auxiliary variables X, Y and Z, are listed in Table 326 .

Table 326. Determination of t_1 and t_2 expressions

Sector	U_0, U_{60}	U_{60}, U_{120}	U_{120}, U_{180}	U_{180}, U_{240}	U_{240}, U_{300}	U_{300}, U_0
t_1	X	Y	-Y	Z	-Z	-X
t_2	-Z	Z	X	-X	-Y	Y

For the determination of auxiliary variables X, Y and Z equations GMCLIB_SvmStd_Eq10, the sector number is required. This information can be obtained by several approaches. One approach discussed here requires the use of a modified Inverse Clark Transformation to transform the direct-α and quadrature-β components into a balanced three-phase quantity u_{ref1}, u_{ref2} and u_{ref3}, used for a straightforward calculation of the sector number, to be shown later.

$$u_{ref1} = u_\beta$$
$$u_{ref2} = \frac{1}{2}\left(-u_\beta + \sqrt{3} \cdot u_\alpha\right)$$
$$u_{ref3} = \frac{1}{2}\left(-u_\beta - \sqrt{3} \cdot u_\alpha\right)$$

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Equation GMCLIB_SvmStd_Eq11

The modified Inverse Clark Transformation projects the quadrature-u_β component into u_{ref1}, as shown in Figure 113 and Figure 114 , whereas voltages generated by the conventional Inverse Clark Transformation project the u_α component into u_{ref1}.

Figure 113. Direct-ua and quadrature-ub components of stator reference voltage

Figure 113 depicts the u_α and u_β components of the stator reference voltage vector U_S that were calculated by the equations $u_\alpha = \cos(\theta)$ and $u_\beta = \sin(\theta)$, respectively.

Figure 114. Reference Voltages u_ref1, u_ref2 and u_ref3

The Sector Identification Tree, shown in Figure 115 , can be a numerical solution of the approach shown in Figure 114 .

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 115. Identification of the sector number

It should be pointed out that, in the worst case, three simple comparisons are required to precisely identify the sector of the stator reference voltage vector. For example, if the stator reference voltage vector resides according to the one shown in Figure 109 , the stator reference voltage vector is phase-advanced by 30° from the α-axis, which results in the positive quantities of u_{ref1} and u_{ref2} and the negative quantity of u_{ref3}; refer to Figure 114 . If these quantities are used as the inputs to the Sector Identification Tree, the product of those comparisons will be Sector I. Using the same approach identifies Sector II, if the stator reference voltage vector is located according to the one shown in Figure 112 . The variables t_1, t_2 and t_3, representing the switching duty-cycle ratios of the respective three-phase system, are given by the following equations:

$$t_1 = \frac{T\text{-}t_1\text{-}t_2}{2}$$
$$t_2 = t_1 + t_1$$
$$t_3 = t_2 + t_2$$

Equation GMCLIB_SvmStd_Eq12

where T is the switching period, t_1 and t_2 are the duty-cycle ratios (see Table 326) of the basic space vectors, given for the respective sector. Equations GMCLIB_SvmStd_Eq12 are specific solely to the Standard Space Vector Modulation technique; consequently, other Space Vector Modulation techniques discussed later will require deriving different equations.

The next step is to assign the correct duty-cycle ratios, t_1, t_2 and t_3, to the respective motor phases. This is a simple task, accomplished in view of the position of the stator reference voltage vector as shown in Table 327 .

Table 327. Assignment of the duty-cycle ratios to motor phases

Sector	U_0, U_{60}	U_{60}, U_{120}	U_{120}, U_{180}	U_{180}, U_{240}	U_{240}, U_{300}	U_{300}, U_0
pwm_a	t_3	t_2	t_1	t_1	t_2	t_3
pwm_b	t_2	t_3	t_3	t_2	t_1	t_1
pwm_c	t_1	t_1	t_2	t_3	t_3	t_2

The principle of the Space Vector Modulation technique consists in applying the basic voltage vectors U_{XXX} and O_{XXX} for the certain time in such a way that the mean vector, generated by the Pulse Width Modulation approach for the period T, is equal to the original stator reference voltage vector U_S. This provides a great variability of the arrangement of the basic vectors during the PWM period T. Those vectors might be arranged either to lower switching losses or to achieve diverse results, such as centre-aligned PWM, edge-aligned PWM or a minimal number of switching states. A brief discussion of the widely-used centre-aligned PWM follows. Generating the centre-aligned PWM pattern is accomplished practically by comparing the threshold levels, pwm_a, pwm_b

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

and pwm_c with a free-running up-down counter. The timer counts to a 1 (representing the maximum counter value) and then down to a 0. It is supposed that when a threshold level is larger than the timer value, the respective PWM output is active. Otherwise, it is inactive; see Figure 116

Figure 116. Standard space vector modulation technique - centre-aligned PWM

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.59.2 Re-entrancy

The function is re-entrant for a different set of input/output pointers.

2.59.3 Function GMCLIB_SvmStd_F32

2.59.3.1 Declaration

```
tU32 GMCLIB_SvmStd_F32(SWLIBS_3Syst_F32 *pOut, const
SWLIBS_2Syst_F32 *const pIn);
```

2.59.3.2 Arguments

Table 328. GMCLIB_SvmStd_F32 arguments

Type	Name	Direction	Description
SWLIBS_3Syst_F32 *	pOut	**input, output**	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.59.3.3 Return

The function returns a 32-bit value in format INT, representing the actual space sector which contains the stator reference vector U_s.

2.59.3.4 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_F32 tr32InVoltage;
SWLIBS_3Syst_F32 tr32PwmABC;
tU32             u32SvmSector;

void main(void)
{
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  // beta component of input voltage vector  = 7.5[V]
  tr32InVoltage.f32Arg1 = FRAC32(12.99/U_MAX);
  tr32InVoltage.f32Arg2 = FRAC32(7.5/U_MAX);

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = 0x7FFF A2C9 = FRAC32(0.9999888... )
  // pwmb dutycycle = 0x4000 5D35 = FRAC32(0.5000111... )
  // pwmc dutycycle = 0x0000 5D35 = FRAC32(0.0000111... )
  // svmSector      = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmStd_F32(&tr32PwmABC,&tr32InVoltage);

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = 0x7FFF A2C9 = FRAC32(0.9999888... )
  // pwmb dutycycle = 0x4000 5D35 = FRAC32(0.5000111... )
  // pwmc dutycycle = 0x0000 5D35 = FRAC32(0.0000111... )
  // svmSector      = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmStd(&tr32PwmABC,&tr32InVoltage,F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = 0x7FFF A2C9 = FRAC32(0.9999888... )
  // pwmb dutycycle = 0x4000 5D35 = FRAC32(0.5000111... )
  // pwmc dutycycle = 0x0000 5D35 = FRAC32(0.0000111... )
  // svmSector      = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmStd(&tr32PwmABC,&tr32InVoltage);
}
```

2.59.4 Function GMCLIB_SvmStd_F16

2.59.4.1 Declaration

```
tU16 GMCLIB_SvmStd_F16(SWLIBS_3Syst_F16 *pOut, const
SWLIBS_2Syst_F16 *const pIn);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.59.4.2 Arguments

Table 329. GMCLIB_SvmStd_F16 arguments

Type	Name	Direction	Description
SWLIBS_3Syst_F16 *	pOut	**input, output**	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.
const SWLIBS_2Syst_F16 *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector.

2.59.4.3 Return

The function returns a 16-bit value in format INT, representing the actual space sector which contains the stator reference vector U_s.

2.59.4.4 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_F16 tr16InVoltage;
SWLIBS_3Syst_F16 tr16PwmABC;
tU16             u16SvmSector;

void main(void)
{
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  // beta component of input voltage vector  = 7.5[V]
  tr16InVoltage.f16Arg1 = FRAC16(12.99/U_MAX);
  tr16InVoltage.f16Arg2 = FRAC16(7.5/U_MAX);

  // output pwm dutycycles stored in structure referenced by tr16PwmABC
  // pwmA dutycycle = 0x7FFF = FRAC16(0.9999... )
  // pwmb dutycycle = 0x4000 = FRAC16(0.5000... )
  // pwmc dutycycle = 0x0000 = FRAC16(0.0000... )
  // svmSector      = 0x1 [sector]
  u16SvmSector = GMCLIB_SvmStd_F16(&tr16PwmABC,&tr16InVoltage);

  // output pwm dutycycles stored in structure referenced by tr16PwmABC
  // pwmA dutycycle = 0x7FFF = FRAC16(0.9999... )
  // pwmb dutycycle = 0x4000 = FRAC16(0.5000... )
  // pwmc dutycycle = 0x0000 = FRAC16(0.0000... )
  // svmSector      = 0x1 [sector]
  u16SvmSector = GMCLIB_SvmStd(&tr16PwmABC,&tr16InVoltage,F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output pwm dutycycles stored in structure referenced by tr16PwmABC
  // pwmA dutycycle = 0x7FFF = FRAC16(0.9999... )
  // pwmb dutycycle = 0x4000 = FRAC16(0.5000... )
  // pwmc dutycycle = 0x0000 = FRAC16(0.0000... )
  // svmSector      = 0x1 [sector]
  u16SvmSector = GMCLIB_SvmStd(&tr16PwmABC,&tr16InVoltage);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.59.5 Function GMCLIB_SvmStd_FLT

2.59.5.1 Declaration

```
tU32 GMCLIB_SvmStd_FLT(SWLIBS_3Syst_FLT *pOut, const
SWLIBS_2Syst_FLT *const pIn);
```

2.59.5.2 Arguments

Table 330. GMCLIB_SvmStd_FLT arguments

Type	Name	Direction	Description
SWLIBS_3Syst_FLT *	pOut	**input, output**	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.
const SWLIBS_2Syst_FLT *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector.

2.59.5.3 Return

The function returns a 32-bit value in format INT, representing the actual space sector which contains the stator reference vector U_s.

2.59.5.4 Implementation details

The function presumes that the input voltages are normalized to fit the range <-1; 1>. In a typical motor control application, this function is preceded by the function GMCLIB_ElimDcBusRip_FLT which ensures that the voltages are correctly normalized.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.59.5.5 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_FLT tInVoltage;
SWLIBS_3Syst_FLT tPwmABC;
tU32             u32SvmSector;

void main(void)
{
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  // beta component of input voltage vector  = 7.5[V]
  tInVoltage.fltArg1 = (tFloat)(12.99/U_MAX);
  tInVoltage.fltArg2 = (tFloat)(7.5/U_MAX);

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = (tFloat)(0.9999888)
  // pwmb dutycycle = (tFloat)(0.5000111)
  // pwmc dutycycle = (tFloat)(0.0000111)
  // u32SvmSector   = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmStd_FLT(&tPwmABC,&tInVoltage);

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // pwmA dutycycle = (tFloat)(0.9999888)
  // pwmb dutycycle = (tFloat)(0.5000111)
  // pwmc dutycycle = (tFloat)(0.0000111)
  // u32SvmSector   = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmStd(&tPwmABC,&tInVoltage,FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = (tFloat)(0.9999888)
  // pwmb dutycycle = (tFloat)(0.5000111)
  // pwmc dutycycle = (tFloat)(0.0000111)
  // u32SvmSector   = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmStd(&tPwmABC,&tInVoltage);
}
```

2.60 Function GMCLIB_SvmStd6Ph

The function implements the the six-phase Space Vector Modulation algorithm.

2.60.1 Description

This function transforms the values from the two-phase (α-β) orthogonal coordinate system to the six-phase system consisting of the two sets of starconnected three-phase stator windings (f32A-f32B-f32C) and (f32D-f32E-f32F) using the Space Vector Modulation algorithm.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.60.2 Re-entrancy

The function is re-entrant.

2.60.3 Function GMCLIB_SvmStd6Ph_F32

2.60.3.1 Declaration

```
void GMCLIB_SvmStd6Ph_F32(SWLIBS_6Syst_F32 *const pOutPwm,
SWLIBS_2Syst_U32 *const pOutSector, const SWLIBS_2Syst_F32 *const
pIn, const GMCLIB_SVMSTD_6PH_T_F32 *const pParam);
```

2.60.3.2 Arguments

Table 331. GMCLIB_SvmStd6Ph_F32 arguments

Type	Name	Direction	Description
Commented parameter pOut does not exist in function GMCLIB_SvmStd6Ph_F32.	pOut	**output**	Pointer to the structure containing calculated duty-cycle ratios of the six-phase machine consisting of two sets of starconnected three-phase stator windings f32A-f32B-f32C and f32D-f32E-f32F. Arguments of the structure contain fixed point 32-bit values.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 331. GMCLIB_SvmStd6Ph_F32 arguments...*continued*

Type	Name	Direction	Description
SWLIBS_2Syst_U32 *const	pOutSector	**output**	Pointer to the structure containing calculated space sectors for each three-phase stator windings of the six-phase system. Arguments of the structure contain 32-bit unsigned integer values.
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector. Arguments of the structure contain fixed point 32-bit values.
const GMCLIB_SVMSTD_6PH_T_F32 *const	pParam	**output**	Pointer to the structure containing the transformation coefficient calculated by GMCLIB_SvmStd6PhInit function. Arguments of the structure contain fixed point 32-bit values.

2.60.3.3 Code Example

```
#include "gmclib.h"

GMCLIB_SVMSTD_6PH_T_F32 pParam;

void main(void)
{
  tFrac32 f32Shift;

  // Initialize GMCLIB_SvmStd6Ph transformation parameters

  // input shift -30° (-pi/6) normalized into [-1, 1) interval
  f32Shift = FRAC32(-1.0/6.0);

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_SvmStd6PhInit_F32(&pParam, f32Shift);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_SvmStd6PhInit(&pParam, f32Shift,F32);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_SvmStd6PhInit(&pParam, f32Shift);

  while(1);
}

// Periodical function or interrupt
void loop(void)
{
  SWLIBS_6Syst_F32 pOutPwm;
  SWLIBS_2Syst_U32 pOutSectors;
  SWLIBS_2Syst_F32 pIn;

  // Read the A/D, Transform from 6-phase to d-q, PI controllers, Inverse Park transform etc.
  // (...)

  // Calculate alpha-beta values:

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_SvmStd6Ph_F32(&pOutPwm, &pOutSectors, &pIn, &pParam);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_SvmStd6Ph(&pOutPwm, &pOutSectors, &pIn, &pParam, F32);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_SvmStd6Ph(&pOutPwm, &pOutSectors, &pIn, &pParam);
}
```

2.60.4 Function GMCLIB_SvmStd6Ph_F16

2.60.4.1 Declaration

```
void GMCLIB_SvmStd6Ph_F16(SWLIBS_6Syst_F16 *const pOutPwm,
SWLIBS_2Syst_U16 *const pOutSector, const SWLIBS_2Syst_F16 *const
pIn, const GMCLIB_SVMSTD_6PH_T_F16 *const pParam);
```

2.60.4.2 Arguments

Table 332. GMCLIB_SvmStd6Ph_F16 arguments

Type	Name	Direction	Description
Commented parameter pOut does not exist in function GMCLIB_SvmStd6Ph_F16.	pOut	**output**	Pointer to the structure containing calculated duty-cycle ratios of the six-phase machine consisting of two sets of starconnected three-phase stator windings f16A-f16B-f16C and f16D-f16E-f16F. Arguments of the structure contain fixed point 16-bit values.
SWLIBS_2Syst_U16 *const	pOutSector	**output**	Pointer to the structure containing calculated space sectors for each three-phase stator windings of the six-phase system. Arguments of the structure contain 16-bit unsigned integer values.
const SWLIBS_2Syst_F16 *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector. Arguments of the structure contain fixed point 16-bit values.
const GMCLIB_SVMSTD_6PH_T_F16 *const	pParam	**output**	Pointer to the structure containing the transformation coefficient calculated by GMCLIB_SvmStd6PhInit function. Arguments of the structure contain fixed point 16-bit values.

2.60.4.3 Code Example

```
#include "gmclib.h"

GMCLIB_SVMSTD_6PH_T_F16 pParam;

void main(void)
{
  tFrac16 f16Shift;

  // Initialize GMCLIB_SvmStd6Ph transformation parameters

  // input shift -30° (-pi/6) normalized into [-1, 1) interval
  f16Shift = FRAC16(-1.0/6.0);

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_SvmStd6PhInit_F16(&pParam, f16Shift);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_SvmStd6PhInit(&pParam, f16Shift,F16);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 16-bit fractional implementation is selected as default.
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  GMCLIB_SvmStd6PhInit(&pParam, f16Shift);

  while(1);
}

// Periodical function or interrupt
void loop(void)
{
  SWLIBS_6Syst_F16 pOutPwm;
  SWLIBS_2Syst_U16 pOutSectors;
  SWLIBS_2Syst_F16 pIn;

  // Read the A/D, Transform from 6-phase to d-q, PI controllers, Inverse Park transform etc.
  // (...)

  // Calculate alpha-beta values:

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_SvmStd6Ph_F16(&pOutPwm, &pOutSectors, &pIn, &pParam);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_SvmStd6Ph(&pOutPwm, &pOutSectors, &pIn, &pParam, F16);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 16-bit fractional implementation is selected as default.
  GMCLIB_SvmStd6Ph(&pOutPwm, &pOutSectors, &pIn, &pParam);
}
```

2.60.5 Function GMCLIB_SvmStd6Ph_FLT

2.60.5.1 Declaration

```
void GMCLIB_SvmStd6Ph_FLT(SWLIBS_6Syst_FLT *const pOutPwm,
SWLIBS_2Syst_U32 *const pOutSector, const SWLIBS_2Syst_FLT *const
pIn, const GMCLIB_SVMSTD_6PH_T_FLT *const pParam);
```

2.60.5.2 Arguments

Table 333. GMCLIB_SvmStd6Ph_FLT arguments

Type	Name	Direction	Description
SWLIBS_2Syst_U32 *const	pOutSector	**output**	Pointer to the structure containing calculated space sectors for each three-phase stator windings of the six-phase system. Arguments of the structure contain 32-bit unsigned integer values.
const SWLIBS_2Syst_FLT *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector. Arguments of the structure contain single precision floating-point values.
const GMCLIB_SVMSTD_6PH_T_FLT *const	pParam	**output**	Pointer to the structure containing the transformation coefficient calculated by GMCLIB_SvmStd6PhInit function. Arguments of the structure contain single precision floating-point values.

2.60.5.3 Implementation details

pOut Pointer to the structure containing calculated duty-cycle ratios of the six-phase machine consisting of two sets of starconnected three-phase stator windings fltA-fltB-fltC and fltD-fltE-fltF. Arguments of the structure contain single precision floating-point values.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The function presumes that the input voltages are normalized to fit the range <-1; 1>. In a typical motor control application, this function is preceded by the function GMCLIB_ElimDcBusRip_FLT which ensures that the voltages are correctly normalized.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.60.5.4 Code Example

```
#include "gmclib.h"

GMCLIB_SVMSTD_6PH_T_F16 pParam;

void main(void)
{
  tFrac32 fltShift;

  // Initialize GMCLIB_SvmStd6Ph transformation parameters

  // input shift -30° (-pi/6)
  fltShift = -FLOAT_PI_DIVBY_6;

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_SvmStd6PhInit_FLT(&pParam, fltShift);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_SvmStd6PhInit(&pParam, fltShift,FLT);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_SvmStd6PhInit(&pParam, fltShift);

  while(1);
}

// Periodical function or interrupt
void loop(void)
{
  SWLIBS_6Syst_FLT pOutPwm;
  SWLIBS_2Syst_U32 pOutSectors;
  SWLIBS_2Syst_FLT pIn;

  // Read the A/D, Transform from 6-phase to d-q, PI controllers, Inverse Park transform etc.
  // (...)

  // Calculate alpha-beta values:

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_SvmStd6Ph_FLT(&pOutPwm, &pOutSectors, &pIn, &pParam);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_SvmStd6Ph(&pOutPwm, &pOutSectors, &pIn, &pParam, FLT);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if single precision floating point implementation selected as default
  GMCLIB_SvmStd6Ph(&pOutPwm, &pOutSectors, &pIn, &pParam);
}
```

2.60.6 Function GMCLIB_SvmStd6PhInit

The function implements the initialization function required to run before GMCLIB_SvmStd6Ph function.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.60.6.1 Description

GMCLIB_SvmStd6PhInit calculates the transformation coefficients required for GMCLIB_SvmStd6Ph function from the shift between two sets of star-connected three phase stator windings, according to the following equations:

$$AlphaD = \sin(-\theta)$$
$$AlphaE = \sin\left(-\theta - \frac{2}{3}\pi\right)$$
$$AlphaF = \sin\left(-\theta + \frac{2}{3}\pi\right)$$
$$BetaD = \cos(-\theta)$$
$$BetaE = \cos\left(-\theta - \frac{2}{3}\pi\right)$$
$$BetaF = \cos\left(-\theta + \frac{2}{3}\pi\right)$$

Equation GMCLIB_SvmStd6PhInit_Eq1

where θ is the shift between the two windings of the six-phase machine.

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.60.6.2 Re-entrancy

The function is re-entrant.

2.60.6.3 Function GMCLIB_SvmStd6PhInit_F32

2.60.6.3.1 Declaration

```
void GMCLIB_SvmStd6PhInit_F32(GMCLIB_SVMSTD_6PH_T_F32 *const
pParam, tFrac32 f32WindingShift);
```

2.60.6.3.2 Arguments

Table 334. GMCLIB_SvmStd6PhInit_F32 arguments

Type	Name	Direction	Description
tFrac32	f32WindingShift	**input**	Input argument is a 32-bit number that contains an angle in radians from interval [-π, π) normalized between [-1, 1). This angle represents the shift between two sets of star-connected three phase stator windings.
GMCLIB_SVMSTD_6PH_T_F32 *const	pParam	**output**	Pointer to the structure containing the transformation coefficient required for GMCLIB_SvmStd6Ph function. Arguments of the structure contain fixed point 32-bit values.

2.60.6.3.3 Code Example

```
#include "gmclib.h"

GMCLIB_SVMSTD_6PH_T_F32 pParam;

void main(void)
{
  tFrac32 f32Shift;

  // Initialize GMCLIB_SvmStd6Ph transformation parameters
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // input shift -30° (-pi/6) normalized into [-1, 1) interval
  f32Shift = FRAC32(-1.0/6.0);

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_SvmStd6PhInit_F32(&pParam, f32Shift);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_SvmStd6PhInit(&pParam, f32Shift,F32);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_SvmStd6PhInit(&pParam, f32Shift);

  while(1);
}

// Periodical function or interrupt
void loop(void)
{
  SWLIBS_6Syst_F32 pOutPwm;
  SWLIBS_2Syst_U32 pOutSectors;
  SWLIBS_2Syst_F32 pIn;

  // Read the A/D, Transform from 6-phase to d-q, PI controllers, Inverse Park transform etc.
  // (...)

  // Calculate alpha-beta values:

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_SvmStd6Ph_F32(&pOutPwm, &pOutSectors, &pIn, &pParam);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_SvmStd6Ph(&pOutPwm, &pOutSectors, &pIn, &pParam, F32);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_SvmStd6Ph(&pOutPwm, &pOutSectors, &pIn, &pParam);
}
```

2.60.6.4 Function GMCLIB_SvmStd6PhInit_FLT

2.60.6.4.1 Declaration

```
void GMCLIB_SvmStd6PhInit_FLT(GMCLIB_SVMSTD_6PH_T_FLT *const
pParam, tFloat fltWindingShift);
```

2.60.6.4.2 Arguments

Table 335. GMCLIB_SvmStd6PhInit_FLT arguments

Type	Name	Direction	Description
tFloat	fltWindingShift	**input**	Input argument is a single precision floating-point number that contains an angle in radians from interval [- π, π). This angle represents the shift between two sets of star-connected three phase stator windings.
GMCLIB_SVMSTD_6PH_T_FLT *const	pParam	**output**	Pointer to the structure containing the transformation coefficient required for GMCLIB_SvmStd6Ph function. Arguments of the structure contain single precision floating-point values.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.60.6.4.3 Code Example

```
#include "gmclib.h"

GMCLIB_SVMSTD_6PH_T_F16 pParam;

void main(void)
{
  tFrac32 fltShift;

  // Initialize GMCLIB_SvmStd6Ph transformation parameters

  // input shift -30° (-pi/6)
  fltShift = -FLOAT_PI_DIVBY_6;

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_SvmStd6PhInit_FLT(&pParam, fltShift);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_SvmStd6PhInit(&pParam, fltShift,FLT);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if 32-bit fractional implementation is selected as default.
  GMCLIB_SvmStd6PhInit(&pParam, fltShift);

  while(1);
}

// Periodical function or interrupt
void loop(void)
{
  SWLIBS_6Syst_FLT pOutPwm;
  SWLIBS_2Syst_U32 pOutSectors;
  SWLIBS_2Syst_FLT pIn;

  // Read the A/D, Transform from 6-phase to d-q, PI controllers, Inverse Park transform etc.
  // (...)

  // Calculate alpha-beta values:

  // Alternative 1: API call with postfix
  // (only one alternative shall be used).
  GMCLIB_SvmStd6Ph_FLT(&pOutPwm, &pOutSectors, &pIn, &pParam);

  // Alternative 2: API call with implementation parameter
  // (only one alternative shall be used).
  GMCLIB_SvmStd6Ph(&pOutPwm, &pOutSectors, &pIn, &pParam, FLT);

  // Alternative 3: API call with global configuration of implementation
  // (only one alternative shall be used). This alternative is available
  // only if single precision floating point implementation selected as default
  GMCLIB_SvmStd6Ph(&pOutPwm, &pOutSectors, &pIn, &pParam);
}
```

2.61 Function GMCLIB_SvmU0n

This function calculates the duty-cycle ratios using the special Space Vector Modulation technique, termed Space Vector Modulation with O_{000} Nulls.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.61.1 Description

The GMCLIB_SvmU0n function for calculating duty-cycle ratios is widely-used in the modern electric drive. This function calculates appropriate duty-cycle ratios, which are needed for generating the given stator reference voltage vector using a special Space Vector Modulation technique, termed Space Vector Modulation with O_{000} Nulls.

The derivation approach of the Space Vector Modulation technique with O_{000} Nulls is identical, in many aspects, to the approach presented in GMCLIB_SvmStd. However, a distinct difference lies in the definition of the variables t_1, t_2 and t_3 that represents switching duty-cycle ratios of the respective phases:

$$t_1 = 0$$
$$t_2 = t_1 + t_1$$
$$t_3 = t_2 + t_2$$

Equation GMCLIB_SvmU0n_Eq1

where T is the switching period and t_1 and t_2 are duty-cycle ratios of basic space vectors that are defined for the respective sector in Table 326 .

The generally-used center-alligned PWM is discussed briefly in the following sections. Generating the centre-aligned PWM pattern is accomplished practically by comparing the threshold levels, pwm_a, pwm_b and pwm_c with a free-running up-down counter. The timer counts to a 1 (representing the maximum counter value) and then down to a 0. It is supposed that when a threshold level is larger than the timer value, the respective PWM output is active. Otherwise, it is inactive; see Figure 117

Figure 117. Space vector modulation technique with 0_{000} Nulls - centre-aligned PWM

Figure 118 shows calculated waveforms of the duty cycle ratios using Space Vector Modulation with O_{000} Nulls.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 118. Space vector modulation technique with 0_{000} Nulls

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.61.2 Re-entrancy

The function is re-entrant for a different set of input/output pointers.

2.61.3 Function GMCLIB_SvmU0n_F32

2.61.3.1 Declaration

```
tU32 GMCLIB_SvmU0n_F32(SWLIBS_3Syst_F32 *pOut, const
SWLIBS_2Syst_F32 *const pIn);
```

2.61.3.2 Arguments

Table 336. GMCLIB_SvmU0n_F32 arguments

Type	Name	Direction	Description
SWLIBS_3Syst_F32 *	pOut	**input, output**	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.61.3.3 Return

The function returns a 32-bit value in format INT, representing the actual space sector which contains the stator reference vector U_s.

2.61.3.4 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_F32 tr32InVoltage;
SWLIBS_3Syst_F32 tr32PwmABC;
tU32             u32SvmSector;

void main(void)
{
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  // beta component of input voltage vector  = 7.5[V]
  tr32InVoltage.f32Arg1 = FRAC32(12.99/U_MAX);
  tr32InVoltage.f32Arg2 = FRAC32(7.5/U_MAX);

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = 0x7FFF 4772 = FRAC32(0.9999788... )
  // pwmb dutycycle = 0x4000 0000 = FRAC32(0.5000000... )
  // pwmc dutycycle = 0x0000 0000 = FRAC32(0.0000000... )
  // svmSector      = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmU0n_F32(&tr32PwmABC,&tr32InVoltage);

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = 0x7FFF 4772 = FRAC32(0.9999788... )
  // pwmb dutycycle = 0x4000 0000 = FRAC32(0.5000000... )
  // pwmc dutycycle = 0x0000 0000 = FRAC32(0.0000000... )
  // svmSector      = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmU0n(&tr32PwmABC,&tr32InVoltage,F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = 0x7FFF 4772 = FRAC32(0.9999788... )
  // pwmb dutycycle = 0x4000 0000 = FRAC32(0.5000000... )
  // pwmc dutycycle = 0x0000 0000 = FRAC32(0.0000000... )
  // svmSector      = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmU0n(&tr32PwmABC,&tr32InVoltage);
}
```

2.61.4 Function GMCLIB_SvmU0n_F16

2.61.4.1 Declaration

```
tU16 GMCLIB_SvmU0n_F16(SWLIBS_3Syst_F16 *pOut, const
SWLIBS_2Syst_F16 *const pIn);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.61.4.2 Arguments

Table 337. GMCLIB_SvmU0n_F16 arguments

Type	Name	Direction	Description
SWLIBS_3Syst_F16 *	pOut	**input, output**	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.
const SWLIBS_2Syst_F16 *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector.

2.61.4.3 Return

The function returns a 16-bit value in format INT, representing the actual space sector which contains the stator reference vector U_s.

2.61.4.4 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_F16 tr16InVoltage;
SWLIBS_3Syst_F16 tr16PwmABC;
tU16             u16SvmSector;

void main(void)
{
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  // beta component of input voltage vector  = 7.5[V]
  tr16InVoltage.f16Arg1 = FRAC16(12.99/U_MAX);
  tr16InVoltage.f16Arg2 = FRAC16(7.5/U_MAX);

  // output pwm dutycycles stored in structure referenced by tr16PwmABC
  // pwmA dutycycle = 0x7FFF = FRAC16(0.9999... )
  // pwmb dutycycle = 0x4000 = FRAC16(0.5000... )
  // pwmc dutycycle = 0x0000 = FRAC16(0.0000... )
  // svmSector      = 0x1 [sector]
  u16SvmSector = GMCLIB_SvmU0n_F16(&tr16PwmABC,&tr16InVoltage);

  // output pwm dutycycles stored in structure referenced by tr16PwmABC
  // pwmA dutycycle = 0x7FFF = FRAC16(0.9999... )
  // pwmb dutycycle = 0x4000 = FRAC16(0.5000... )
  // pwmc dutycycle = 0x0000 = FRAC16(0.0000... )
  // svmSector      = 0x1 [sector]
  u16SvmSector = GMCLIB_SvmU0n(&tr16PwmABC,&tr16InVoltage,F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output pwm dutycycles stored in structure referenced by tr16PwmABC
  // pwmA dutycycle = 0x7FFF = FRAC16(0.9999... )
  // pwmb dutycycle = 0x4000 = FRAC16(0.5000... )
  // pwmc dutycycle = 0x0000 = FRAC16(0.0000... )
  // svmSector      = 0x1 [sector]
  u16SvmSector = GMCLIB_SvmU0n(&tr16PwmABC,&tr16InVoltage);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.61.5 Function GMCLIB_SvmU0n_FLT

2.61.5.1 Declaration

```
tU32 GMCLIB_SvmU0n_FLT(SWLIBS_3Syst_FLT *pOut, const
SWLIBS_2Syst_FLT *const pIn);
```

2.61.5.2 Arguments

Table 338. GMCLIB_SvmU0n_FLT arguments

Type	Name	Direction	Description
SWLIBS_3Syst_FLT *	pOut	**input, output**	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.
const SWLIBS_2Syst_FLT *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector.

2.61.5.3 Return

The function returns a 32-bit value in format INT, representing the actual space sector which contains the stator reference vector U_s.

2.61.5.4 Implementation details

The function presumes that the input voltages are normalized to fit the range <-1; 1>. In a typical motor control application, this function is preceded by the function GMCLIB_ElimDcBusRip_FLT which ensures that the voltages are correctly normalized.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.61.5.5 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_FLT tInVoltage;
SWLIBS_3Syst_FLT tPwmABC;
tU32             u32SvmSector;

void main(void)
{
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  // beta component of input voltage vector  = 7.5[V]
  tInVoltage.fltArg1 = (tFloat)(12.99/U_MAX);
  tInVoltage.fltArg2 = (tFloat)(7.5/U_MAX);

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = (tFloat)(0.9999780)
  // pwmb dutycycle = (tFloat)(0.5000000)
  // pwmc dutycycle = (tFloat)(0.0000000)
  // u32SvmSector   = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmU0n_FLT(&tPwmABC,&tInVoltage);

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // pwmA dutycycle = (tFloat)(0.9999780)
  // pwmb dutycycle = (tFloat)(0.5000000)
  // pwmc dutycycle = (tFloat)(0.0000000)
  // u32SvmSector   = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmU0n(&tPwmABC,&tInVoltage,FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = (tFloat)(0.9999780)
  // pwmb dutycycle = (tFloat)(0.5000000)
  // pwmc dutycycle = (tFloat)(0.0000000)
  // u32SvmSector   = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmU0n(&tPwmABC,&tInVoltage);
}
```

2.62 Function GMCLIB_SvmU7n

This function calculates the duty-cycle ratios using the special Space Vector Modulation technique, termed Space Vector Modulation with O_{111} Ones.

2.62.1 Description

The GMCLIB_SvmU7n function for calculating duty-cycle ratios is widely-used in the modern electric drive. This function calculates appropriate duty-cycle ratios, which are needed for generating the given stator reference voltage vector using a special Space Vector Modulation technique, termed Space Vector Modulation with O_{111} Ones.

The derivation approach of the Space Vector Modulation technique with O_{111} Ones is identical, in many aspects, to the approach presented in GMCLIB_SvmStd. However, a distinct difference lies in the definition of the variables t_1, t_2 and t_3 that represents switching duty-cycle ratios of the respective phases:

$$t_1 = T - t_1 - t_2$$
$$t_2 = t_1 + t_1$$
$$t_3 = t_2 + t_2$$

Equation GMCLIB_SvmU7n_Eq1

where T is the switching period and t_1 and t_2 are duty-cycle ratios of basic space vectors that are defined for the respective sector in Table 326 .

The generally-used center-alligned PWM is discussed briefly in the following sections. Generating the centre-aligned PWM pattern is accomplished practically by comparing the threshold levels, pwm_a, pwm_b and pwm_c with a free-running up-down counter. The timer counts to a 1 (representing the maximum counter value) and then down to a 0. It is supposed that when a threshold level is larger than the timer value, the respective PWM output is active. Otherwise, it is inactive; see Figure 119

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 119. Space vector modulation technique with 0_{111} Ones - centre-aligned PWM

Figure 120 shows calculated waveforms of the duty cycle ratios using Space Vector Modulation with O_{111} Ones.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Figure 120. Space vector modulation technique with 0_{111} Ones

Note: *The input/output pointers must contain valid addresses, otherwise a fault may occur (MemManage, BusFault, UsageFault, HardFault).*

2.62.2 Re-entrancy

The function is re-entrant for a different set of input/output pointers.

2.62.3 Function GMCLIB_SvmU7n_F32

2.62.3.1 Declaration

```
tU32 GMCLIB_SvmU7n_F32(SWLIBS_3Syst_F32 *pOut, const
SWLIBS_2Syst_F32 *const pIn);
```

2.62.3.2 Arguments

Table 339. GMCLIB_SvmU7n_F32 arguments

Type	Name	Direction	Description
SWLIBS_3Syst_F32 *	pOut	**input, output**	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 339. GMCLIB_SvmU7n_F32 arguments...*continued*

Type	Name	Direction	Description
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector.

2.62.3.3 Return

The function returns a 32-bit value in format INT, representing the actual space sector which contains the stator reference vector U_s.

2.62.3.4 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_F32 tr32InVoltage;
SWLIBS_3Syst_F32 tr32PwmABC;
tU32             u32SvmSector;

void main(void)
{
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  // beta component of input voltage vector  = 7.5[V]
  tr32InVoltage.f32Arg1 = FRAC32(12.99/U_MAX);
  tr32InVoltage.f32Arg2 = FRAC32(7.5/U_MAX);

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = 0x7FFF FFFF = FRAC32(1.0000000... )
  // pwmb dutycycle = 0x4000 B88D = FRAC32(0.5000220... )
  // pwmc dutycycle = 0x0000 B88D = FRAC32(0.0000220... )
  // svmSector      = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmU7n_F32(&tr32PwmABC,&tr32InVoltage);

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = 0x7FFF FFFF = FRAC32(1.0000000... )
  // pwmb dutycycle = 0x4000 B88D = FRAC32(0.5000220... )
  // pwmc dutycycle = 0x0000 B88D = FRAC32(0.0000220... )
  // svmSector      = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmU7n(&tr32PwmABC,&tr32InVoltage,F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = 0x7FFF FFFF = FRAC32(1.0000000... )
  // pwmb dutycycle = 0x4000 B88D = FRAC32(0.5000220... )
  // pwmc dutycycle = 0x0000 B88D = FRAC32(0.0000220... )
  // svmSector      = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmU7n(&tr32PwmABC,&tr32InVoltage);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.62.4 Function GMCLIB_SvmU7n_F16

2.62.4.1 Declaration

```
tU16 GMCLIB_SvmU7n_F16(SWLIBS_3Syst_F16 *pOut, const
SWLIBS_2Syst_F16 *const pIn);
```

2.62.4.2 Arguments

Table 340. GMCLIB_SvmU7n_F16 arguments

Type	Name	Direction	Description
SWLIBS_3Syst_F16 *	pOut	**input, output**	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.
const SWLIBS_2Syst_F16 *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector.

2.62.4.3 Return

The function returns a 16-bit value in format INT, representing the actual space sector which contains the stator reference vector U_s.

2.62.4.4 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_F16 tr16InVoltage;
SWLIBS_3Syst_F16 tr16PwmABC;
tU16             u16SvmSector;

void main(void)
{
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  // beta component of input voltage vector  = 7.5[V]
  tr16InVoltage.f16Arg1 = FRAC16(12.99/U_MAX);
  tr16InVoltage.f16Arg2 = FRAC16(7.5/U_MAX);

  // output pwm dutycycles stored in structure referenced by tr16PwmABC
  // pwmA dutycycle = 0x7FFF = FRAC32(1.000... )
  // pwmb dutycycle = 0x4000 = FRAC32(0.500... )
  // pwmc dutycycle = 0x0000 = FRAC32(0.000... )
  // svmSector      = 0x1 [sector]
  u16SvmSector = GMCLIB_SvmU7n_F16(&tr16PwmABC,&tr16InVoltage);

  // output pwm dutycycles stored in structure referenced by tr16PwmABC
  // pwmA dutycycle = 0x7FFF = FRAC32(1.000... )
  // pwmb dutycycle = 0x4000 = FRAC32(0.500... )
  // pwmc dutycycle = 0x0000 = FRAC32(0.000... )
  // svmSector      = 0x1 [sector]
  u16SvmSector = GMCLIB_SvmU7n(&tr16PwmABC,&tr16InVoltage,F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // output pwm dutycycles stored in structure referenced by tr16PwmABC
  // pwmA dutycycle = 0x7FFF = FRAC32(1.000... )
  // pwmb dutycycle = 0x4000 = FRAC32(0.500... )
  // pwmc dutycycle = 0x0000 = FRAC32(0.000... )
  // svmSector      = 0x1 [sector]
  u16SvmSector = GMCLIB_SvmU7n(&tr16PwmABC,&tr16InVoltage);
}
```

2.62.5 Function GMCLIB_SvmU7n_FLT

2.62.5.1 Declaration

```
tU32 GMCLIB_SvmU7n_FLT(SWLIBS_3Syst_FLT *pOut, const
SWLIBS_2Syst_FLT *const pIn);
```

2.62.5.2 Arguments

Table 341. GMCLIB_SvmU7n_FLT arguments

Type	Name	Direction	Description
SWLIBS_3Syst_FLT *	pOut	**input, output**	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.
const SWLIBS_2Syst_FLT *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector.

2.62.5.3 Return

The function returns a 32-bit value in format INT, representing the actual space sector which contains the stator reference vector U_s.

2.62.5.4 Implementation details

The function presumes that the input voltages are normalized to fit the range <-1; 1>. In a typical motor control application, this function is preceded by the function GMCLIB_ElimDcBusRip_FLT which ensures that the voltages are correctly normalized.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.62.5.5 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_FLT tInVoltage;
SWLIBS_3Syst_FLT tPwmABC;
tU32             u32SvmSector;

void main(void)
{
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  // beta component of input voltage vector  = 7.5[V]
  tInVoltage.fltArg1 = (tFloat)(12.99/U_MAX);
  tInVoltage.fltArg2 = (tFloat)(7.5/U_MAX);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = (tFloat)(1.0000000)
  // pwmb dutycycle = (tFloat)(0.5000220)
  // pwmc dutycycle = (tFloat)(0.0000220)
  // u32SvmSector   = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmU7n_FLT(&tPwmABC,&tInVoltage);

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = (tFloat)(1.0000000)
  // pwmb dutycycle = (tFloat)(0.5000220)
  // pwmc dutycycle = (tFloat)(0.0000220)
  // u32SvmSector   = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmU7n(&tPwmABC,&tInVoltage,FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle = (tFloat)(1.0000000)
  // pwmb dutycycle = (tFloat)(0.5000220)
  // pwmc dutycycle = (tFloat)(0.0000220)
  // u32SvmSector   = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmU7n(&tPwmABC,&tInVoltage);
}
```

2.63 Function GMCLIB_Vmag2coord

This function calculates the coordinate of the vector from the magnitude and the second coordinate.

2.63.1 Description

If the input vector is defined by the magnitude *r* and one of the coordinates *x* or *y*, then the second coordinate is calculated by one of the following equations:

$$x = \sqrt{r^2 - y^2}$$

$$y = \sqrt{r^2 - x^2}$$

Equation GMCLIB_Vmag2coord_Eq1

2.63.2 Re-entrancy

The function is re-entrant.

2.63.3 Function GMCLIB_Vmag2coord_F32

2.63.3.1 Declaration

```
tFrac32 GMCLIB_Vmag2coord_F32(tFrac32 f32InMag, tFrac32
f32InCoord);
```


© NXP B.V. 2023. All rights reserved.

2.63.3.2 Arguments

Table 342. GMCLIB_Vmag2coord_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InMag	**input**	Input magnitude of the vector.
tFrac32	f32InCoord	**input**	Input coordinate of the vector.

2.63.3.3 Return

Second coordinate of the vector.

2.63.3.4 Implementation details

The input values as well as output value is considered as 32-bit fractional values representing the coordinates and the magnitude of a vector.

2.63.3.5 Code Example:

```
#include "gmclib.h"

tFrac32 f32InMag;
tFrac32 f32InCoord;
tFrac32 f32OutCoord;

void main(void)
{
  f32InMag = FRAC32(0.707106781);
  f32InCoord = FRAC32(0.5);

  // Second coordinate should be FRAC32(0.5)
  f32OutCoord = GMCLIB_Vmag2coord_F32(f32InMag, f32InCoord);

  // Second coordinate should be FRAC32(0.5)
  f32OutCoord = GMCLIB_Vmag2coord(f32InMag, f32InCoord, F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // Second coordinate should be FRAC32(0.5)
  f32OutCoord = GMCLIB_Vmag2coord(f32InMag, f32InCoord);
}
```

2.63.4 Function GMCLIB_Vmag2coord_F16

2.63.4.1 Declaration

```
tFrac16 GMCLIB_Vmag2coord_F16(tFrac16 f16InMag, tFrac16
f16InCoord);
```

2.63.4.2 Arguments

Table 343. GMCLIB_Vmag2coord_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InMag	**input**	Input magnitude of the vector.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 343. GMCLIB_Vmag2coord_F16 arguments...*continued*

Type	Name	Direction	Description
tFrac16	f16InCoord	**input**	Input coordinate of the vector.

2.63.4.3 Return

Second coordinate of the vector.

2.63.4.4 Implementation details

The input values as well as output value is considered as 16-bit fractional values representing the coordinates and the magnitude of a vector.

2.63.4.5 Code Example:

```
#include "gmclib.h"

tFrac16 f16InMag;
tFrac16 f16InCoord;
tFrac16 f16OutCoord;

void main(void)
{
  f16InMag = FRAC16(0.707106781);
  f16InCoord = FRAC16(0.5);

  // Second coordinate should be FRAC16(0.5)
  f16OutCoord = GMCLIB_Vmag2coord_F16(f16InMag, f16InCoord);

  // Second coordinate should be FRAC16(0.5)
  f16OutCoord = GMCLIB_Vmag2coord(f16InMag, f16InCoord, F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // Second coordinate should be FRAC16(0.5)
  f16OutCoord = GMCLIB_Vmag2coord(f16InMag, f16InCoord);
}
```

2.63.5 Function GMCLIB_Vmag2coord_FLT

2.63.5.1 Declaration

```
tFloat GMCLIB_Vmag2coord_FLT(tFloat fltInMag, tFloat fltInCoord);
```

2.63.5.2 Arguments

Table 344. GMCLIB_Vmag2coord_FLT arguments

Type	Name	Direction	Description
Commented parameter f32InMag does not exist in function GMCLIB_Vmag2coord_FLT.	f32InMag	**input**	Input magnitude of the vector.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 344. GMCLIB_Vmag2coord_FLT arguments...*continued*

Type	Name	Direction	Description
Commented parameter f32InCoord does not exist in function GMCLIB_Vmag2coord_FLT.	f32InCoord	**input**	Input coordinate of the vector.

2.63.5.3 Return

Second coordinate of the vector.

2.63.5.4 Implementation details

The input values as well as output value is considered as single precision floating-point values representing the coordinates and the magnitude of a vector.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.63.5.5 Code Example

```
#include "gmclib.h"

tFloat fltInMag;
tFloat fltInCoord;
tFloat fltOutCoord;

void main(void)
{
  fltInMag = 0.707106781f;
  fltInCoord = 0.5f;

  // Second coordinate should be 0.5
  fltOutCoord = GMCLIB_Vmag2coord_FLT(fltInMag, fltInCoord);

  // Second coordinate should be 0.5
  fltOutCoord = GMCLIB_Vmag2coord(fltInMag, fltInCoord, FLT);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // Second coordinate should be 0.5
  fltOutCoord = GMCLIB_Vmag2coord(fltInMag, fltInCoord);
}
```

2.64 Function GMCLIB_VRot

This function returns a vector rotated by a given angle.

2.64.1 Description

This function performs an addition of a given angle to an input vector.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

If the angle is defined as α and the input vector as

$$\vec{u} = (u_1, u_2)$$

Equation GMCLIB_VRot_Eq1

then the output of the function u' is calculated by the following simple equation:

$$\vec{u'} = (u_1 \cdot \cos\alpha - u_2 \cdot \sin\alpha,\ u_1 \cdot \sin\alpha + u_2 \cdot \cos\alpha)$$

Equation GMCLIB_VRot_Eq2

2.64.2 Re-entrancy

The function is re-entrant.

2.64.3 Function GMCLIB_VRot_F32

2.64.3.1 Declaration

```
void GMCLIB_VRot_F32(SWLIBS_2Syst_F32 *const f32OutVec, const
SWLIBS_2Syst_F32 *const f32InVec, tFrac32 f32Angle);
```

2.64.3.2 Arguments

Table 345. GMCLIB_VRot_F32 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F32 *const	f32OutVec	**output**	Rotated vector.
const SWLIBS_2Syst_F32 *const	f32InVec	**input**	Input vector to be rotated.
tFrac32	f32Angle	**input**	Angle in radians from interval [- π, π) normalized between [-1, 1) the input vector shall be rotated by.

2.64.3.3 Implementation details

The first input as well as output value are considered as vectors defined by two 32-bit fractional coordinates stored in SWLIBS_2Syst_F32 structure. Input vector is rotated by angle defined by 32-bit fractional number in radians from interval [- π, π) normalized between [-1, 1).

2.64.3.4 Code Example:

```
#include "gmclib.h"

SWLIBS_2Syst_F32 f32InVec;
tFrac32 f32Angle;
SWLIBS_2Syst_F32 f32Out;

void main(void)
{
  // input vector = (0.5,0.5)
  f32InVec.f32Arg1 = FRAC32(0.5);
  f32InVec.f32Arg2 = FRAC32(0.5);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // angle (-pi) = -1
  f32Angle = FRAC32(-1);

  // Both f32Out.f32Arg1 and f32Out.f32Arg2 should be FRAC32(-0.5) = 0xC0000000
  GMCLIB_VRot_F32(&f32Out, &f32InVec, f32Angle);

  // Both f32Out.f32Arg1 and f32Out.f32Arg2 should be FRAC32(-0.5) = 0xC0000000
  GMCLIB_VRot(&f32Out, &f32InVec, f32Angle, F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // Both f32Out.f32Arg1 and f32Out.f32Arg2 should be FRAC32(-0.5) = 0xC0000000
  GMCLIB_VRot(&f32Out, &f32InVec, f32Angle);
}
```

2.64.4 Function GMCLIB_VRot_F16

2.64.4.1 Declaration

```
void GMCLIB_VRot_F16(SWLIBS_2Syst_F16 *const f16OutVec, const
SWLIBS_2Syst_F16 *const f16InVec, tFrac16 f16Angle);
```

2.64.4.2 Arguments

Table 346. GMCLIB_VRot_F16 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *const	f16OutVec	**output**	Rotated vector.
const SWLIBS_2Syst_F16 *const	f16InVec	**input**	Input vector to be rotated.
tFrac16	f16Angle	**input**	Angle in radians from interval [- π, π) normalized between [-1, 1) the input vector shall be rotated by.

2.64.4.3 Implementation details

The first input as well as output value are considered as vectors defined by two 16-bit fractional coordinates stored in SWLIBS_2Syst_F16 structure. Input vector is rotated by angle defined by 16-bit fractional number in radians from interval [- π, π) normalized between [-1, 1).

2.64.4.4 Code Example:

```
#include "gmclib.h"

SWLIBS_2Syst_F16 f16InVec;
tFrac32 f16Angle;
SWLIBS_2Syst_F16 f16Out;

void main(void)
{
  // input vector = (0.5,0.5)
  f16InVec.f16Arg1 = FRAC16(0.5);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  f16InVec.f16Arg2 = FRAC16(0.5);
  // angle (-pi) = -1
  f16Angle = FRAC16(-1);

  // Both f16Out.f16Arg1 and f16Out.f16Arg2 should be FRAC16(-0.5) = 0xC000
  GMCLIB_VRot_F16(&f16Out, &f16InVec, f16Angle);

  // Both f16Out.f16Arg1 and f16Out.f16Arg2 should be FRAC16(-0.5) = 0xC000
  GMCLIB_VRot(&f16Out, &f16InVec, f16Angle, F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // Both f16Out.f16Arg1 and f16Out.f16Arg2 should be FRAC16(-0.5) = 0xC000
  GMCLIB_VRot(&f16Out, &f16InVec, f16Angle);
}
```

2.64.5 Function GMCLIB_VRot_FLT

2.64.5.1 Declaration

```
void GMCLIB_VRot_FLT(SWLIBS_2Syst_FLT *const fltOutVec, const
SWLIBS_2Syst_FLT *const fltInVec, tFloat fltAngle);
```

2.64.5.2 Arguments

Table 347. GMCLIB_VRot_FLT arguments

Type	Name	Direction	Description
SWLIBS_2Syst_FLT *const	fltOutVec	**output**	Rotated vector.
const SWLIBS_2Syst_FLT *const	fltInVec	**input**	Input vector to be rotated.
tFloat	fltAngle	**input**	Angle the input vector shall be rotated by.

2.64.5.3 Implementation details

The first input as well as output value are considered as vectors defined by two single precision floating-point coordinates stored in SWLIBS_2Syst_FLT structure. Input vector is rotated by angle defined by single precision floating-point number in radians from interval [- π, π).

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.64.5.4 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_FLT fltInVec;
tFrac32 fltAngle;
SWLIBS_2Syst_FLT fltOut;

void main(void)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
{
  // input vector = (0.5,0.5)
  fltInVec.fltArg1 = 0.5;
  fltInVec.fltArg2 = 0.5;
  // angle = pi
  fltAngle = FLOAT_PI;

  // Both fltOut.fltArg1 and fltOut.fltArg2 should be -0.5
  GMCLIB_VRot_FLT(&fltOut, &fltInVec, fltAngle);

  // Both fltOut.fltArg1 and fltOut.fltArg2 should be -0.5
  GMCLIB_VRot(&fltOut, &fltInVec, fltAngle, FLT);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // Both fltOut.fltArg1 and fltOut.fltArg2 should be -0.5
  GMCLIB_VRot(&fltOut, &fltInVec, fltAngle);
}
```

2.65 Function GMCLIB_VUnit

This function returns an unit vector created by scaling input vector.

2.65.1 Description

This function performs a vector normalization so the output vector has the same angle as input vector and the magnitude equal to 1.

If the input vector is defined as

$$\vec{u} = (u_1, u_2)$$

Equation GMCLIB_VUnit_Eq1

then the output unit vector u' is calculated by the following simple equation:

$$\vec{u'} = \left(\frac{u_1}{|u|}, \frac{u_2}{|u|}\right)$$

Equation GMCLIB_VUnit_Eq2

where |u| is magnitude of the input vector.

2.65.2 Re-entrancy

The function is re-entrant.

2.65.3 Function GMCLIB_VUnit_F32

2.65.3.1 Declaration

```
void GMCLIB_VUnit_F32(SWLIBS_2Syst_F32 *const f32OutVec, const
SWLIBS_2Syst_F32 *const f32InVec);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.65.3.2 Arguments

Table 348. GMCLIB_VUnit_F32 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F32 *const	f32OutVec	**output**	Output unit vector.
const SWLIBS_2Syst_F32 *const	f32InVec	**input**	Input vector to be scaled to unit vector.

2.65.3.3 Implementation details

The input as well as output are considered as vectors defined by two 32-bit fractional coordinates stored in SWLIBS_2Syst_F32 structure.

2.65.3.4 Code Example:

```
#include "gmclib.h"

SWLIBS_2Syst_F32 f32In;
SWLIBS_2Syst_F32 f32Out;

void main(void)
{
  // input vector = (0.5,0.5)
  f32InVec.f32Arg1 = FRAC32(0.5);
  f32InVec.f32Arg2 = FRAC32(0.5);

  // Both f32Out.f32Arg1 and f32Out.f32Arg2 should be FRAC32(0.707106781186548)
  GMCLIB_VUnit_F32(&f32Out, &f32In);

  // Both f32Out.f32Arg1 and f32Out.f32Arg2 should be FRAC32(0.707106781186548)
  GMCLIB_VUnit(&f32Out, &f32In, F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // Both f32Out.f32Arg1 and f32Out.f32Arg2 should be FRAC32(0.707106781186548)
  GMCLIB_VUnit(&f32Out, &f32In);
}
```

2.65.4 Function GMCLIB_VUnit_F16

2.65.4.1 Declaration

```
void GMCLIB_VUnit_F16(SWLIBS_2Syst_F16 *const f16OutVec, const
SWLIBS_2Syst_F16 *const f16InVec);
```

2.65.4.2 Arguments

Table 349. GMCLIB_VUnit_F16 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *const	f16OutVec	**output**	Output unit vector.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 349. GMCLIB_VUnit_F16 arguments...*continued*

Type	Name	Direction	Description
const SWLIBS_2Syst_F16 *const	f16InVec	**input**	Input vector to be scaled to unit vector.

2.65.4.3 Implementation details

The input as well as output are considered as vectors defined by two 16-bit fractional coordinates stored in SWLIBS_2Syst_F16 structure.

2.65.4.4 Code Example:

```
#include "gmclib.h"

SWLIBS_2Syst_F16 f16In;
SWLIBS_2Syst_F16 f16Out;

void main(void)
{
  // input vector = (0.5,0.5)
  f16InVec.f16Arg1 = FRAC16(0.5);
  f16InVec.f16Arg2 = FRAC16(0.5);

  // Both f16Out.f16Arg1 and f16Out.f16Arg2 should be FRAC16(0.707106781186548)
  GMCLIB_VUnit_F16(&f16Out, &f16In);

  // Both f16Out.f16Arg1 and f16Out.f16Arg2 should be FRAC16(0.707106781186548)
  GMCLIB_VUnit(&f16Out, &f16In, F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // Both f16Out.f16Arg1 and f16Out.f16Arg2 should be FRAC16(0.707106781186548)
  GMCLIB_VUnit(&f16Out, &f16In);
}
```

2.65.5 Function GMCLIB_VUnit_FLT

2.65.5.1 Declaration

```
void GMCLIB_VUnit_FLT(SWLIBS_2Syst_FLT *const fltOutVec, const
SWLIBS_2Syst_FLT *const fltInVec);
```

2.65.5.2 Arguments

Table 350. GMCLIB_VUnit_FLT arguments

Type	Name	Direction	Description
SWLIBS_2Syst_FLT *const	fltOutVec	**output**	Output unit vector.
const SWLIBS_2Syst_FLT *const	fltInVec	**input**	Input vector to be scaled to unit vector.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.65.5.3 Implementation details

The input as well as output are considered as vectors defined by two single precision floating-point coordinates stored in SWLIBS_2Syst_FLT structure.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

2.65.5.4 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_FLT fltIn;
SWLIBS_2Syst_FLT fltOut;

void main(void)
{
  // input vector = (0.5,0.5)
  fltInVec.fltArg1 = 0.5;
  fltInVec.fltArg2 = 0.5;

  // Both fltOut.fltArg1 and fltOut.fltArg2 should be 0.707106781186548
  GMCLIB_VUnit_FLT(&fltOut, &fltIn);

  // Both fltOut.fltArg1 and fltOut.fltArg2 should be 0.707106781186548
  GMCLIB_VUnit(&fltOut, &fltIn, FLT);

  // ##############################################################
  // Available only if single precision floating-point implementation selected
  // as default
  // ##############################################################

  // Both fltOut.fltArg1 and fltOut.fltArg2 should be 0.707106781186548
  GMCLIB_VUnit(&fltOut, &fltIn);
}
```

2.66 Function MLIB_Abs

This function returns absolute value of input parameter.

2.66.1 Description

This inline function returns the absolute value of input parameter.

2.66.2 Re-entrancy

The function is re-entrant.

2.66.3 Function MLIB_Abs_F32

2.66.3.1 Declaration

```
INLINE tFrac32 MLIB_Abs_F32(register tFrac32 f32In);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.66.3.2 Arguments

Table 351. MLIB_Abs_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value.

2.66.3.3 Return

Absolute value of input parameter.

2.66.3.4 Implementation details

The input value as well as output value is considered as 32-bit fractional data type. The output saturation is not implemented in this function, thus in case the absolute value of input parameter is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} f32In & if \quad f32In \geq 0 \\ -f32In & if \quad f32In < 0 \end{cases}$$

Equation MLIB_Abs_F32_Eq1

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

2.66.3.5 Code Example

```
#include "mlib.h"

tFrac32 f32In;
tFrac32 f32Out;

void main(void)
{
    // input value = -0.25
    f32In = FRAC32(-0.25);

    // output should be FRAC32(0.25)
    f32Out = MLIB_Abs_F32(f32In);

    // output should be FRAC32(0.25)
    f32Out = MLIB_Abs(f32In, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be FRAC32(0.25)
    f32Out = MLIB_Abs(f32In);
}
```

2.66.4 Function MLIB_Abs_F16

2.66.4.1 Declaration

```
INLINE tFrac16 MLIB_Abs_F16(register tFrac16 f16In);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.66.4.2 Arguments

Table 352. MLIB_Abs_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	Input value.

2.66.4.3 Return

Absolute value of input parameter.

2.66.4.4 Implementation details

The input value as well as output value is considered as 16-bit fractional data type. The output saturation is not implemented in this function, thus in case the absolute value of input parameter is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} f16In & if \quad f16In \geq 0 \\ -f16In & if \quad f16In < 0 \end{cases}$$

Equation MLIB_Abs_F16_Eq1

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

2.66.4.5 Code Example

```
#include "mlib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
{
    // input value = -0.25
    f16In = FRAC16(-0.25);

    // output should be FRAC16(0.25)
    f16Out = MLIB_Abs_F16(f16In);

    // output should be FRAC16(0.25)
    f16Out = MLIB_Abs(f16In, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be FRAC16(0.25)
    f16Out = MLIB_Abs(f16In);
}
```

2.66.5 Function MLIB_Abs_FLT

2.66.5.1 Declaration

```
INLINE tFloat MLIB_Abs_FLT(register tFloat fltIn);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.66.5.2 Arguments

Table 353. MLIB_Abs_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn	**input**	Input value.

2.66.5.3 Return

Absolute value of input parameter.

2.66.5.4 Implementation details

The input value as well as output value is considered as single precision floating point data type.

The output of the function is defined by the following simple equation:

$$fltOut = \begin{cases} fltIn & if \quad fltIn \geq 0 \\ -fltIn & if \quad fltIn < 0 \end{cases}$$

Equation MLIB_Abs_FLT_Eq1

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

This function is implemented as inline, and thus is not ANSI-C compliant.

2.66.5.5 Code Example

```
#include "mlib.h"

tFloat fltIn;
tFloat fltOut;

void main(void)
{
    // input value = -0.25
    fltIn = (tFloat)-0.25;

    // output should be 0.25
    fltOut = MLIB_Abs_FLT(fltIn);

    // output should be 0.25
    fltOut = MLIB_Abs(fltIn, FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be 0.25
    fltOut = MLIB_Abs(fltIn);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.67 Function MLIB_AbsSat

This function returns absolute value of input parameter and saturate if necessary.

2.67.1 Description

This inline function returns the absolute value of input parameter and saturates if necessary.

2.67.2 Re-entrancy

The function is re-entrant.

2.67.3 Function MLIB_AbsSat_F32

2.67.3.1 Declaration

```
INLINE tFrac32 MLIB_AbsSat_F32(register tFrac32 f32In);
```

2.67.3.2 Arguments

Table 354. MLIB_AbsSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value.

2.67.3.3 Return

Absolute value of input parameter, saturated if necessary.

2.67.3.4 Implementation details

The input values as well as output value is considered as 32-bit fractional data type.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32_MIN & if \quad |f32In| < FRAC32_MIN \\ |f32In| & if \quad FRAC32_MIN \le |f32In| \le FRAC32_MAX \\ FRAC32_MAX & if \quad |f32In| > FRAC32_MAX \end{cases}$$

Equation MLIB_AbsSat_F32_Eq1

Note: *This function is implemented as inline assembly, and thus is not ANSI-C compliant.*

2.67.3.5 Code Example

```
#include "mlib.h"

tFrac32 f32In;
tFrac32 f32Out;

void main(void)
{
    // input value = -0.25
    f32In = FRAC32(-0.25);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // output should be FRAC32(0.25)
    f32Out = MLIB_AbsSat_F32(f32In);

    // output should be FRAC32(0.25)
    f32Out = MLIB_AbsSat(f32In, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be FRAC32(0.25)
    f32Out = MLIB_AbsSat(f32In);
}
```

2.67.4 Function MLIB_AbsSat_F16

2.67.4.1 Declaration

```
INLINE tFrac16 MLIB_AbsSat_F16(register tFrac16 f16In);
```

2.67.4.2 Arguments

Table 355. MLIB_AbsSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	Input value.

2.67.4.3 Return

Absolute value of input parameter, saturated if necessary.

2.67.4.4 Implementation details

The input values as well as output value is considered as 16-bit fractional data type.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16_MIN & if \quad |f16In| < FRAC16_MIN \\ |f16In| & if \quad FRAC16_MIN \le |f16In| \le FRAC16_MAX \\ FRAC16_MAX & if \quad |f16In| > FRAC16_MAX \end{cases}$$

Equation MLIB_AbsSat_F16_Eq1

Note: *This function is implemented as inline assembly, and thus is not ANSI-C compliant.*

2.67.4.5 Code Example

```
#include "mlib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
{
    // input value = -0.25
    f16In = FRAC16(-0.25);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // output should be FRAC16(0.25)
    f16Out = MLIB_AbsSat_F16(f16In);

    // output should be FRAC16(0.25)
    f16Out = MLIB_AbsSat(f16In, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be FRAC16(0.25)
    f16Out = MLIB_AbsSat(f16In);
}
```

2.68 Function MLIB_Add

This function returns sum of two input parameters.

2.68.1 Description

This inline function returns the sum of two input values.

2.68.2 Re-entrancy

The function is re-entrant.

2.68.3 Function MLIB_Add_F32

2.68.3.1 Declaration

```
INLINE tFrac32 MLIB_Add_F32(register tFrac32 f32In1, register
tFrac32 f32In2);
```

2.68.3.2 Arguments

Table 356. MLIB_Add_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	First value to be add.
register tFrac32	f32In2	**input**	Second value to be add.

2.68.3.3 Return

Sum of two input values.

2.68.3.4 Implementation details

The input values as well as output value is considered as 32-bit fractional data type. The output saturation is not implemented in this function, thus in case the sum of input values is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f32Out = f32In1 + f32In2$$

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Equation MLIB_Add_F32_Eq1

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.68.3.5 Code Example:

```
#include "mlib.h"

tFrac32 f32In1, f32In2;
tFrac32 f32Out;

void main(void)
{
  // input value 1 = 0.25
  f32In1 = FRAC32(0.25);
  // input value 2 = 0.25
  f32In2 = FRAC32(0.25);

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_Add_F32(f32In1, f32In2);

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_Add(f32In1, f32In2, F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_Add(f32In1, f32In2);
}
```

2.68.4 Function MLIB_Add_F16

2.68.4.1 Declaration

```
INLINE tFrac16 MLIB_Add_F16(register tFrac16 f16In1, register
tFrac16 f16In2);
```

2.68.4.2 Arguments

Table 357. MLIB_Add_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	First value to be add.
register tFrac16	f16In2	**input**	Second value to be add.

2.68.4.3 Return

Sum of two input values.

2.68.4.4 Implementation details

The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the sum of input values is outside the [-1, 1) interval, the output value will overflow.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The output of the function is defined by the following simple equation:

$$f16Out = f16In1 + f16In2$$

Equation MLIB_Add_F16_Eq1

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.68.4.5 Code Example

```
#include "mlib.h"

tFrac16 f16In1, f16In2;
tFrac16 f16Out;

void main(void)
{
  // input value 1 = 0.25
  f16In1 = FRAC16(0.25);
  // input value 2 = 0.25
  f16In2 = FRAC16(0.25);

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_Add_F16(f16In1, f16In2);

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_Add(f16In1, f16In2, F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_Add(f16In1, f16In2);
}
```

2.68.5 Function MLIB_Add_FLT

2.68.5.1 Declaration

```
INLINE tFloat MLIB_Add_FLT(register tFloat fltIn1, register
tFloat fltIn2);
```

2.68.5.2 Arguments

Table 358. MLIB_Add_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn1	**input**	First value to be add.
register tFloat	fltIn2	**input**	Second value to be add.

2.68.5.3 Return

Sum of two input values.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

This function is implemented as inline and thus is not ANSI-C compliant.

2.68.5.4 Code Example

```
#include "mlib.h"

tFloat fltIn1, fltIn2;
tFloat fltOut;

void main(void)
{
  // input value 1 = 0.25
  fltIn1 = (tFloat)0.25;
  // input value 2 = 0.25
  fltIn2 = (tFloat)0.25;

  // output should be 0.5
  fltOut = MLIB_Add_FLT(fltIn1, fltIn2);

  // output should be 0.5
  fltOut = MLIB_Add(fltIn1, fltIn2, FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output should be 0.5
  fltOut = MLIB_Add(fltIn1, fltIn2);
}
```

2.69 Function MLIB_AddSat

This function returns sum of two input parameters and saturate if necessary.

2.69.1 Description

This inline function returns the sum of two input values and saturates the result if necessary.

2.69.2 Re-entrancy

The function is re-entrant.

2.69.3 Function MLIB_AddSat_F32

2.69.3.1 Declaration

```
INLINE tFrac32 MLIB_AddSat_F32(register tFrac32 f32In1, register
tFrac32 f32In2);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.69.3.2 Arguments

Table 359. MLIB_AddSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	First value to be add.
register tFrac32	f32In2	**input**	Second value to be add.

2.69.3.3 Return

Sum of two input values, saturated if necessary.

2.69.3.4 Implementation details

The input values as well as output value is considered as 32-bit fractional data type.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32_MIN & if \quad (f32In1 + f32In2) < FRAC32_MIN \\ f32In1 + f32In2 & if \quad FRAC32_MIN \le (f32In1 + f32In2) \le FRAC32_MAX \\ FRAC32_MAX & if \quad (f32In1 + f32In2) > FRAC32_MAX \end{cases}$$

Equation MLIB_AddSat_F32_Eq1

Note: *This function is implemented as inline assembly, and thus is not ANSI-C compliant.*

2.69.3.5 Code Example

```
#include "mlib.h"

tFrac32 f32In1, f32In2;
tFrac32 f32Out;

void main(void)
{
  // input value 1 = 0.25
  f32In1 = FRAC32(0.25);
  // input value 2 = 0.25
  f32In2 = FRAC32(0.25);

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_AddSat_F32(f32In1, f32In2);

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_AddSat(f32In1, f32In2, F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_AddSat(f32In1, f32In2);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.69.4 Function MLIB_AddSat_F16

2.69.4.1 Declaration

```
INLINE tFrac16 MLIB_AddSat_F16(register tFrac16 f16In1, register
tFrac16 f16In2);
```

2.69.4.2 Arguments

Table 360. MLIB_AddSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	First value to be add.
register tFrac16	f16In2	**input**	Second value to be add.

2.69.4.3 Return

Sum of two input values, saturated if necessary.

2.69.4.4 Implementation details

The input values as well as output value is considered as 16-bit fractional data type.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16_MIN & if \quad (f16In1 + f16In2) < FRAC16_MIN \\ f16In1 + f16In2 & if \quad FRAC16_MIN \le (f16In1 + f16In2) \le FRAC16_MAX \\ FRAC16_MAX & if \quad (f16In1 + f16In2) > FRAC16_MAX \end{cases}$$

Equation MLIB_AddSat_F16_Eq1

Note: *This function is implemented as inline assembly, and thus is not ANSI-C compliant.*

2.69.4.5 Code Example

```
#include "mlib.h"

tFrac16 f16In1, f16In2;
tFrac16 f16Out;

void main(void)
{
  // input value 1 = 0.25
  f16In1 = FRAC16(0.25);
  // input value 2 = 0.25
  f16In2 = FRAC16(0.25);

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_AddSat_F16(f16In1, f16In2);

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_AddSat(f16In1, f16In2, F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_AddSat(f16In1, f16In2);
}
```

2.70 Function MLIB_Convert

This function converts the input value to different representation with scale.

2.70.1 Description

This inline function converts the input value to a different data type. The second input argument represents the scale factor.

2.70.2 Re-entrancy

The function is re-entrant.

2.70.3 Function MLIB_Convert_F32F16

2.70.3.1 Declaration

```
INLINE tFrac32 MLIB_Convert_F32F16(register tFrac16 f16In1,
register tFrac16 f16In2);
```

2.70.3.2 Arguments

Table 361. MLIB_Convert_F32F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Input value in 16-bit fractional format to be converted.
register tFrac16	f16In2	**input**	Scale factor in 16-bit fractional format.

2.70.3.3 Return

Converted input value in 32-bit fractional format.

2.70.3.4 Implementation details

The input value is considered as 16-bit fractional data type and output value is considered as 32-bit fractional data type. The second argument is considered as 16-bit fractional data type. The sign of the second value represents the scale mechanism. In case the second value is positive the first input value is multiplied with the second one and converted to the output format. In case the second value is negative, the first input value is divided by absolute value of second input value and converted to the output format. The output saturation is not implemented in this function, thus in case the input value is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} (tFrac32)\frac{f16In1}{|f16In2|} & if \quad f16In2 < 0 \\ (tFrac32)(f16In1 \cdot f16In2) & if \quad f16In2 \geq 0 \end{cases}$$

Equation MLIB_Convert_F32F16_Eq1

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.70.3.5 Code Example

```
#include "mlib.h"

tFrac16 f16In1, f16In2;
tFrac32 f32Out;

void main(void)
{
    // input value = 0.25 = 0x2000
    f16In1 = FRAC16(0.25);

    // scale value = 0.5 = 0x4000
    f16In2 = FRAC16(0.5);

    // output should be FRAC32(0.125) = 0x10000000
    f32Out = MLIB_Convert_F32F16(f16In1, f16In2);

    // output should be FRAC32(0.125) = 0x10000000
    f32Out = MLIB_Convert(f16In1, f16In2, F32F16);

    // scale value = -0.5 = 0xC000
    f16In2 = FRAC16(-0.5);

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_Convert_F32F16(f16In1, f16In2);

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_Convert(f16In1, f16In2, F32F16);
}
```

2.70.4 Function MLIB_Convert_F32FLT

2.70.4.1 Declaration

```
INLINE tFrac32 MLIB_Convert_F32FLT(register tFloat fltIn1,
register tFloat fltIn2);
```

2.70.4.2 Arguments

Table 362. MLIB_Convert_F32FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn1	**input**	Input value in single precision floating point format to be converted.
register tFloat	fltIn2	**input**	Scale factor in single precision floating point format.

2.70.4.3 Return

Converted input value in 32-bit fractional format.

2.70.4.4 Implementation details

The input value is considered as single precision floating point data type and output value is considered as 32-bit fractional data type. The second argument is considered as single

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

precision floating point data type. The output saturation is implemented in this function, thus in case the input value is outside the [-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32_MIN & if \quad (fltIn1 \cdot fltIn2) < FRAC32_MIN \\ fltIn1 \cdot fltIn2 \cdot ((tFloat)INT32_MAX + 1) & if \quad FRAC32_MIN \le (fltIn1 \cdot fltIn2) \le FRAC32_MAX \\ FRAC32_MAX & if \quad (fltIn1 \cdot fltIn2) > FRAC32_MAX \end{cases}$$

Equation MLIB_Convert_F32FLT_Eq1

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

This function is implemented as inline and thus is not ANSI-C compliant.

2.70.4.5 Code Example

```
#include "mlib.h"

tFloat fltIn1, fltIn2;
tFrac32 f32Out;

void main(void)
{
    // input value = 0.25
    fltIn1 = (tFloat)0.25;

    // scale value = 0.5
    fltIn2 = (tFloat)0.5;

    // output should be FRAC32(0.125) = 0x10000000
    f32Out = MLIB_Convert_F32FLT(fltIn1, fltIn2);

    // output should be FRAC32(0.125) = 0x10000000
    f32Out = MLIB_Convert(fltIn1, fltIn2, F32FLT);

    // scale value = 2
    fltIn2 = (tFloat)2;

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_Convert_F32FLT(fltIn1, fltIn2);

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_Convert(fltIn1, fltIn2, F32FLT);
}
```

2.70.5 Function MLIB_Convert_F16F32

2.70.5.1 Declaration

```
INLINE tFrac16 MLIB_Convert_F16F32(register tFrac32 f32In1,
register tFrac32 f32In2);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.70.5.2 Arguments

Table 363. MLIB_Convert_F16F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Input value in 32-bit fractional format to be converted.
register tFrac32	f32In2	**input**	Scale factor in 32-bit fractional format.

2.70.5.3 Return

Converted input value in 16-bit fractional format.

2.70.5.4 Implementation details

The input value is considered as 32-bit fractional data type and output value is considered as 16-bit fractional data type. The second value is considered as 32-bit fractional data type. The sign of the second value represents the scale mechanism. In case the second value is positive the first input value is multiplied with the second one and converted to the output format. In case the second value is negative, the first input value is divided by absolute value of second input value and converted to the output format. The output saturation is not implemented in this function, thus in case the input value is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} (tFrac16)\frac{f32In1}{|f32In2|} & if \quad f32In2 < 0 \\ (tFrac16)(f32In1 \cdot f32In2) & if \quad f32In2 \geq 0 \end{cases}$$

Equation MLIB_Convert_F16F32_Eq1

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.70.5.5 Code Example

```
#include "mlib.h"

tFrac32 f32In1, f32In2;
tFrac16 f16Out;

void main(void)
{
    // input value = 0.25 = 0x20000000
    f32In1 = FRAC32(0.25);

    // scale value = 0.5 = 0x40000000
    f32In2 = FRAC32(0.5);

    // output should be FRAC16(0.125) = 0x1000
    f16Out = MLIB_Convert_F16F32(f32In1, f32In2);

    // output should be FRAC16(0.125) = 0x1000
    f16Out = MLIB_Convert(f32In1, f32In2, F16F32);

    // scale value = -0.5 = 0xC0000000
    f32In2 = FRAC32(-0.5);

    // output should be FRAC16(0.5) = 0x4000
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    f16Out = MLIB_Convert_F16F32(f32In1, f32In2);

    // output should be FRAC16(0.5) = 0x4000
    f16Out = MLIB_Convert(f32In1, f32In2, F16F32);
}
```

2.70.6 Function MLIB_Convert_F16FLT

2.70.6.1 Declaration

```
INLINE tFrac16 MLIB_Convert_F16FLT(register tFloat fltIn1,
register tFloat fltIn2);
```

2.70.6.2 Arguments

Table 364. MLIB_Convert_F16FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn1	**input**	Input value in single precision floating point format to be converted.
register tFloat	fltIn2	**input**	Scale factor in single precision floating point format.

2.70.6.3 Return

Converted input value in 16-bit fractional format.

2.70.6.4 Implementation details

The input value is considered as single precision floating point data type and output value is considered as 16-bit fractional data type. The second value is considered as single precision floating point data type. The output saturation is implemented in this function, thus in case the input value is outside the [-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16_MIN & if \quad (fltIn1 \cdot fltIn2) < FRAC16_MIN \\ fltIn1 \cdot fltIn2 \cdot ((tFloat)INT16_MAX + 1) & if \quad FRAC16_MIN \le (fltIn1 \cdot fltIn2) \le FRAC16_MAX \\ FRAC16_MAX & if \quad (fltIn1 \cdot fltIn2) > FRAC16_MAX \end{cases}$$

Equation MLIB_Convert_F16FLT_Eq1

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

This function is implemented as inline and thus is not ANSI-C compliant.

2.70.6.5 Code Example

```
#include "mlib.h"

tFloat fltIn1, fltIn2;
tFrac16 f16Out;

void main(void)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
{
    // input value = 0.25
    fltIn1 = (tFloat)0.25;

    // scale value = 0.5
    fltIn2 = (tFloat)0.5;

    // output should be FRAC16(0.125) = 0x1000
    f16Out = MLIB_Convert_F16FLT(fltIn1, fltIn2);

    // output should be FRAC16(0.125) = 0x1000
    f16Out = MLIB_Convert(fltIn1, fltIn2, F16FLT);

    // scale value = 2
    fltIn2 = (tFloat)2;

    // output should be FRAC16(0.5) = 0x4000
    f16Out = MLIB_Convert_F16FLT(fltIn1, fltIn2);

    // output should be FRAC16(0.5) = 0x4000
    f16Out = MLIB_Convert(fltIn1, fltIn2, F16FLT);
}
```

2.70.7 Function MLIB_Convert_FLTF16

2.70.7.1 Declaration

```
INLINE tFloat MLIB_Convert_FLTF16(register tFrac16 f16In1,
register tFrac16 f16In2);
```

2.70.7.2 Arguments

Table 365. MLIB_Convert_FLTF16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Input value in 16-bit fractional format to be converted.
register tFrac16	f16In2	**input**	Scale factor in 16-bit fractional format.

2.70.7.3 Return

Converted input value in single precision floating point format.

2.70.7.4 Implementation details

The input value is considered as 16-bit fractional data type and output value is considered as single precision floating point data type. The second value is considered as 16-bit fractional data type. The sign of the second value represents the scale mechanism. In case the second value is positive the first input value is multiplied with the second one and converted to the output format. In case the second value is negative, the first input value is divided by absolute value of second input value and converted to the output format. The output saturation is not implemented in this function.

The output of the function is defined by the following simple equation:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$fltOut = \begin{cases} \frac{f16In1 \cdot f16In2}{(tFloat)INT16_MAX+1} & if \quad f16In2 \geq 0 \\ (tFloat)1 & if \quad f16In2 < 0 \wedge f16In1 \geq -f16In2 \\ (tFloat)-1 & if \quad f16In2 < 0 \wedge f16In1 \leq f16In2 \\ \frac{(tFloat)f16In1}{(tFloat)|f16In2|} & otherwise \end{cases}$$

Equation MLIB_Convert_FLTF16_Eq1

Note: *The function may raise floating-point exceptions (invalid operation, division by zero, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

This function is implemented as inline and thus is not ANSI-C compliant.

2.70.7.5 Code Example

```
#include "mlib.h"

tF16 f16In1, f16In2;
tFloat fltOut;

void main(void)
{
    // input value = 0.25 = 0x2000
    f16In1 = FRAC16(0.25);

    // scale value = 0.5 = 0x4000
    f16In2 = FRAC16(0.5);

    // output should be 0.125
    fltOut = MLIB_Convert_FLTF16(f16In1, f16In2);

    // output should be 0.125
    fltOut = MLIB_Convert(f16In1, f16In2, FLTF16);

    // scale value = -0.5 = 0xC000
    f16In2 = FRAC16(-0.5);

    // output should be 0.5
    fltOut = MLIB_Convert_FLTF16(f16In1, f16In2);

    // output should be 0.5
    fltOut = MLIB_Convert(f16In1, f16In2, FLTF16);
}
```

2.70.8 Function MLIB_Convert_FLTF32

2.70.8.1 Declaration

```
INLINE tFloat MLIB_Convert_FLTF32(register tFrac32 f32In1,
register tFrac32 f32In2);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.70.8.2 Arguments

Table 366. MLIB_Convert_FLTF32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Input value in 32-bit fractional format to be converted.
register tFrac32	f32In2	**input**	Scale factor in 32-bit fractional format.

2.70.8.3 Return

Converted input value in single precision floating point format.

2.70.8.4 Implementation details

The input value is considered as 32-bit fractional data type and output value is considered as single precision floating point data type. The second value is considered as 32-bit fractional data type. The sign of the second value represents the scale mechanism. In case the second value is positive the first input value is multiplied with the second one and converted to the output format. In case the second value is negative, the first input value is divided by absolute value of second input value and converted to the output format. The output saturation is not implemented in this function.

The output of the function is defined by the following simple equation:

$$fltOut = \begin{cases} \frac{f32In1 \cdot f32In2}{(tFloat)INT32_MAX+1} & if \quad f32In2 \geq 0 \\ (tFloat)1 & if \quad f32In2 < 0 \wedge f32In1 \geq -f32In2 \\ (tFloat)-1 & if \quad f32In2 < 0 \wedge f32In1 \leq f32In2 \\ \frac{(tFloat)f32In1}{(tFloat)|f32In2|} & otherwise \end{cases}$$

Equation MLIB_Convert_FLTF32_Eq1

Note: *The function may raise floating-point exceptions (invalid operation, division by zero, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

This function is implemented as inline and thus is not ANSI-C compliant.

2.70.8.5 Code Example

```
#include "mlib.h"

tF32 f32In1, f32In2;
tFloat fltOut;

void main(void)
{
    // input value = 0.25 = 0x2000
    f32In1 = FRAC32(0.25);

    // scale value = 0.5 = 0x4000
    f32In2 = FRAC32(0.5);

    // output should be 0.125
    fltOut = MLIB_Convert_FLTF32(f32In1, f32In2);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    // output should be 0.125
    fltOut = MLIB_Convert(f32In1, f32In2, FLTF32);

    // scale value = -0.5 = 0xC000
    f32In2 = FRAC32(-0.5);

    // output should be 0.5
    fltOut = MLIB_Convert_FLTF32(f32In1, f32In2);

    // output should be 0.5
    fltOut = MLIB_Convert(f32In1, f32In2, FLTF32);
}
```

2.71 Function MLIB_ConvertPU

This function converts the input value to a different data type.

2.71.1 Description

This inline function converts the input value to a different data type.

2.71.2 Re-entrancy

The function is re-entrant.

2.71.3 Function MLIB_ConvertPU_F32F16

2.71.3.1 Declaration

```
INLINE tFrac32 MLIB_ConvertPU_F32F16(register tFrac16 f16In);
```

2.71.3.2 Arguments

Table 367. MLIB_ConvertPU_F32F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	Input value in 16-bit fractional format to be converted.

2.71.3.3 Return

Converted input value in 32-bit fractional format.

2.71.3.4 Implementation details

The input value is considered as 16-bit fractional data type and output value is considered as 32-bit fractional data type.

The output of the function is defined by the following simple equation:

$$f32Out = (tFrac32)f16In$$

Equation MLIB_ConvertPU_F32F16_Eq1

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.71.3.5 Code Example

```
#include "mlib.h"

tFrac16 f16In;
tFrac32 f32Out;

void main(void)
{
    // input value = 0.25 = 0x2000
    f16In = FRAC16(0.25);

    // output should be FRAC32(0.25) = 0x20000000
    f32Out = MLIB_ConvertPU_F32F16(f16In);

    // output should be FRAC32(0.25) = 0x20000000
    f32Out = MLIB_ConvertPU(f16In, F32F16);
}
```

2.71.4 Function MLIB_ConvertPU_F32FLT

2.71.4.1 Declaration

```
INLINE tFrac32 MLIB_ConvertPU_F32FLT(register tFloat fltIn);
```

2.71.4.2 Arguments

Table 368. MLIB_ConvertPU_F32FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn	**input**	Input value in single precision floating point format to be converted.

2.71.4.3 Return

Converted input value in 32-bit fractional format.

2.71.4.4 Implementation details

The input value is considered as single precision floating point data type and output value is considered as 32-bit fractional data type. The output saturation is implemented in this function, thus in case the input value is outside the [-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32_MIN & if \quad fltIn < FRAC32_MIN \\ \frac{fltIn}{(tFloat)INT32_MAX+1} & if \quad FRAC32_MIN \le fltIn \le FRAC32_MAX \\ FRAC32_MAX & if \quad fltIn > FRAC32_MAX \end{cases}$$

Equation MLIB_ConvertPU_F32FLT_Eq1

Note: *The function may raise floating-point exceptions (invalid operation, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

This function is implemented as inline, and thus is not ANSI-C compliant.

2.71.4.5 Code Example

```
#include "mlib.h"

tFloat fltIn;
tFrac32 f32Out;

void main(void)
{
    // input value = 0.25
    fltIn = (tFloat)0.25;

    // output should be FRAC32(0.25) = 0x20000000
    f32Out = MLIB_ConvertPU_F32FLT(fltIn);

    // output should be FRAC32(0.25) = 0x20000000
    f32Out = MLIB_ConvertPU(fltIn, F32FLT);
}
```

2.71.5 Function MLIB_ConvertPU_F16F32

2.71.5.1 Declaration

```
INLINE tFrac16 MLIB_ConvertPU_F16F32(register tFrac32 f32In);
```

2.71.5.2 Arguments

Table 369. MLIB_ConvertPU_F16F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value in 32-bit fractional format to be converted.

2.71.5.3 Return

Converted input value in 16-bit fractional format.

2.71.5.4 Implementation details

The input value is considered as 32-bit fractional data type and output value is considered as 16-bit fractional data type.

The output of the function is defined by the following simple equation:

$$f16Out = (tFrac16)f32In$$

Equation MLIB_ConvertPU_F16F32_Eq1

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

2.71.5.5 Code Example

```
#include "mlib.h"

tFrac32 f32In;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
tFrac16 f16Out;

void main(void)
{
    // input value = 0.25 = 0x2000 0000
    f32In = FRAC32(0.25);

    // output should be FRAC16(0.25) = 0x2000
    f16Out = MLIB_ConvertPU_F16F32(f32In);

    // output should be FRAC16(0.25) = 0x2000
    f16Out = MLIB_ConvertPU(f32In, F16F32);
}
```

2.71.6 Function MLIB_ConvertPU_F16FLT

2.71.6.1 Declaration

```
INLINE tFrac16 MLIB_ConvertPU_F16FLT(register tFloat fltIn);
```

2.71.6.2 Arguments

Table 370. MLIB_ConvertPU_F16FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn	**input**	Input value in single precision floating point format to be converted.

2.71.6.3 Return

Converted input value in 16-bit fractional format.

2.71.6.4 Implementation details

The input value is considered as single precision floating point data type and output value is considered as 16-bit fractional data type. The output saturation is implemented in this function, thus in case the input value is outside the [-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16_MIN & if \quad fltIn < FRAC16_MIN \\ \frac{fltIn}{(tFloat)INT16_MAX+1} & if \quad FRAC16_MIN \le fltIn \le FRAC16_MAX \\ FRAC16_MAX & if \quad fltIn > FRAC16_MAX \end{cases}$$

Equation MLIB_ConvertPU_F16FLT_Eq1

Note: *The function may raise floating-point exceptions (invalid operation, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

This function is implemented as inline, and thus is not ANSI-C compliant.

2.71.6.5 Code Example

```
#include "mlib.h"
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
tFloat fltIn;
tFrac16 f16Out;

void main(void)
{
    // input value = 0.25
    fltIn = (tFloat)0.25;

    // output should be FRAC16(0.25) = 0x2000
    f16Out = MLIB_ConvertPU_F16FLT(fltIn);

    // output should be FRAC16(0.25) = 0x2000
    f16Out = MLIB_ConvertPU(fltIn, F16FLT);
}
```

2.71.7 Function MLIB_ConvertPU_FLTF16

2.71.7.1 Declaration

```
INLINE tFloat MLIB_ConvertPU_FLTF16(register tFrac16 f16In);
```

2.71.7.2 Arguments

Table 371. MLIB_ConvertPU_FLTF16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	Input value in 16-bit fractional format to be converted.

2.71.7.3 Return

Converted input value in single precision floating point format.

2.71.7.4 Implementation details

The input value is considered as 16-bit fractional data type and output value is considered as single precision floating point data type. The output saturation is not implemented in this function.

The output of the function is defined by the following simple equation:

$$fltOut = \frac{(tFloat)f16In}{(tFloat)INT16_MAX+1}$$

Equation MLIB_ConvertPU_FLTF16_Eq1

Note: *The function may raise floating-point exceptions (invalid operation, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

This function is implemented as inline, and thus is not ANSI-C compliant.

2.71.7.5 Code Example

```
#include "mlib.h"

tF16 f16In;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
tFloat fltOut;

void main(void)
{
    // input value = 0.25 = 0x2000
    f16In = FRAC16(0.25);

    // output should be 0.25
    fltOut = MLIB_ConvertPU_FLTF16(f16In);

    // output should be 0.25
    fltOut = MLIB_ConvertPU(f16In, FLTF16);
}
```

2.71.8 Function MLIB_ConvertPU_FLTF32

2.71.8.1 Declaration

```
INLINE tFloat MLIB_ConvertPU_FLTF32(register tFrac32 f32In);
```

2.71.8.2 Arguments

Table 372. MLIB_ConvertPU_FLTF32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value in 32-bit fractional format to be converted.

2.71.8.3 Return

Converted input value in single precision floating point format.

2.71.8.4 Implementation details

The input value is considered as 32-bit fractional data type and output value is considered as single precision floating point data type. The output saturation is not implemented in this function.

The output of the function is defined by the following simple equation:

$$fltOut = \frac{(tFloat)f32In}{(tFloat)INT32_MAX+1}$$

Equation MLIB_ConvertPU_FLTF32_Eq1

Note: *The function may raise floating-point exceptions (invalid operation, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

This function is implemented as inline, and thus is not ANSI-C compliant.

2.71.8.5 Code Example

```
#include "mlib.h"

tF32 f32In;
tFloat fltOut;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
void main(void)
{
    // input value = 0.25 = 0x20000000
    f32In = FRAC32(0.25);

    // output should be 0.25
    fltOut = MLIB_ConvertPU_FLTF32(f32In);

    // output should be 0.25
    fltOut = MLIB_ConvertPU(f32In, FLTF32);
}
```

2.72 Function MLIB_Div

This function divides the first parameter by the second one.

2.72.1 Description

This inline function returns the division of two input values. The first input value is numerator and the second input value is denominator.

2.72.2 Re-entrancy

The function is re-entrant.

2.72.3 Function MLIB_Div_F32

2.72.3.1 Declaration

```
INLINE tFrac32 MLIB_Div_F32(register tFrac32 f32In1, register
tFrac32 f32In2);
```

2.72.3.2 Arguments

Table 373. MLIB_Div_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Numerator of division.
register tFrac32	f32In2	**input**	Denominator of division.

2.72.3.3 Return

Division of two input values.

2.72.3.4 Implementation details

The input values as well as output value is considered as 32-bit fractional data type. The output saturation is not implemented in this function, thus in case the numerator is greater or equal to denominator, the output value is undefined. The function will never cause division by zero exception. This is the fastest variant of the 32-bit division and should be used only if the inputs are limited elsewhere to prevent overflow. Note that only the most significant 16 bits of the result are accurate. Use MLIB_DivSat_F32 instead if the inputs to the function can cause overflow. If full 32-bit accuracy of the result is needed, use MLIB_DivSatAccurate_F32 (note that the accurate variant consumes more clock cycles).

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

2.72.3.5 Code Example

```
#include "mlib.h"

tFrac32 f32In1,f32In2;
tFrac32 f32Out;

void main(void)
{
  // input value 1 = 0.25
  f32In1 = FRAC32(0.25);
  // input value 2 = 0.5
  f32In2 = FRAC32(0.5);

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_Div_F32(f32In1,f32In2);

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_Div(f32In1,f32In2,F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_Div(f32In1,f32In2);
}
```

2.72.4 Function MLIB_Div_F16

2.72.4.1 Declaration

```
INLINE tFrac16 MLIB_Div_F16(register tFrac16 f16In1, register
tFrac16 f16In2);
```

2.72.4.2 Arguments

Table 374. MLIB_Div_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Numerator of division.
register tFrac16	f16In2	**input**	Denominator of division.

2.72.4.3 Return

Division of two input values.

2.72.4.4 Implementation details

The input values as well as output value is considered as 16-bit fractional data type. The output saturation is not implemented in this function, thus in case the numerator is greater or equal to denominator, the output value is undefined. The function will never cause division by zero exception. This is the fastest variant of the 16-bit division

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

and should be used only if the inputs are limited elsewhere to prevent overflow. Use MLIB_DivSat_F16 instead if the inputs to the function can cause overflow.

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

2.72.4.5 Code Example

```
#include "mlib.h"

tFrac16 f16In1,f16In2;
tFrac16 f16Out;

void main(void)
{
  // input value 1 = 0.25
  f16In1 = FRAC16(0.25);
  // input value 2 = 0.5
  f16In2 = FRAC16(0.5);

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_Div_F16(f16In1,f16In2);

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_Div(f16In1,f16In2,F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_Div(f16In1,f16In2);
}
```

2.72.5 Function MLIB_Div_FLT

2.72.5.1 Declaration

```
INLINE tFloat MLIB_Div_FLT(register tFloat fltIn1, register
tFloat fltIn2);
```

2.72.5.2 Arguments

Table 375. MLIB_Div_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn1	**input**	Numerator of division.
register tFloat	fltIn2	**input**	Denominator of division.

2.72.5.3 Return

Division of two input values.

2.72.5.4 Implementation details

The input values as well as output value is considered as single precision floating point data type. Testing of input values for division by zero is not implemented in this function.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Note: *The function may raise floating-point exceptions (invalid operation,division by zero, overflow,underflow,inexact,input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

This function is implemented as inline, and thus is not ANSI-C compliant.

2.72.5.5 Code Example

```
#include "mlib.h"

tFloat fltIn1,fltIn2;
tFloat fltOut;

void main(void)
{
  // input value 1 = 0.25
  fltIn1 = (tFloat)0.25;
  // input value 2 = 0.5
  fltIn2 = (tFloat)0.5;

  // output should be 0.5
  fltOut = MLIB_Div_FLT(fltIn1,fltIn2);

  // output should be 0.5
  fltOut = MLIB_Div(fltIn1,fltIn2,FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output should be 0.5
  fltOut = MLIB_Div(fltIn1,fltIn2);
}
```

2.73 Function MLIB_DivSat

This function divides the first parameter by the second one and saturate.

2.73.1 Description

This inline function returns the saturated division of two input values. The first input value is numerator and the second input value is denominator.

2.73.2 Re-entrancy

The function is re-entrant.

2.73.3 Function MLIB_DivSat_F32

2.73.3.1 Declaration

```
INLINE tFrac32 MLIB_DivSat_F32(register tFrac32 f32In1, register
tFrac32 f32In2);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.73.3.2 Arguments

Table 376. MLIB_DivSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Numerator of division.
register tFrac32	f32In2	**input**	Denominator of division.

2.73.3.3 Return

Fast division of two input values, saturated if necessary.

2.73.3.4 Implementation details

The input values as well as output value is considered as 32-bit fractional data type. This is a fast variant of the 32-bit division with only the most significant 16 bits of the result being accurate. If full 32-bit accuracy of the result is needed, use MLIB_DivSatAccurate_F32 (note that the accurate variant consumes more clock cycles).

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32_MIN & if \quad \frac{f32In1}{f32In2} < FRAC32_MIN \\ \frac{f32In1}{f32In2} & if \quad FRAC32_MIN \le \frac{f32In1}{f32In2} \le FRAC32_MAX \\ FRAC32_MAX & if \quad \frac{f32In1}{f32In2} > FRAC32_MAX \end{cases}$$

Equation MLIB_DivSat_F32_Eq1

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

2.73.3.5 Code Example

```
#include "mlib.h"

tFrac32 f32In1, f32In2;
tFrac32 f32Out;

void main(void)
{
  // input value 1 = 0.25
  f32In1 = FRAC32(0.25);
  // input value 2 = 0.5
  f32In2 = FRAC32(0.5);

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_DivSat_F32(f32In1, f32In2);

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_DivSat(f32In1, f32In2, F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_DivSat(f32In1, f32In2);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
}
```

2.73.4 Function MLIB_DivSatAccurate_F32

2.73.4.1 Declaration

```
INLINE tFrac32 MLIB_DivSatAccurate_F32(register tFrac32 f32In1,
register tFrac32 f32In2);
```

2.73.4.2 Arguments

Table 377. MLIB_DivSatAccurate_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Numerator of division.
register tFrac32	f32In2	**input**	Denominator of division.

2.73.4.3 Return

Accurate division of two input values, saturated if necessary.

2.73.4.4 Implementation details

The input values as well as output value is considered as 32-bit fractional data type. This variant provides full 32-bit accuracy of the result at the cost of consuming more clock cycles. If only the most significant 16 bits of the result need to be accurate, use the faster variant MLIB_DivSat_F32.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32_MIN & if \quad \frac{f32In1}{f32In2} < FRAC32_MIN \\ \frac{f32In1}{f32In2} & if \quad FRAC32_MIN \le \frac{f32In1}{f32In2} \le FRAC32_MAX \\ FRAC32_MAX & if \quad \frac{f32In1}{f32In2} > FRAC32_MAX \end{cases}$$

Equation MLIB_DivSat_F32_Eq1

Note: *Only the implementation suffix-based format of the API is provided.*

This function is implemented as inline, and thus is not ANSI-C compliant.

2.73.4.5 Code Example

```
#include "mlib.h"

tFrac32 f32In1, f32In2;
tFrac32 f32Out;

void main(void)
{
  // input value 1 = 0.25
  f32In1 = FRAC32(0.25);
  // input value 2 = 0.5
  f32In2 = FRAC32(0.5);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // Note: Only the implementation suffix-based format of the API is provided.
  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_DivSatAccurate_F32(f32In1, f32In2);
}
```

2.73.5 Function MLIB_DivSat_F16

2.73.5.1 Declaration

```
INLINE tFrac16 MLIB_DivSat_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

2.73.5.2 Arguments

Table 378. MLIB_DivSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Numerator of division.
register tFrac16	f16In2	**input**	Denominator of division.

2.73.5.3 Return

Division of two input values, saturated if necessary.

2.73.5.4 Implementation details

The input values as well as output value is considered as 16-bit fractional data type.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16_MIN & if \quad \frac{f16In1}{f16In2} < FRAC16_MIN \\ \frac{f16In1}{f16In2} & if \quad FRAC16_MIN \le \frac{f16In1}{f16In2} \le FRAC16_MAX \\ FRAC16_MAX & if \quad \frac{f16In1}{f16In2} > FRAC16_MAX \end{cases}$$

Equation MLIB_DivSat_F16_Eq1

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

2.73.5.5 Code Example

```
#include "mlib.h"

tFrac16 f16In1, f16In2;
tFrac16 f16Out;

void main(void)
{
  // input value 1 = 0.25
  f16In1 = FRAC16(0.25);
  // input value 2 = 0.5
  f16In2 = FRAC16(0.5);

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_DivSat_F16(f16In1, f16In2);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_DivSat(f16In1, f16In2, F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_DivSat(f16In1, f16In2);
}
```

2.74 Function MLIB_Mac

This function implements the multiply accumulate function.

2.74.1 Description

This inline function returns the multiplied second and third input value with adding of first input value.

2.74.2 Re-entrancy

The function is re-entrant.

2.74.3 Function MLIB_Mac_F32

2.74.3.1 Declaration

```
INLINE tFrac32 MLIB_Mac_F32(register tFrac32 f32In1, register
tFrac32 f32In2, register tFrac32 f32In3);
```

2.74.3.2 Arguments

Table 379. MLIB_Mac_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Input value to be add.
register tFrac32	f32In2	**input**	First value to be multiplied.
register tFrac32	f32In3	**input**	Second value to be multiplied.

2.74.3.3 Return

Multiplied second and third input value with adding of first input value.

2.74.3.4 Implementation details

The input values as well as output value is considered as 32-bit fractional values. The output saturation is not implemented in this function, thus in case the output value is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f32Out = f32In1 + (f32In2 \cdot f32In3)$$

Equation MLIB_Mac_F32_Eq1

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

2.74.3.5 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32In3;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.25
  f32In1 = FRAC32(0.25);

  // input2 value = 0.15
  f32In2 = FRAC32(0.15);

  // input3 value = 0.35
  f32In3 = FRAC32(0.35);

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_Mac_F32(f32In1, f32In2, f32In3);

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_Mac(f32In1, f32In2, f32In3, F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_Mac(f32In1, f32In2, f32In3);
}
```

2.74.4 Function MLIB_Mac_F32F16F16

2.74.4.1 Declaration

```
INLINE tFrac32 MLIB_Mac_F32F16F16(register tFrac32 f32In1,
register tFrac16 f16In2, register tFrac16 f16In3);
```

2.74.4.2 Arguments

Table 380. MLIB_Mac_F32F16F16 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Input value to be add.
register tFrac16	f16In2	**input**	First value to be multiplied.
register tFrac16	f16In3	**input**	Second value to be multiplied.

2.74.4.3 Return

Multiplied second and third input value with adding of first input value.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.74.4.4 Implementation details

The first input value as well as output value is considered as 32-bit fractional values. The second and third input values are considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the output value is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f32Out = f32In1 + (f16In2 \cdot f16In3)$$

Equation MLIB_Mac_F32F16F16_Eq1

This implementation is available if 32-bit fractional implementations are enabled. However it is not possible to use the default implementation based function call, thus the implementation post-fix or additional parameter function call shall be used.

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

2.74.4.5 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.25
  f32In1 = FRAC32(0.25);

  // input2 value = 0.15
  f16In2 = FRAC16(0.15);

  // input3 value = 0.35
  f16In3 = FRAC16(0.35);

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_Mac_F32F16F16(f32In1, f16In2, f16In3);

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_Mac(f32In1, f32In2, f32In3, F32F16F16);

}
```

2.74.5 Function MLIB_Mac_F16

2.74.5.1 Declaration

```
INLINE tFrac16 MLIB_Mac_F16(register tFrac16 f16In1, register
tFrac16 f16In2, register tFrac16 f16In3);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.74.5.2 Arguments

Table 381. MLIB_Mac_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Input value to be add.
register tFrac16	f16In2	**input**	First value to be multiplied.
register tFrac16	f16In3	**input**	Second value to be multiplied.

2.74.5.3 Return

Multiplied second and third input value with adding of first input value.

2.74.5.4 Implementation details

The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the output value is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f16Out = f16In1 + (f16In2 \cdot f16In3)$$

Equation MLIB_Mac_F16_Eq1

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

2.74.5.5 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac16 f16Out;

void main(void)
{
  // input1 value = 0.25
  f16In1 = FRAC16(0.25);

  // input2 value = 0.15
  f16In2 = FRAC16(0.15);

  // input3 value = 0.35
  f16In3 = FRAC16(0.35);

  // output should be FRAC16(0.3025) = 0x26B8
  f16Out = MLIB_Mac_F16(f16In1, f16In2, f16In3);

  // output should be FRAC16(0.3025) = 0x26B8
  f16Out = MLIB_Mac(f16In1, f16In2, f16In3, F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC16(0.3025) = 0x26B8
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  f16Out = MLIB_Mac(f16In1, f16In2, f16In3);
}
```

2.74.6 Function MLIB_Mac_FLT

2.74.6.1 Declaration

```
INLINE tFloat MLIB_Mac_FLT(register tFloat fltIn1, register
tFloat fltIn2, register tFloat fltIn3);
```

2.74.6.2 Arguments

Table 382. MLIB_Mac_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn1	**input**	Input value to be add.
register tFloat	fltIn2	**input**	First value to be multiplied.
register tFloat	fltIn3	**input**	Second value to be multiplied.

2.74.6.3 Return

Multiplied second and third input value with adding of first input value.

2.74.6.4 Implementation details

The input values as well as output value are considered as single precision floating point data type. Intermediate results are computed in infinite precision.

The output of the function is defined by the following simple equation:

$$fltOut = fltIn1 + (fltIn2 \cdot fltIn3)$$

Equation MLIB_Mac_FLT_Eq1

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

This function is implemented as inline assembly, and thus is not ANSI-C compliant.

2.74.6.5 Code Example

```
#include "mlib.h"

tFloat fltIn1;
tFloat fltIn2;
tFloat fltIn3;
tFloat fltOut;

void main(void)
{
  // input1 value = 0.25
  fltIn1 = (tFloat)0.25;

  // input2 value = 0.15
  fltIn2 = (tFloat)0.15;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // input3 value = 0.35
  fltIn3 = (tFloat)0.35;

  // output should be 0.3025
  fltOut = MLIB_Mac_FLT(fltIn1, fltIn2, fltIn3);

  // output should be 0.3025
  fltOut = MLIB_Mac(fltIn1, fltIn2, fltIn3, FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output should be 0.3025
  fltOut = MLIB_Mac(fltIn1, fltIn2, fltIn3);
}
```

2.75 Function MLIB_MacSat

This function implements the multiply accumulate function saturated if necessary.

2.75.1 Description

This inline function returns the multiplied second and third input value with adding of first input value. The output value is saturated if necessary.

2.75.2 Re-entrancy

The function is re-entrant.

2.75.3 Function MLIB_MacSat_F32

2.75.3.1 Declaration

```
INLINE tFrac32 MLIB_MacSat_F32(register tFrac32 f32In1, register
tFrac32 f32In2, register tFrac32 f32In3);
```

2.75.3.2 Arguments

Table 383. MLIB_MacSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Input value to be add.
register tFrac32	f32In2	**input**	First value to be multiplied.
register tFrac32	f32In3	**input**	Second value to be multiplied.

2.75.3.3 Return

Multiplied second and third input value with adding of first input value. The output value is saturated if necessary.

2.75.3.4 Implementation details

The input values as well as output value is considered as 32-bit fractional values.

The output of the function is defined by the following simple equation:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$f32Out = \begin{cases} FRAC32_MIN & if \quad (f32In1 + (f32In2 \cdot f32In3)) < FRAC32_MIN \\ f32In1 + (f32In2 \cdot f32In3) & if \quad FRAC32_MIN \le (f32In1 + (f32In2 \cdot f32In3)) \le FRAC32_MAX \\ FRAC32_MAX & if \quad (f32In1 + (f32In2 \cdot f32In3)) > FRAC32_MAX \end{cases}$$

Equation MLIB_MacSat_F32_Eq1

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.75.3.5 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32In3;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.25
  f32In1  = FRAC32(0.25);

  // input2 value = 0.15
  f32In2  = FRAC32(0.15);

  // input3 value = 0.35
  f32In3  = FRAC32(0.35);

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_MacSat_F32(f32In1, f32In2, f32In3);

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_MacSat(f32In1, f32In2, f32In3, F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_MacSat(f32In1, f32In2, f32In3);
}
```

2.75.4 Function MLIB_MacSat_F32F16F16

2.75.4.1 Declaration

```
INLINE tFrac32 MLIB_MacSat_F32F16F16(register tFrac32 f32In1,
register tFrac16 f16In2, register tFrac16 f16In3);
```

2.75.4.2 Arguments

Table 384. MLIB_MacSat_F32F16F16 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Input value to be add.
register tFrac16	f16In2	**input**	First value to be multiplied.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 384. MLIB_MacSat_F32F16F16 arguments...*continued*

Type	Name	Direction	Description
register tFrac16	f16In3	**input**	Second value to be multiplied.

2.75.4.3 Return

Multiplied second and third input value with adding of first input value. The output value is saturated if necessary.

2.75.4.4 Implementation details

The first input values as well as output value is considered as 32-bit fractional values, second and third input values are considered as 16-bit fractional values.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32_MIN & if \quad (f32In1 + (f16In2 \cdot f16In3)) < FRAC32_MIN \\ f32In1 + (f16In2 \cdot f16In3) & if \quad FRAC32_MIN \le (f32In1 + (f16In2 \cdot f16In3)) \le FRAC32_MAX \\ FRAC32_MAX & if \quad (f32In1 + (f16In2 \cdot f16In3)) > FRAC32_MAX \end{cases}$$

Equation MLIB_MacSat_F32F16F16_Eq1

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.75.4.5 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.25
  f32In1 = FRAC32(0.25);

  // input2 value = 0.15
  f16In2 = FRAC16(0.15);

  // input3 value = 0.35
  f16In3 = FRAC16(0.35);

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_MacSat_F32F16F16(f32In1, f16In2, f16In3);

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_MacSat(f32In1, f16In2, f16In3, F32F16F16);
}
```

2.75.5 Function MLIB_MacSat_F16

2.75.5.1 Declaration

```
INLINE tFrac16 MLIB_MacSat_F16(register tFrac16 f16In1, register
tFrac16 f16In2, register tFrac16 f16In3);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.75.5.2 Arguments

Table 385. MLIB_MacSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Input value to be add.
register tFrac16	f16In2	**input**	First value to be multiplied.
register tFrac16	f16In3	**input**	Second value to be multiplied.

2.75.5.3 Return

Multiplied second and third input value with adding of first input value. The output value is saturated if necessary.

2.75.5.4 Implementation details

The input values as well as output value is considered as 16-bit fractional values.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16_MIN & if \quad (f16In1 + (f16In2 \cdot f16In3)) < FRAC16_MIN \\ f16In1 + (f16In2 \cdot f16In3) & if \quad FRAC16_MIN \le (f16In1 + (f16In2 \cdot f16In3)) \le FRAC16_MAX \\ FRAC16_MAX & if \quad (f16In1 + (f16In2 \cdot f16In3)) > FRAC16_MAX \end{cases}$$

Equation MLIB_MacSat_F16_Eq1

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.75.5.5 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac16 f16Out;

void main(void)
{
  // input1 value = 0.25
  f16In1 = FRAC16(0.25);

  // input2 value = 0.15
  f16In2 = FRAC16(0.15);

  // input3 value = 0.35
  f16In3 = FRAC16(0.35);

  // output should be FRAC16(0.3025) = 0x26B8
  f16Out = MLIB_MacSat_F16(f16In1, f16In2, f16In3);

  // output should be FRAC16(0.3025) = 0x26B8
  f16Out = MLIB_MacSat(f16In1, f16In2, f16In3, F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // output should be FRAC16(0.3025) = 0x26B8
  f16Out = MLIB_MacSat(f16In1, f16In2, f16In3);
}
```

2.76 Function MLIB_Mnac

This function implements the multiply-subtract function.

2.76.1 Description

This inline function returns the multiplied second and third input value with subtracted first input value.

2.76.2 Re-entrancy

The function is re-entrant.

2.76.3 Function MLIB_Mnac_F32

2.76.3.1 Declaration

```
INLINE tFrac32 MLIB_Mnac_F32(register tFrac32 f32In1, register
tFrac32 f32In2, register tFrac32 f32In3);
```

2.76.3.2 Arguments

Table 386. MLIB_Mnac_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Input value to be subtracted.
register tFrac32	f32In2	**input**	First value to be multiplied.
register tFrac32	f32In3	**input**	Second value to be multiplied.

2.76.3.3 Return

Multiplied second and third input value with subtracted first input value.

2.76.3.4 Implementation details

The input values as well as output value is considered as 32-bit fractional values. The output saturation is not implemented in this function, thus in case the output value is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f32Out = -f32In1 + (f32In2 \cdot f32In3)$$

Equation MLIB_Mnac_F32_Eq1

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

2.76.3.5 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
tFrac32 f32In2;
tFrac32 f32In3;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.0625
  f32In1 = FRAC32(0.0625);

  // input2 value = 0.5
  f32In2 = FRAC32(0.5);

  // input3 value = 0.25
  f32In3 = FRAC32(0.25);

  // output should be FRAC32(0.0625) = 0x08000000
  f32Out = MLIB_Mnac_F32(f32In1, f32In2, f32In3);

  // output should be FRAC32(0.0625) = 0x08000000
  f32Out = MLIB_Mnac(f32In1, f32In2, f32In3, F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC32(0.0625) = 0x08000000
  f32Out = MLIB_Mnac(f32In1, f32In2, f32In3);
}
```

2.76.4 Function MLIB_Mnac_F32F16F16

2.76.4.1 Declaration

```
INLINE tFrac32 MLIB_Mnac_F32F16F16(register tFrac32 f32In1,
register tFrac16 f16In2, register tFrac16 f16In3);
```

2.76.4.2 Arguments

Table 387. MLIB_Mnac_F32F16F16 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Input value to be subtracted.
register tFrac16	f16In2	**input**	First value to be multiplied.
register tFrac16	f16In3	**input**	Second value to be multiplied.

2.76.4.3 Return

Multiplied second and third input value with subtracted first input value.

2.76.4.4 Implementation details

The first input value as well as output value is considered as 32-bit fractional values. The second and third input values are considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the output value is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

$$f32Out = -f32In1 + (f16In2 \cdot f16In3)$$

Equation MLIB_Mnac_F32F16F16_Eq1

This implementation is available if 32-bit fractional implementations are enabled. However it is not possible to use the default implementation based function call, thus the implementation post-fix or additional parameter function call shall be used.

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

2.76.4.5 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.0625
  f32In1 = FRAC32(0.0625);

  // input2 value = 0.5
  f16In2 = FRAC16(0.5);

  // input3 value = 0.25
  f16In3 = FRAC16(0.25);

  // output should be FRAC32(0.0625) = 0x08000000
  f32Out = MLIB_Mnac_F32F16F16(f32In1, f16In2, f16In3);

  // output should be FRAC32(0.0625) = 0x08000000
  f32Out = MLIB_Mnac(f32In1, f32In2, f32In3, F32F16F16);

}
```

2.76.5 Function MLIB_Mnac_F16

2.76.5.1 Declaration

```
INLINE tFrac16 MLIB_Mnac_F16(register tFrac16 f16In1, register
tFrac16 f16In2, register tFrac16 f16In3);
```

2.76.5.2 Arguments

Table 388. MLIB_Mnac_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Input value to be subtracted.
register tFrac16	f16In2	**input**	First value to be multiplied.
register tFrac16	f16In3	**input**	Second value to be multiplied.

2.76.5.3 Return

Multiplied second and third input value with subtracted first input value.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.76.5.4 Implementation details

The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the output value is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f16Out = -f16In1 + (f16In2 \cdot f16In3)$$

Equation MLIB_Mnac_F16_Eq1

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

2.76.5.5 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac16 f16Out;

void main(void)
{
  // input1 value = 0.0625
  f16In1 = FRAC16(0.0625);

  // input2 value = 0.5
  f16In2 = FRAC16(0.5);

  // input3 value = 0.25
  f16In3 = FRAC16(0.25);

  // output should be FRAC16(0.0625) = 0x0800
  f16Out = MLIB_Mnac_F16(f16In1, f16In2, f16In3);

  // output should be FRAC16(0.0625) = 0x0800
  f16Out = MLIB_Mnac(f16In1, f16In2, f16In3, F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC16(0.0625) = 0x0800
  f16Out = MLIB_Mnac(f16In1, f16In2, f16In3);
}
```

2.76.6 Function MLIB_Mnac_FLT

2.76.6.1 Declaration

```
INLINE tFloat MLIB_Mnac_FLT(register tFloat fltIn1, register
tFloat fltIn2, register tFloat fltIn3);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.76.6.2 Arguments

Table 389. MLIB_Mnac_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn1	**input**	Input value to be subtracted.
register tFloat	fltIn2	**input**	First value to be multiplied.
register tFloat	fltIn3	**input**	Second value to be multiplied.

2.76.6.3 Return

Multiplied second and third input value with subtracted first input value.

2.76.6.4 Implementation details

The input values as well as output value are considered as single precision floating point data type. Intermediate results are computed in infinite precision.

The output of the function is defined by the following simple equation:

$$fltOut = -fltIn1 + (fltIn2 \cdot fltIn3)$$

Equation MLIB_Mnac_FLT_Eq1

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

This function is implemented as inline assembly, and thus is not ANSI-C compliant.

2.76.6.5 Code Example

```
#include "mlib.h"

tFloat fltIn1;
tFloat fltIn2;
tFloat fltIn3;
tFloat fltOut;

void main(void)
{
  // input1 value = 5.775817036628723e-01
  fltIn1 = (tFloat)5.775817036628723e-01f;

  // input2 value = 9.133758544921875e-01
  fltIn2 = (tFloat)9.133758544921875e-01f;

  // input3 value = 6.323592662811279e-01
  fltIn3 = (tFloat)6.323592662811279e-01f;

  // output should be -1.8477294e-08
  fltOut = MLIB_Mnac_FLT(fltIn1, fltIn2, fltIn3);

  // output should be -1.8477294e-08
  fltOut = MLIB_Mnac(fltIn1, fltIn2, fltIn3, FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // ##########################################################

  // output should be -1.8477294e-08
  fltOut = MLIB_Mnac(fltIn1, fltIn2, fltIn3);
}
```

2.77 Function MLIB_Msu

This function implements the multiply-subtract-from function.

2.77.1 Description

This inline function returns the first input value from which the multiplication result of the second and third input values is subtracted.

2.77.2 Re-entrancy

The function is re-entrant.

2.77.3 Function MLIB_Msu_F32

2.77.3.1 Declaration

```
INLINE tFrac32 MLIB_Msu_F32(register tFrac32 f32In1, register
tFrac32 f32In2, register tFrac32 f32In3);
```

2.77.3.2 Arguments

Table 390. MLIB_Msu_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Input value from which to subtract.
register tFrac32	f32In2	**input**	First value to be multiplied.
register tFrac32	f32In3	**input**	Second value to be multiplied.

2.77.3.3 Return

First input value from which the multiplication result of the second and third input values is subtracted.

2.77.3.4 Implementation details

The input values as well as output value is considered as 32-bit fractional values. The output saturation is not implemented in this function, thus in case the output value is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f32Out = f32In1 - (f32In2 \cdot f32In3)$$

Equation MLIB_Msu_F32_Eq1

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.77.3.5 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32In3;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.25
  f32In1 = FRAC32(0.25);

  // input2 value = 0.5
  f32In2 = FRAC32(0.5);

  // input3 value = 0.125
  f32In3 = FRAC32(0.125);

  // output should be FRAC32(0.1875) = 0x18000000
  f32Out = MLIB_Msu_F32(f32In1, f32In2, f32In3);

  // output should be FRAC32(0.1875) = 0x18000000
  f32Out = MLIB_Msu(f32In1, f32In2, f32In3, F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC32(0.1875) = 0x18000000
  f32Out = MLIB_Msu(f32In1, f32In2, f32In3);
}
```

2.77.4 Function MLIB_Msu_F32F16F16

2.77.4.1 Declaration

```
INLINE tFrac32 MLIB_Msu_F32F16F16(register tFrac32 f32In1,
register tFrac16 f16In2, register tFrac16 f16In3);
```

2.77.4.2 Arguments

Table 391. MLIB_Msu_F32F16F16 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Input value from which to subtract.
register tFrac16	f16In2	**input**	First value to be multiplied.
register tFrac16	f16In3	**input**	Second value to be multiplied.

2.77.4.3 Return

First input value from which the multiplication result of the second and third input values is subtracted.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.77.4.4 Implementation details

The first input value as well as output value is considered as 32-bit fractional values. The second and third input values are considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the output value is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f32Out = f32In1 - (f16In2 \cdot f16In3)$$

Equation MLIB_Msu_F32F16F16_Eq1

This implementation is available if 32-bit fractional implementations are enabled. However it is not possible to use the default implementation based function call, thus the implementation post-fix or additional parameter function call shall be used.

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

2.77.4.5 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.25
  f32In1 = FRAC32(0.25);

  // input2 value = 0.5
  f16In2 = FRAC16(0.5);

  // input3 value = 0.125
  f16In3 = FRAC16(0.125);

  // output should be FRAC32(0.1875) = 0x18000000
  f32Out = MLIB_Msu_F32F16F16(f32In1, f16In2, f16In3);

  // output should be FRAC32(0.1875) = 0x18000000
  f32Out = MLIB_Msu(f32In1, f32In2, f32In3, F32F16F16);

}
```

2.77.5 Function MLIB_Msu_F16

2.77.5.1 Declaration

```
INLINE tFrac16 MLIB_Msu_F16(register tFrac16 f16In1, register
tFrac16 f16In2, register tFrac16 f16In3);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.77.5.2 Arguments

Table 392. MLIB_Msu_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Input value from which to subtract.
register tFrac16	f16In2	**input**	First value to be multiplied.
register tFrac16	f16In3	**input**	Second value to be multiplied.

2.77.5.3 Return

First input value from which the multiplication result of the second and third input values is subtracted.

2.77.5.4 Implementation details

The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the output value is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f16Out = f16In1 - (f16In2 \cdot f16In3)$$

Equation MLIB_Msu_F16_Eq1

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

2.77.5.5 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac16 f16Out;

void main(void)
{
  // input1 value = 0.25
  f16In1 = FRAC16(0.25);

  // input2 value = 0.5
  f16In2 = FRAC16(0.5);

  // input3 value = 0.125
  f16In3 = FRAC16(0.125);

  // output should be FRAC16(0.1875) = 0x1800
  f16Out = MLIB_Msu_F16(f16In1, f16In2, f16In3);

  // output should be FRAC16(0.1875) = 0x1800
  f16Out = MLIB_Msu(f16In1, f16In2, f16In3, F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // output should be FRAC16(0.1875) = 0x1800
  f16Out = MLIB_Msu(f16In1, f16In2, f16In3);
}
```

2.77.6 Function MLIB_Msu_FLT

2.77.6.1 Declaration

```
INLINE tFloat MLIB_Msu_FLT(register tFloat fltIn1, register
tFloat fltIn2, register tFloat fltIn3);
```

2.77.6.2 Arguments

Table 393. MLIB_Msu_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn1	**input**	Input value from which to subtract.
register tFloat	fltIn2	**input**	First value to be multiplied.
register tFloat	fltIn3	**input**	Second value to be multiplied.

2.77.6.3 Return

First input value from which the multiplication result of the second and third input values is subtracted.

2.77.6.4 Implementation details

The input values as well as output value are considered as single precision floating point data type. Intermediate results are computed in infinite precision.

The output of the function is defined by the following simple equation:

$$fltOut = fltIn1 - (fltIn2 \cdot fltIn3)$$

Equation MLIB_Msu_FLT_Eq1

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

This function is implemented as inline assembly, and thus is not ANSI-C compliant.

2.77.6.5 Code Example

```
#include "mlib.h"

tFloat fltIn1;
tFloat fltIn2;
tFloat fltIn3;
tFloat fltOut;

void main(void)
{
  // input1 value = 1.150236353278160e-01
  fltIn1 = (tFloat)1.150236353278160e-01f;

  // input2 value = 9.057919383049011e-01
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  fltIn2 = (tFloat)9.057919383049011e-01f;

  // input3 value = 1.269868165254593e-01
  fltIn3 = (tFloat)1.269868165254593e-01f;

  // output should be 6.4805139e-10
  fltOut = MLIB_Msu_FLT(fltIn1, fltIn2, fltIn3);

  // output should be 6.4805139e-10
  fltOut = MLIB_Msu(fltIn1, fltIn2, fltIn3, FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output should be 6.4805139e-10
  fltOut = MLIB_Msu(fltIn1, fltIn2, fltIn3);
}
```

2.78 Function MLIB_Mul

This function multiplies two input parameters.

2.78.1 Description

This inline function multiplies the two input values.

2.78.2 Re-entrancy

The function is re-entrant.

2.78.3 Function MLIB_Mul_F32

2.78.3.1 Declaration

```
INLINE tFrac32 MLIB_Mul_F32(register tFrac32 f32In1, register
tFrac32 f32In2);
```

2.78.3.2 Arguments

Table 394. MLIB_Mul_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Operand is a 32-bit number normalized between [-1,1).
register tFrac32	f32In2	**input**	Operand is a 32-bit number normalized between [-1,1).

2.78.3.3 Return

Fractional multiplication of the input arguments.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.78.3.4 Implementation details

The input values as well as output value is considered as 32-bit fractional values. The output saturation is not implemented in this function, thus in case the multiplication of input values is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f32Out = f32In1 \cdot f32In2$$

Equation MLIB_Mul_F32_Eq1

Note: *Overflow is not detected. This function is implemented as inline assembly and thus is not ANSI-C compliant.*

2.78.3.5 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32Out;

void main(void)
{
  // first input = 0.5
  f32In1 = FRAC32(0.5);

  // second input = 0.25
  f32In2 = FRAC32(0.25);

  // output should be 0x10000000 = FRAC32(0.125)
  f32Out = MLIB_Mul_F32(f32In1,f32In2);

  // output should be 0x10000000 = FRAC32(0.125)
  f32Out = MLIB_Mul(f32In1,f32In2,F32);


  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be 0x10000000 = FRAC32(0.125)
  f32Out = MLIB_Mul(f32In1,f32In2);
}
```

2.78.4 Function MLIB_Mul_F32F16F16

2.78.4.1 Declaration

```
INLINE tFrac32 MLIB_Mul_F32F16F16(register tFrac16 f16In1,
register tFrac16 f16In2);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.78.4.2 Arguments

Table 395. MLIB_Mul_F32F16F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In2	**input**	Operand is a 16-bit number normalized between [-1,1).

2.78.4.3 Return

Fractional multiplication of the input arguments.

2.78.4.4 Implementation details

The input values are considered as 16-bit fractional values and the output value is considered as 32-bit fractional value. The output saturation is not implemented in this function, thus in case the multiplication of input values is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f32Out = f16In1 \cdot f16In2$$

Equation MLIB_Mul_F32F16F16_Eq1

Note: *Overflow is not detected. This function is implemented as inline assembly and thus is not ANSI-C compliant.*

2.78.4.5 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac32 f32Out;

void main(void)
{
  // first input = 0.5
  f16In1 = FRAC16(0.5);

  // second input = 0.25
  f16In2 = FRAC16(0.25);

  // output should be 0x10000000 = FRAC32(0.125)
  f32Out = MLIB_Mul_F32F16F16(f16In1, f16In2);

  // output should be 0x10000000 = FRAC32(0.125)
  f32Out = MLIB_Mul(f16In1, f16In2, F32F16F16);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.78.5 Function MLIB_Mul_F16

2.78.5.1 Declaration

```
INLINE tFrac16 MLIB_Mul_F16(register tFrac16 f16In1, register
tFrac16 f16In2);
```

2.78.5.2 Arguments

Table 396. MLIB_Mul_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In2	**input**	Operand is a 16-bit number normalized between [-1,1).

2.78.5.3 Return

Fractional multiplication of the input arguments.

2.78.5.4 Implementation details

The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the multiplication of input values is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f16Out = f16In1 \cdot f16In2$$

Equation MLIB_Mul_F16_Eq1

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.78.5.5 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16Out;

void main(void)
{
  // first input = 0.5
  f16In1 = FRAC16(0.5);

  // second input = 0.25
  f16In2 = FRAC16(0.25);

  // output should be 0x1000 = FRAC16(0.125)
  f16Out = MLIB_Mul_F16(f16In1,f16In2);

  // output should be 0x1000 = FRAC16(0.125)
  f16Out = MLIB_Mul(f16In1,f16In2,F16);


  // ##############################################################
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be 0x1000 = FRAC16(0.125)
  f16Out = MLIB_Mul(f16In1,f16In2);
}
```

2.78.6 Function MLIB_Mul_FLT

2.78.6.1 Declaration

```
INLINE tFloat MLIB_Mul_FLT(register tFloat fltIn1, register
tFloat fltIn2);
```

2.78.6.2 Arguments

Table 397. MLIB_Mul_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn1	**input**	Operand is a single precision floating point number.
register tFloat	fltIn2	**input**	Operand is a single precision floating point number.

2.78.6.3 Return

Floating point multiplication of the input arguments.

2.78.6.4 Implementation details

The input values as well as output value is considered as single precision floating point data type.

The output of the function is defined by the following simple equation:

$$fltOut = fltIn1 \cdot fltIn2$$

Equation MLIB_Mul_FLT_Eq1

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

This function is implemented as inline and thus is not ANSI-C compliant.

2.78.6.5 Code Example

```
#include "mlib.h"

tFloat fltIn1;
tFloat fltIn2;
tFloat fltOut;

void main(void)
{
  // first input = 50.5
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  fltIn1 = (tFloat)50.5;

  // second input = 25.25
  fltIn2 = (tFloat)25.25;

  // output should be 1275.125
  fltOut = MLIB_Mul_FLT(fltIn1,fltIn2);

  // output should be 1275.125
  fltOut = MLIB_Mul(fltIn1,fltIn2,FLT);


  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output should be 1275.125
  fltOut = MLIB_Mul(fltIn1,fltIn2);
}
```

2.79 Function MLIB_MulSat

This function multiplies two input parameters and saturate if necessary.

2.79.1 Description

This inline function multiplies the two input values and saturates the result.

2.79.2 Re-entrancy

The function is re-entrant.

2.79.3 Function MLIB_MulSat_F32

2.79.3.1 Declaration

```
INLINE tFrac32 MLIB_MulSat_F32(register tFrac32 f32In1, register
tFrac32 f32In2);
```

2.79.3.2 Arguments

Table 398. MLIB_MulSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Operand is a 32-bit number normalized between [-1,1).
register tFrac32	f32In2	**input**	Operand is a 32-bit number normalized between [-1,1).

2.79.3.3 Return

Fractional multiplication of the input arguments.

2.79.3.4 Implementation details

The input values as well as output value are considered as 32-bit fractional data type.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32_MIN & if \quad (f32In1 \cdot f32In2) < FRAC32_MIN \\ f32In1 \cdot f32In2 & if \quad FRAC32_MIN \le (f32In1 \cdot f32In2) \le FRAC32_MAX \\ FRAC32_MAX & if \quad (f32In1 \cdot f32In2) > FRAC32_MAX \end{cases}$$

Equation MLIB_MulSat_F32_Eq1

Note: *This function is implemented as inline assembly and thus is not ANSI-C compliant.*

2.79.3.5 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32Out;

void main(void)
{
  // first input = 0.8
  f32In1 = FRAC32(0.8);

  // second input = 0.75
  f32In2 = FRAC32(0.75);

  // output should be 0x4cccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat_F32(f32In1,f32In2);

  // output should be 0x4cccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat(f32In1,f32In2,F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be 0x4cccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat(f32In1,f32In2);
}
```

2.79.4 Function MLIB_MulSat_F32F16F16

2.79.4.1 Declaration

```
INLINE tFrac32 MLIB_MulSat_F32F16F16(register tFrac16 f16In1,
register tFrac16 f16In2);
```

2.79.4.2 Arguments

Table 399. MLIB_MulSat_F32F16F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In2	**input**	Operand is a 16-bit number normalized between [-1,1).

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.79.4.3 Return

Fractional multiplication of the input arguments.

2.79.4.4 Implementation details

The input values are considered as 16-bit fractional data type and the output value is considered as 32-bit fractional data type.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32_MIN & if \quad (f16In1 \cdot f16In2) < FRAC32_MIN \\ f16In1 \cdot f16In2 & if \quad FRAC32_MIN \le (f16In1 \cdot f16In2) \le FRAC32_MAX \\ FRAC32_MAX & if \quad (f16In1 \cdot f16In2) > FRAC32_MAX \end{cases}$$

Equation MLIB_MulSat_F32F16F16_Eq1

Note: *This function is implemented as inline assembly and thus is not ANSI-C compliant.*

2.79.4.5 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac32 f32Out;

void main(void)
{
  // first input = 0.8
  f16In1 = FRAC16(0.8);

  // second input = 0.75
  f16In2 = FRAC16(0.75);

  // output should be 0x4cccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat_F32F16F16(f16In1,f16In2);

  // output should be 0x4cccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat(f32In1,f32In2,F32F16f16);
}
```

2.79.5 Function MLIB_MulSat_F16

2.79.5.1 Declaration

```
INLINE tFrac16 MLIB_MulSat_F16(register tFrac16 f16In1, register
tFrac16 f16In2);
```

2.79.5.2 Arguments

Table 400. MLIB_MulSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Operand is a 16-bit number normalized between [-1,1).

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 400. MLIB_MulSat_F16 arguments...*continued*

Type	Name	Direction	Description
register tFrac16	f16In2	**input**	Operand is a 16-bit number normalized between [-1,1).

2.79.5.3 Return

Fractional multiplication of the input arguments.

2.79.5.4 Implementation details

The input values as well as output value are considered as 16-bit fractional data type.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16_MIN & if \quad (f16In1 \cdot f16In2) < FRAC16_MIN \\ f16In1 \cdot f16In2 & if \quad FRAC16_MIN \le (f16In1 \cdot f16In2) \le FRAC16_MAX \\ FRAC16_MAX & if \quad (f16In1 \cdot f16In2) > FRAC16_MAX \end{cases}$$

Equation MLIB_MulSat_F16_Eq1

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.79.5.5 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16Out;

void main(void)
{
  // first input = 0.8
  f16In1 = FRAC16(0.8);

  // second input = 0.75
  f16In2 = FRAC16(0.75);

  // output should be 0x4ccc = FRAC16(0.6)
  f16Out = MLIB_MulSat_F16(f16In1,f16In2);

  // output should be 0x4ccc = FRAC16(0.6)
  f16Out = MLIB_MulSat(f16In1,f16In2,F16);

  // ##########################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##########################################################

  // output should be 0x4ccc = FRAC32(0.6)
  f16Out = MLIB_MulSat(f16In1,f16In2);
}
```

2.80 Function MLIB_Neg

This function returns negative value of input parameter.

© NXP B.V. 2023. All rights reserved.

2.80.1 Description

This inline function returns the negative value of input parameter.

2.80.2 Re-entrancy

The function is re-entrant.

2.80.3 Function MLIB_Neg_F32

2.80.3.1 Declaration

```
INLINE tFrac32 MLIB_Neg_F32(register tFrac32 f32In);
```

2.80.3.2 Arguments

Table 401. MLIB_Neg_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value which negative value should be returned.

2.80.3.3 Return

Negative value of input parameter.

2.80.3.4 Implementation details

The input values as well as output value is considered as 32-bit fractional values. The output saturation is not implemented in this function, thus in case the negation of input values is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f32Out = -f32In$$

Equation MLIB_Neg_F32_Eq1

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

2.80.3.5 Code Example

```
#include "mlib.h"

tFrac32 f32In;
tFrac32 f32Out;

void main(void)
{
  // input value = 0.25
  f32In = FRAC32(0.25);

  // output should be FRAC32(-0.25) = 0xA0000000
  f32Out = MLIB_Neg_F32(f32In);

  // output should be FRAC32(-0.25) = 0xA0000000
  f32Out = MLIB_Neg(f32In, F32);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC32(-0.25) = 0xA0000000
  f32Out = MLIB_Neg(f32In);
}
```

2.80.4 Function MLIB_Neg_F16

2.80.4.1 Declaration

```
INLINE tFrac16 MLIB_Neg_F16(register tFrac16 f16In);
```

2.80.4.2 Arguments

Table 402. MLIB_Neg_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	Input value which negative value should be returned.

2.80.4.3 Return

Negative value of input parameter.

2.80.4.4 Implementation details

The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the negation of input values is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f16Out = -f16In$$

Equation MLIB_Neg_F16_Eq1

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

2.80.4.5 Code Example

```
#include "mlib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
{
  // input value = 0.25
  f16In = FRAC16(0.25);

  // output should be FRAC16(-0.25) = 0xA000
  f16Out = MLIB_Neg_F16(f16In);

  // output should be FRAC16(-0.25) = 0xA000
  f16Out = MLIB_Neg(f16In, F16);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC16(-0.25) = 0xA000
  f16Out = MLIB_Neg(f16In);
}
```

2.80.5 Function MLIB_Neg_FLT

2.80.5.1 Declaration

```
INLINE tFloat MLIB_Neg_FLT(register tFloat fltIn);
```

2.80.5.2 Arguments

Table 403. MLIB_Neg_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn	**input**	Input value which negative value should be returned.

2.80.5.3 Return

Negative value of input parameter.

2.80.5.4 Implementation details

The input values as well as output value is considered as single precision floating point data type.

The output of the function is defined by the following simple equation:

$$fltOut = -fltIn$$

Equation MLIB_Neg_FLT_Eq1

Note: *This function is implemented as inline, and thus is not ANSI-C compliant.*

The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.

2.80.5.5 Code Example

```
#include "mlib.h"

tFloat fltIn;
tFloat fltOut;

void main(void)
{
  // input value = 0.25
  fltIn = (tFloat)0.25;

  // output should be (-0.25)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  fltOut = MLIB_Neg_FLT(fltIn);

  // output should be (-0.25)
  fltOut = MLIB_Neg(fltIn, FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output should be (-0.25)
  fltOut = MLIB_Neg(fltIn);
}
```

2.81 Function MLIB_NegSat

This function returns negative value of input parameter and saturate if necessary.

2.81.1 Description

This inline function returns the negative value of input parameter and saturates the result if necessary.

2.81.2 Re-entrancy

The function is re-entrant.

2.81.3 Function MLIB_NegSat_F32

2.81.3.1 Declaration

```
INLINE tFrac32 MLIB_NegSat_F32(register tFrac32 f32In);
```

2.81.3.2 Arguments

Table 404. MLIB_NegSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value which negative value should be returned.

2.81.3.3 Return

Negative value of input parameter.

2.81.3.4 Implementation details

The input values as well as output value is considered as 32-bit fractional data type.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32_MIN & if \quad (-f32In) < FRAC32_MIN \\ -f32In & if \quad FRAC32_MIN \le (-f32In) \le FRAC32_MAX \\ FRAC32_MAX & if \quad (-f32In) > FRAC32_MAX \end{cases}$$

Equation MLIB_NegSat_F32_Eq1

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Note: *This function is implemented as inline assembly, and thus is not ANSI-C compliant.*

2.81.3.5 Code Example

```
#include "mlib.h"

tFrac32 f32In;
tFrac32 f32Out;

void main(void)
{
  // input value = 0.25
  f32In = FRAC32(0.25);

  // output should be FRAC32(-0.25) = 0xA0000000
  f32Out = MLIB_NegSat_F32(f32In);

  // output should be FRAC32(-0.25) = 0xA0000000
  f32Out = MLIB_NegSat(f32In, F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC32(-0.25) = 0xA0000000
  f32Out = MLIB_NegSat(f32In);
}
```

2.81.4 Function MLIB_NegSat_F16

2.81.4.1 Declaration

```
INLINE tFrac16 MLIB_NegSat_F16(register tFrac16 f16In);
```

2.81.4.2 Arguments

Table 405. MLIB_NegSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	Input value which negative value should be returned.

2.81.4.3 Return

Negative value of input parameter.

2.81.4.4 Implementation details

The input values as well as output value is considered as 16-bit fractional data type.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16_MIN & if \quad (-f16In) < FRAC16_MIN \\ -f16In & if \quad FRAC16_MIN \le (-f16In) \le FRAC16_MAX \\ FRAC16_MAX & if \quad (-f16In) > FRAC16_MAX \end{cases}$$

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Equation MLIB_NegSat_F16_Eq1

Note: *This function is implemented as inline assembly, and thus is not ANSI-C compliant.*

2.81.4.5 Code Example

```
#include "mlib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
{
  // input value = 0.25
  f16In = FRAC16(0.25);

  // output should be FRAC16(-0.25) = 0xA000
  f16Out = MLIB_NegSat_F16(f16In);

  // output should be FRAC16(-0.25) = 0xA000
  f16Out = MLIB_NegSat(f16In, F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC16(-0.25) = 0xA000
  f16Out = MLIB_NegSat(f16In);
}
```

2.82 Function MLIB_Norm

This function returns the number of left shifts needed to normalize the input parameter.

2.82.1 Description

The function returns the number of left shifts needed to remove redundant leading sign bits.

2.82.2 Re-entrancy

The function is re-entrant.

2.82.3 Function MLIB_Norm_F32

2.82.3.1 Declaration

```
INLINE tU16 MLIB_Norm_F32(register tFrac32 f32In);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.82.3.2 Arguments

Table 406. MLIB_Norm_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	The first value to be normalized.

2.82.3.3 Return

The number of left shift needed to normalize the argument.

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.82.3.4 Code Example

```
#include "mlib.h"

tFrac32 f32In;
tU16 u16Out;

void main(void)
{
    // first input = 0.00005
    f32In = FRAC32(0.00005);

    // output should be 14
    u16Out = MLIB_Norm_F32(f32In);

    // output should be 14
    u16Out = MLIB_Norm(f32In,F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 14
    u16Out = MLIB_Norm(f32In);
}
```

2.82.4 Function MLIB_Norm_F16

2.82.4.1 Declaration

```
INLINE tU16 MLIB_Norm_F16(register tFrac16 f16In);
```

2.82.4.2 Arguments

Table 407. MLIB_Norm_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	The first value to be normalized.

2.82.4.3 Return

The number of left shift needed to normalize the argument.

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.82.4.4 Code Example

```
#include "mlib.h"

tFrac16 f16In;
tU16 u16Out;

void main(void)
{
    // first input = 0.00005
    f16In = FRAC16(0.00005);

    // output should be 14
    u16Out = MLIB_Norm_F16(f16In);

    // output should be 14
    u16Out = MLIB_Norm(f16In,F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 14
    u16Out = MLIB_Norm(f16In);
}
```

2.83 Function MLIB_RndSat_F16F32

This function rounds the input value to the nearest saturated value in the output format.

2.83.1 Declaration

```
INLINE tFrac16 MLIB_RndSat_F16F32(register tFrac32 f32In);
```

2.83.2 Arguments

Table 408. MLIB_RndSat_F16F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value.

2.83.3 Return

Rounded saturated value.

2.83.4 Description

This inline function rounds the input value to the nearest saturated value in the output format. The input value is considered as 32-bit fractional data type and output value is considered as 16-bit fractional data type.

2.83.5 Re-entrancy

The function is re-entrant.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.83.6 Code Example

```
#include "mlib.h"

tFrac32 f32In;
tFrac16 f16Out;

void main(void)
{
    // input value = 0.25 = 0x2000 0000
    f32In = FRAC32(0.25);

    // output should be FRAC16(0.25) = 0x2000
    f16Out = MLIB_RndSat_F16F32(f32In);
}
```

2.84 Function MLIB_Round

The function rounds the input and saturates.

2.84.1 Description

The function rounds the first input argument to the nearest value (round half up). The number of trailing zeros in the rounded result is equal to the second input argument. The result is saturated to the fractional range.

2.84.2 Re-entrancy

The function is re-entrant.

2.84.3 Function MLIB_Round_F32

2.84.3.1 Declaration

```
INLINE tFrac32 MLIB_Round_F32(register tFrac32 f32In1, register
tU16 u16In2);
```

2.84.3.2 Arguments

Table 409. MLIB_Round_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	The value to be rounded.
register tU16	u16In2	**input**	The number of trailing zeros in the rounded result.

2.84.3.3 Return

Rounded 32-bit fractional value.

Note: *The second input argument must not exceed 30 for positive and 31 for negative f32In1, respectively, otherwise the result is undefined. This function is implemented as inline and thus is not ANSI-C compliant.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.84.3.4 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32Out;
tU16 u16In2;

void main(void)
{
    // Example no. 1
    // first input = 0.25
    f32In1 = FRAC32(0.25);
    // second input = 29
    u16In2 = (tU16)29;

    // output should be 0x20000000 ~ FRAC32(0.25)
    f32Out = MLIB_Round_F32(f32In1,u16In2);

    // output should be 0x20000000 ~ FRAC32(0.25)
    f32Out = MLIB_Round(f32In1,u16In2,F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x20000000 ~ FRAC32(0.25)
    f32Out = MLIB_Round(f32In1,u16In2);


    // Example no. 2
    // first input = 0.375
    f32In1 = FRAC32(0.375);
    // second input = 29
    u16In2 = (tU16)29;

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = MLIB_Round_F32(f32In1,u16In2);


    // Example no. 3
    // first input = -0.375
    f32In1 = FRAC32(-0.375);
    // second input = 29
    u16In2 = (tU16)29;

    // output should be 0xE0000000 ~ FRAC32(-0.25)
    f32Out = MLIB_Round_F32(f32In1,u16In2);
}
```

2.84.4 Function MLIB_Round_F16

2.84.4.1 Declaration

```
INLINE tFrac16 MLIB_Round_F16(register tFrac16 f16In1, register
tU16 u16In2);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.84.4.2 Arguments

Table 410. MLIB_Round_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	The value to be rounded.
register tU16	u16In2	**input**	The number of trailing zeros in the rounded result.

2.84.4.3 Return

Rounded 16-bit fractional value.

Note: *The second input argument must not exceed 14 for positive and 15 for negative f16In1, respectively, otherwise the result is undefined. This function is implemented as inline and thus is not ANSI-C compliant.*

2.84.4.4 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16Out;
tU16 u16In2;

void main(void)
{
    // Example no. 1
    // first input = 0.25
    f16In1 = FRAC16(0.25);
    // second input = 13
    u16In2 = (tU16)13;

    // output should be 0x2000 ~ FRAC16(0.25)
    f16Out = MLIB_Round_F16(f16In1,u16In2);

    // output should be 0x2000 ~ FRAC16(0.25)
    f16Out = MLIB_Round(f16In1,u16In2);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x2000 ~ FRAC16(0.25)
    f16Out = MLIB_Round(f16In1,u16In2);


    // Example no. 2
    // first input = 0.375
    f16In1 = FRAC16(0.375);
    // second input = 13
    u16In2 = (tU16)13;

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = MLIB_Round_F16(f16In1,u16In2);


    // Example no. 3
    // first input = -0.375
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    f16In1 = FRAC16(-0.375);
    // second input = 13
    u16In2 = (tU16)13;

    // output should be 0xE000 ~ FRAC16(-0.25)
    f16Out = MLIB_Round_F16(f16In1,u16In2);
}
```

2.85 Function MLIB_ShBi

This function shifts the first argument to left or right by number defined by second argument.

2.85.1 Description

This function shifts the first parameter by the amount specified in the second parameter. Positive values of the second argument correspond to the left shift, negative values to the right shift. The result of the left shift may overflow.

2.85.2 Re-entrancy

The function is re-entrant.

2.85.3 Function MLIB_ShBi_F32

2.85.3.1 Declaration

```
INLINE tFrac32 MLIB_ShBi_F32(register tFrac32 f32In1, register
tS16 s16In2);
```

2.85.3.2 Arguments

Table 411. MLIB_ShBi_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	First value to be shift.
register tS16	s16In2	**input**	The shift amount value.

2.85.3.3 Return

32-bit fractional value shifted to left or right by the shift amount. The bits beyond the 32-bit boundary are discarded.

Note: *The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range -31...31. Otherwise the result of the function is undefined. This function is implemented as inline, and thus is not ANSI-C compliant.*

2.85.3.4 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32Out;
tS16 s16In2;

void main(void)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
{
    // first input = 0.25
    f32In1 = FRAC32(0.25);
    // second input = -1
    s16In2 = (tS16)-1;

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShBi_F32(f32In1, s16In2);

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShBi(f32In1, s16In2, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShBi(f32In1, s16In2);
}
```

2.85.4 Function MLIB_ShBi_F16

2.85.4.1 Declaration

```
INLINE tFrac16 MLIB_ShBi_F16(register tFrac16 f16In1, register
tS16 s16In2);
```

2.85.4.2 Arguments

Table 412. MLIB_ShBi_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	First value to be left shift.
register tS16	s16In2	**input**	The shift amount value.

2.85.4.3 Return

16-bit fractional value shifted to left or right by the shift amount. The bits beyond the 16-bit boundary are discarded.

Note: *The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range -15...15. Otherwise the result of the function is undefined. This function is implemented as inline, and thus is not ANSI-C compliant.*

2.85.4.4 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16Out;
tS16 s16In2;

void main(void)
{
    // first input = 0.25
    f16In1 = FRAC16(0.25);
    // second input = -1
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
    s16In2 = (tS16)-1;

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShBi_F16(f16In1, s16In2);

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShBi(f16In1, s16In2, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShBi(f16In1, s16In2);
}
```

2.86 Function MLIB_ShBiSat

This function shifts the first argument to left or right by number defined by second argument and saturate if necessary.

2.86.1 Description

This function shifts the first parameter by the amount specified in the second parameter. Positive values of the second argument correspond to the left shift, negative values to the right shift. The result of the left shift is saturated if necessary.

2.86.2 Re-entrancy

The function is re-entrant.

2.86.3 Function MLIB_ShBiSat_F32

2.86.3.1 Declaration

```
INLINE tFrac32 MLIB_ShBiSat_F32(register tFrac32 f32In1, register
tS16 s16In2);
```

2.86.3.2 Arguments

Table 413. MLIB_ShBiSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	First value to be shift.
register tS16	s16In2	**input**	The shift amount value.

2.86.3.3 Return

32-bit fractional value shifted to left or right by the shift amount. The bits beyond the 32-bit boundary are discarded.

Note: *The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range -31...31. Otherwise the result of the function is undefined. This function is implemented as inline, and thus is not ANSI-C compliant.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.86.3.4 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32Out;
tS16 s16In2;

void main(void)
{
    // first input = 0.25
    f32In1 = FRAC32(0.25);
    // second input = -1
    s16In2 = (tS16)-1;

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShBiSat_F32(f32In1, s16In2);

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShBiSat(f32In1, s16In2, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShBiSat(f32In1, s16In2);
}
```

2.86.4 Function MLIB_ShBiSat_F16

2.86.4.1 Declaration

```
INLINE tFrac16 MLIB_ShBiSat_F16(register tFrac16 f16In1, register
tS16 s16In2);
```

2.86.4.2 Arguments

Table 414. MLIB_ShBiSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	First value to be left shift.
register tS16	s16In2	**input**	The shift amount value.

2.86.4.3 Return

16-bit fractional value shifted to left or right by the shift amount. The bits beyond the 16-bit boundary are discarded.

Note: *The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range -15...15. Otherwise the result of the function is undefined. This function is implemented as inline, and thus is not ANSI-C compliant.*

2.86.4.4 Code Example

```
#include "mlib.h"
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
tFrac16 f16In1;
tFrac16 f16Out;
tS16 s16In2;

void main(void)
{
    // first input = 0.25
    f16In1 = FRAC16(0.25);
    // second input = -1
    s16In2 = (tS16)-1;

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShBiSat_F16(f16In1, s16In2);

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShBiSat(f16In1, s16In2, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShBiSat(f16In1, s16In2);
}
```

2.87 Function MLIB_ShL

This function shifts the first parameter to left by number defined by second parameter.

2.87.1 Description

This function shifts the first argument to the left by the amount specified in the second argument. Overflow is not detected.

2.87.2 Re-entrancy

The function is re-entrant.

2.87.3 Function MLIB_ShL_F32

2.87.3.1 Declaration

```
INLINE tFrac32 MLIB_ShL_F32(register tFrac32 f32In1, register
tU16 u16In2);
```

2.87.3.2 Arguments

Table 415. MLIB_ShL_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	First value to be left shift.
register tU16	u16In2	**input**	The shift amount value.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.87.3.3 Return

32-bit fractional value shifted to left by the shift amount. The bits beyond the 32-bit boundary are discarded.

Note: *The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range 0...31. Otherwise the result of the function is undefined. This function is implemented as inline, and thus is not ANSI-C compliant.*

2.87.3.4 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.25
    f32In1 = FRAC32(0.25);
    // second input = 1
    u16In2 = (tU16)1;

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = MLIB_ShL_F32(f32In1, u16In2);

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = MLIB_ShL(f32In1, u16In2, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = MLIB_ShL(f32In1, u16In2);
}
```

2.87.4 Function MLIB_ShL_F16

2.87.4.1 Declaration

```
INLINE tFrac16 MLIB_ShL_F16(register tFrac16 f16In1, register
tU16 u16In2);
```

2.87.4.2 Arguments

Table 416. MLIB_ShL_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	First value to be left shift.
register tU16	u16In2	**input**	The shift amount value.

2.87.4.3 Return

16-bit fractional value shifted to left by the shift amount. The bits beyond the 16-bit boundary are discarded.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Note: *The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range 0...15. Otherwise the result of the function is undefined. This function is implemented as inline, and thus is not ANSI-C compliant.*

2.87.4.4 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.25
    f16In1 = FRAC16(0.25);
    // second input = 1
    u16In2 = (tU16)1;

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = MLIB_ShL_F16(f16In1, u16In2);

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = MLIB_ShL(f16In1, u16In2, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = MLIB_ShL(f16In1, u16In2);
}
```

2.88 Function MLIB_ShLSat

This function shifts the first parameter to left by number defined by second parameter and saturate if necessary.

2.88.1 Description

This function shifts the first argument to the left by the amount specified in the second argument. The result is saturated.

2.88.2 Re-entrancy

The function is re-entrant.

2.88.3 Function MLIB_ShLSat_F32

2.88.3.1 Declaration

```
INLINE tFrac32 MLIB_ShLSat_F32(register tFrac32 f32In1, register
tU16 u16In2);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.88.3.2 Arguments

Table 417. MLIB_ShLSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	First value to be left shift.
register tU16	u16In2	**input**	The shift amount value.

2.88.3.3 Return

32-bit fractional value shifted to left by the shift amount. The bits beyond the 32-bit boundary are discarded.

Note: *The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range 0...31. Otherwise the result of the function is undefined. This function is implemented as inline assembly, and thus is not ANSI-C compliant.*

2.88.3.4 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.25
    f32In1 = FRAC32(0.25);
    // second input = 1
    u16In2 = (tU16)1;

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = MLIB_ShLSat_F32(f32In1, u16In2);

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = MLIB_ShLSat(f32In1, u16In2, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = MLIB_ShLSat(f32In1, u16In2);
}
```

2.88.4 Function MLIB_ShLSat_F16

2.88.4.1 Declaration

```
INLINE tFrac16 MLIB_ShLSat_F16(register tFrac16 f16In1, register
tU16 u16In2);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.88.4.2 Arguments

Table 418. MLIB_ShLSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	First value to be left shift.
register tU16	u16In2	**input**	The shift amount value.

2.88.4.3 Return

16-bit fractional value shifted to left by the shift amount. The bits beyond the 16-bit boundary are discarded.

Note: *The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range 0...15. Otherwise the result of the function is undefined. This function is implemented as inline assembly, and thus is not ANSI-C compliant.*

2.88.4.4 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.25
    f16In1 = FRAC16(0.25);
    // second input = 1
    u16In2 = (tU16)1;

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = MLIB_ShLSat_F16(f16In1, u16In2);

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = MLIB_ShLSat(f16In1, u16In2, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = MLIB_ShLSat(f16In1, u16In2);
}
```

2.89 Function MLIB_ShR

This function shifts the first parameter to right by number defined by second parameter.

2.89.1 Description

This function shifts the first argument to the right by the amount specified in the second argument.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.89.2 Re-entrancy

The function is re-entrant.

2.89.3 Function MLIB_ShR_F32

2.89.3.1 Declaration

```
INLINE tFrac32 MLIB_ShR_F32(register tFrac32 f32In1, register
tU16 u16In2);
```

2.89.3.2 Arguments

Table 419. MLIB_ShR_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	First value to be right shift.
register tU16	u16In2	**input**	The shift amount value.

2.89.3.3 Return

32-bit fractional value shifted right by the shift amount. The bits beyond the 32-bit boundary of the result are discarded.

Note: *The shift amount cannot exceed in magnitude the bit-width of the shifted value, that means it must be within the range 0...31. Otherwise the result of the function is undefined. This function is implemented as inline, and thus is not ANSI-C compliant.*

2.89.3.4 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.25
    f32In1 = FRAC32(0.25);
    // second input = 1
    u16In2 = (tU16)1;

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShR_F32(f32In1, u16In2);

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShR(f32In1, u16In2, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShR(f32In1, u16In2);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.89.4 Function MLIB_ShR_F16

2.89.4.1 Declaration

```
INLINE tFrac16 MLIB_ShR_F16(register tFrac16 f16In1, register
tU16 u16In2);
```

2.89.4.2 Arguments

Table 420. MLIB_ShR_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	First value to be right shift.
register tU16	u16In2	**input**	The shift amount value.

2.89.4.3 Return

16-bit fractional value shifted right by the shift amount. The bits beyond the 16-bit boundary of the result are discarded.

Note: *The shift amount cannot exceed in magnitude the bit-width of the shifted value, that means it must be within the range 0...15. Otherwise the result of the function is undefined. This function is implemented as inline, and thus is not ANSI-C compliant.*

2.89.4.4 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.25
    f16In1 = FRAC16(0.25);
    // second input = 1
    u16In2 = (tU16)1;

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShR_F16(f16In1, u16In2);

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShR(f16In1, u16In2, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShR(f16In1, u16In2);
}
```

2.90 Function MLIB_Sub

This function subtracts the second parameter from the first one.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.90.1 Description

The second argument is subtracted from the first one.

2.90.2 Re-entrancy

The function is re-entrant.

2.90.3 Function MLIB_Sub_F32

2.90.3.1 Declaration

```
INLINE tFrac32 MLIB_Sub_F32(register tFrac32 f32In1, register
tFrac32 f32In2);
```

2.90.3.2 Arguments

Table 421. MLIB_Sub_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Operand is a 32-bit number normalized between[-1,1).
register tFrac32	f32In2	**input**	Operand is a 32-bit number normalized between[-1,1).

2.90.3.3 Return

The subtraction of the second argument from the first argument.

2.90.3.4 Implementation details

The input values as well as output value are considered as 32-bit fractional data type. The output saturation is not implemented in this function, thus in case the subtraction of input parameters is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f32Out = f32In1 - f32In2$$

Equation MLIB_Sub_F32_Eq1

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.90.3.5 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32Out;

void main(void)
{
  // first input = 0.5
  f32In1 = FRAC32(0.5);

  // second input = 0.25
  f32In2 = FRAC32(0.25);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // output should be 0x20000000
  f32Out = MLIB_Sub_F32(f32In1,f32In2);

  // output should be 0x20000000
  f32Out = MLIB_Sub(f32In1,f32In2,F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be 0x20000000
  f32Out = MLIB_Sub(f32In1,f32In2);
}
```

2.90.4 Function MLIB_Sub_F16

2.90.4.1 Declaration

```
INLINE tFrac16 MLIB_Sub_F16(register tFrac16 f16In1, register
tFrac16 f16In2);
```

2.90.4.2 Arguments

Table 422. MLIB_Sub_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In2	**input**	Operand is a 16-bit number normalized between [-1,1).

2.90.4.3 Return

The subtraction of the second argument from the first argument.

2.90.4.4 Implementation details

The input values as well as output value are considered as 16-bit fractional data type. The output saturation is not implemented in this function, thus in case the subtraction of input parameters is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f16Out = f16In1 - f16In2$$

Equation MLIB_Sub_F16_Eq1

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.90.4.5 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16Out;
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
void main(void)
{
  // first input = 0.5
  f16In1 = FRAC16(0.5);

  // second input = 0.25
  f16In2 = FRAC16(0.25);

  // output should be 0x2000
  f16Out = MLIB_Sub_F16(f16In1,f16In2);

  // output should be 0x2000
  f16Out = MLIB_Sub(f16In1,f16In2,F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be 0x20000000
  f16Out = MLIB_Sub(f16In1,f16In2);
}
```

2.90.5 Function MLIB_Sub_FLT

2.90.5.1 Declaration

```
INLINE tFloat MLIB_Sub_FLT(register tFloat fltIn1, register
tFloat fltIn2);
```

2.90.5.2 Arguments

Table 423. MLIB_Sub_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn1	**input**	Operand is a single precision floating point number.
register tFloat	fltIn2	**input**	Operand is a single precision floating point number.

2.90.5.3 Return

The subtraction of the second argument from the first argument.

2.90.5.4 Implementation details

The input value as well as output value is considered as single precision floating point data type.

The output of the function is defined by the following simple equation:

$$fltOut = fltIn1 - fltIn2$$

Equation MLIB_Sub_FLT_Eq1

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

This function is implemented as inline and thus is not ANSI-C compliant.

2.90.5.5 Code Example

```
#include "mlib.h"

tFloat fltIn1;
tFloat fltIn2;
tFloat fltOut;

void main(void)
{
  // first input = 50.5
  fltIn1 = (tFloat)50.5;

  // second input = 25.25
  fltIn2 = (tFloat)25.25;

  // output should be 25.25
  fltOut = MLIB_Sub_FLT(fltIn1,fltIn2);

  // output should be 25.25
  fltOut = MLIB_Sub(fltIn1,fltIn2,FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output should be 25.25
  fltOut = MLIB_Sub(fltIn1,fltIn2);
}
```

2.91 Function MLIB_SubSat

This function subtracts the second parameter from the first one and saturate if necessary.

2.91.1 Description

The second argument is subtracted from the first one. The result is saturated if necessary

2.91.2 Re-entrancy

The function is re-entrant.

2.91.3 Function MLIB_SubSat_F32

2.91.3.1 Declaration

```
INLINE tFrac32 MLIB_SubSat_F32(register tFrac32 f32In1, register
tFrac32 f32In2);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.91.3.2 Arguments

Table 424. MLIB_SubSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Operand is a 32-bit number normalized between [-1,1).
register tFrac32	f32In2	**input**	Operand is a 32-bit number normalized between [-1,1).

2.91.3.3 Return

The subtraction of the second argument from the first argument.

2.91.3.4 Implementation details

The input values as well as output value are considered as 32-bit fractional data type.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32_MIN & if \quad (f32In1 - f32In2) < FRAC32_MIN \\ f32In1 - f32In2 & if \quad FRAC32_MIN \le (f32In1 - f32In2) \le FRAC32_MAX \\ FRAC32_MAX & if \quad (f32In1 - f32In2) > FRAC32_MAX \end{cases}$$

Equation MLIB_SubSat_F32_Eq1

Note: *This function is implemented as inline assembly and thus is not ANSI-C compliant.*

2.91.3.5 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32Out;

void main(void)
{
  // first input = 0.5
  f32In1 = FRAC32(0.5);

  // second input = 0.25
  f32In2 = FRAC32(0.25);

  // output should be 0x20000000
  f32Out = MLIB_SubSat_F32(f32In1,f32In2);

  // output should be 0x20000000
  f32Out = MLIB_SubSat(f32In1,f32In2,F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be 0x20000000
  f32Out = MLIB_SubSat(f32In1,f32In2);
}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.91.4 Function MLIB_SubSat_F16

2.91.4.1 Declaration

```
INLINE tFrac16 MLIB_SubSat_F16(register tFrac16 f16In1, register
tFrac16 f16In2);
```

2.91.4.2 Arguments

Table 425. MLIB_SubSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In2	**input**	Operand is a 16-bit number normalized between [-1,1).

2.91.4.3 Return

The subtraction of the second argument from the first argument.

2.91.4.4 Implementation details

The input values as well as output value are considered as 16-bit fractional data type.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16_MIN & if \quad (f16In1 - f16In2) < FRAC16_MIN \\ f16In1 - f16In2 & if \quad FRAC16_MIN \le (f16In1 - f16In2) \le FRAC16_MAX \\ FRAC16_MAX & if \quad (f16In1 - f16In2) > FRAC16_MAX \end{cases}$$

Equation MLIB_SubSat_F16_Eq1

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.91.4.5 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16Out;

void main(void)
{
  // first input = 0.5
  f16In1 = FRAC16(0.5);

  // second input = 0.25
  f16In2 = FRAC16(0.25);

  // output should be 0x2000
  f16Out = MLIB_SubSat_F16(f16In1,f16In2);

  // output should be 0x2000
  f16Out = MLIB_SubSat(f16In1,f16In2,F16);

  // ##########################################################
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##########################################################

  // output should be 0x2000
  f16Out = MLIB_SubSat(f16In1,f16In2);
}
```

2.92 Function MLIB_VAdd

This function returns vector sum of two input vectors.

2.92.1 Description

This inline function returns the vector sum of two input vectors.

If the input vectors are defined as

$$\vec{u} = (u_1, u_2)$$
$$\vec{v} = (v_1, v_2)$$

Equation MLIB_VAdd_Eq1

then the output of the function is calculated by the following simple equations:

$$\vec{u} + \vec{v} = (u_1 + v_1, u_2 + v_2)$$

Equation MLIB_VAdd_Eq2

2.92.2 Re-entrancy

The function is re-entrant.

2.92.3 Function MLIB_VAdd_F32

2.92.3.1 Declaration

```
INLINE void MLIB_VAdd_F32(SWLIBS_2Syst_F32 *const f32Out, const
SWLIBS_2Syst_F32 *const f32In1, const SWLIBS_2Syst_F32 *const
f32In2);
```

2.92.3.2 Arguments

Table 426. MLIB_VAdd_F32 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F32 *const	f32Out	**output**	Sum of two input vectors.
const SWLIBS_2Syst_F32 *const	f32In1	**input**	First vector to be added.
const SWLIBS_2Syst_F32 *const	f32In2	**input**	Second vector to be added.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.92.3.3 Implementation details

The input values as well as output value is considered as vectors defined by two 32-bit fractional coordinates stored in SWLIBS_2Syst_F32 structure.

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.92.3.4 Code Example:

```
#include "mlib.h"

SWLIBS_2Syst_F32 f32In1, f32In2;
SWLIBS_2Syst_F32 f32Out;

void main(void)
{
  // input vector 1 = (0.25,0.25)
  f32In1.f32Arg1 = FRAC32(0.25);
  f32In1.f32Arg2 = FRAC32(0.25);
  // input vector 2 = (0.25,0.25)
  f32In2.f32Arg1 = FRAC32(0.25);
  f32In2.f32Arg2 = FRAC32(0.25);

  // Both f32Out.f32Arg1 and f32Out.f32Arg2 should be FRAC32(0.5) = 0x40000000
  MLIB_VAdd_F32(&f32Out, &f32In1, &f32In2);

  // Both f32Out.f32Arg1 and f32Out.f32Arg2 should be FRAC32(0.5) = 0x40000000
  MLIB_VAdd(&f32Out, &f32In1, &f32In2, F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // Both f32Out.f32Arg1 and f32Out.f32Arg2 should be FRAC32(0.5) = 0x40000000
  MLIB_VAdd(&f32Out, &f32In1, &f32In2);
}
```

2.92.4 Function MLIB_VAdd_F16

2.92.4.1 Declaration

```
INLINE void MLIB_VAdd_F16(SWLIBS_2Syst_F16 *const f16Out, const
SWLIBS_2Syst_F16 *const f16In1, const SWLIBS_2Syst_F16 *const
f16In2);
```

2.92.4.2 Arguments

Table 427. MLIB_VAdd_F16 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *const	f16Out	**output**	Sum of two input vectors.
const SWLIBS_2Syst_F16 *const	f16In1	**input**	First vector to be added.
const SWLIBS_2Syst_F16 *const	f16In2	**input**	Second vector to be added.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.92.4.3 Implementation details

The input values as well as output value is considered as vectors defined by two 16-bit fractional coordinates stored in SWLIBS_2Syst_F16 structure.

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.92.4.4 Code Example:

```
#include "mlib.h"

SWLIBS_2Syst_F16 f16In1, f16In2;
SWLIBS_2Syst_F16 f16Out;

void main(void)
{
  // input vector 1 = (0.25,0.25)
  f16In1.f16Arg1 = FRAC16(0.25);
  f16In1.f16Arg2 = FRAC16(0.25);
  // input vector 2 = (0.25,0.25)
  f16In2.f16Arg1 = FRAC16(0.25);
  f16In2.f16Arg2 = FRAC16(0.25);

  // Both f16Out.f16Arg1 and f16Out.f16Arg2 should be FRAC16(0.5) = 0x4000
  MLIB_VAdd_F16(&f16Out, &f16In1, &f16In2);

  // Both f16Out.f16Arg1 and f16Out.f16Arg2 should be FRAC16(0.5) = 0x4000
  MLIB_VAdd(&f16Out, &f16In1, &f16In2, F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // Both f16Out.f16Arg1 and f16Out.f16Arg2 should be FRAC16(0.5) = 0x4000
  MLIB_VAdd(&f16Out, &f16In1, &f16In2);
}
```

2.92.5 Function MLIB_VAdd_FLT

2.92.5.1 Declaration

```
INLINE void MLIB_VAdd_FLT(SWLIBS_2Syst_FLT *const fltOut, const
SWLIBS_2Syst_FLT *const fltIn1, const SWLIBS_2Syst_FLT *const
fltIn2);
```

2.92.5.2 Arguments

Table 428. MLIB_VAdd_FLT arguments

Type	Name	Direction	Description
SWLIBS_2Syst_FLT *const	fltOut	**output**	Sum of two input vectors.
const SWLIBS_2Syst_FLT *const	fltIn1	**input**	First vector to be added.
const SWLIBS_2Syst_FLT *const	fltIn2	**input**	Second vector to be added.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.92.5.3 Implementation details

The input values as well as output value is considered as vectors defined by two single precision floating-point coordinates stored in SWLIBS_2Syst_FLT structure.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

This function is implemented as inline and thus is not ANSI-C compliant.

2.92.5.4 Code Example

```
#include "mlib.h"

SWLIBS_2Syst_FLT fltIn1, fltIn2;
SWLIBS_2Syst_FLT fltOut;

void main(void)
{
  // input vector 1 = (0.25,0.25)
  fltIn1.fltArg1 = 0.25;
  fltIn1.fltArg2 = 0.25;
  // input vector 2 = (0.25,0.25)
  fltIn2.fltArg1 = 0.25;
  fltIn2.fltArg2 = 0.25;

  // Both fltOut.fltArg1 and fltOut.fltArg2 should be 0.5
  MLIB_VAdd_FLT(&fltOut, &fltIn1, &fltIn2);

  // Both fltOut.fltArg1 and fltOut.fltArg2 should be 0.5
  MLIB_VAdd(&fltOut, &fltIn1, &fltIn2, FLT);

  // ##########################################################
  // Available only if single precision floating-point
  // implementation selected as default
  // ##########################################################

  // Both fltOut.fltArg1 and fltOut.fltArg2 should be 0.5
  MLIB_VAdd(&fltOut, &fltIn1, &fltIn2);
}
```

2.93 Function MLIB_VMac

This function implements the vector multiply accumulate function.

2.93.1 Description

This inline function returns the dot product of four input values.

2.93.2 Re-entrancy

The function is re-entrant.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.93.3 Function MLIB_VMac_F32

2.93.3.1 Declaration

```
INLINE tFrac32 MLIB_VMac_F32(register tFrac32 f32In1, register
tFrac32 f32In2, register tFrac32 f32In3, register tFrac32
f32In4);
```

2.93.3.2 Arguments

Table 429. MLIB_VMac_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	First input value to first multiplication.
register tFrac32	f32In2	**input**	Second input value to first multiplication.
register tFrac32	f32In3	**input**	First input value to second multiplication.
register tFrac32	f32In4	**input**	Second input value to second multiplication.

2.93.3.3 Return

Vector multiplied input values with addition.

2.93.3.4 Implementation details

The input values as well as output value is considered as 32-bit fractional values. The output saturation is implemented only for multiplication result inside this function. The output saturation is not implemented for addition in this function, thus in case the add of input values is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f32Out = (f32In1 \cdot f32In2) + (f32In3 \cdot f32In4)$$

Equation MLIB_VMac_F32_Eq1

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.93.3.5 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32In3;
tFrac32 f32In4;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.25
  f32In1 = FRAC32(0.25);

  // input2 value = 0.15
  f32In2 = FRAC32(0.15);

  // input3 value = 0.35
  f32In3 = FRAC32(0.35);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // input4 value = 0.45
  f32In4 = FRAC32(0.45);

  // output should be FRAC32(0.195) = 0x18F5C28F
  f32Out = MLIB_VMac_F32(f32In1,f32In2,f32In3,f32In4);

  // output should be FRAC32(0.195) = 0x18F5C28F
  f32Out = MLIB_VMac(f32In1,f32In2,f32In3,f32In4, F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC32(0.195) = 0x18F5C28F
  f32Out = MLIB_VMac(f32In1,f32In2,f32In3,f32In4);
}
```

2.93.4 Function MLIB_VMac_F32F16F16

2.93.4.1 Declaration

```
INLINE tFrac32 MLIB_VMac_F32F16F16(register tFrac16 f16In1,
register tFrac16 f16In2, register tFrac16 f16In3, register
tFrac16 f16In4);
```

2.93.4.2 Arguments

Table 430. MLIB_VMac_F32F16F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	First input value to first multiplication.
register tFrac16	f16In2	**input**	Second input value to first multiplication.
register tFrac16	f16In3	**input**	First input value to second multiplication.
register tFrac16	f16In4	**input**	Second input value to second multiplication.

2.93.4.3 Return

Vector multiplied input values with addition.

2.93.4.4 Implementation details

The input values are considered as 16-bit fractional values and the output value is considered as 32-bit fractional value. The output saturation is implemented only for multiplication result inside this function. The output saturation is not implemented for addition in this function, thus in case the add of input values is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f32Out = (f16In1 \cdot f16In2) + (f16In3 \cdot f16In4)$$

Equation MLIB_VMac_F32F16F16_Eq1

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.93.4.5 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac16 f16In4;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.25
  f16In1 = FRAC16(0.25);

  // input2 value = 0.15
  f16In2 = FRAC16(0.15);

  // input3 value = 0.35
  f16In3 = FRAC16(0.35);

  // input4 value = 0.45
  f16In4 = FRAC16(0.45);

  // output should be FRAC32(0.195) = 0x18F5C28F
  f32Out = MLIB_VMac_F32F16F16(f16In1,f16In2,f16In3,f16In4);

  // output should be FRAC32(0.195) = 0x18F5C28F
  f32Out = MLIB_VMac(f16In1,f16In2,f16In3,f16In4, F32F16F16);
}
```

2.93.5 Function MLIB_VMac_F16

2.93.5.1 Declaration

```
INLINE tFrac16 MLIB_VMac_F16(register tFrac16 f16In1, register
tFrac16 f16In2, register tFrac16 f16In3, register tFrac16
f16In4);
```

2.93.5.2 Arguments

Table 431. MLIB_VMac_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	First input value to first multiplication.
register tFrac16	f16In2	**input**	Second input value to first multiplication.
register tFrac16	f16In3	**input**	First input value to second multiplication.
register tFrac16	f16In4	**input**	Second input value to second multiplication.

2.93.5.3 Return

Vector multiplied input values with addition.

2.93.5.4 Implementation details

The input values as well as output value is considered as 16-bit fractional values. The output saturation is implemented only for multiplication result inside this function. The

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

output saturation is not implemented for addition in this function, thus in case the add of input values is outside the [-1, 1) interval, the output value will overflow.

The output of the function is defined by the following simple equation:

$$f16Out = (f16In1 \cdot f16In2) + (f16In3 \cdot f16In4)$$

Equation MLIB_VMac_F16_Eq1

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.93.5.5 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac16 f16In4;
tFrac16 f16Out;

void main(void)
{
  // input1 value = 0.25
  f16In1 = FRAC16(0.25);

  // input2 value = 0.15
  f16In2 = FRAC16(0.15);

  // input3 value = 0.35
  f16In3 = FRAC16(0.35);

  // input4 value = 0.45
  f16In4 = FRAC16(0.45);

  // output should be FRAC16(0.195) = 0x18F5
  f16Out = MLIB_VMac_F16(f16In1,f16In2,f16In3,f16In4);

  // output should be FRAC16(0.195) = 0x18F5
  f16Out = MLIB_VMac(f16In1,f16In2,f16In3,f16In4, F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC16(0.195) = 0x18F5
  f16Out = MLIB_VMac(f16In1,f16In2,f16In3,f16In4);
}
```

2.93.6 Function MLIB_VMac_FLT

2.93.6.1 Declaration

```
INLINE tFloat MLIB_VMac_FLT(register tFloat fltIn1, register
tFloat fltIn2, register tFloat fltIn3, register tFloat fltIn4);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.93.6.2 Arguments

Table 432. MLIB_VMac_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn1	**input**	First input value to first multiplication.
register tFloat	fltIn2	**input**	Second input value to first multiplication.
register tFloat	fltIn3	**input**	First input value to second multiplication.
register tFloat	fltIn4	**input**	Second input value to second multiplication.

2.93.6.3 Return

Vector multiplied input values with addition.

2.93.6.4 Implementation details

The input values as well as output value is considered as single precision floating point values.

The output of the function is defined by the following simple equation:

$$fltOut = (fltIn1 \cdot fltIn2) + (fltIn3 \cdot fltIn4)$$

Equation MLIB_VMac_FLT_Eq1

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

This function is implemented as inline assembly and thus is not ANSI-C compliant.

2.93.6.5 Code Example

```
#include "mlib.h"

tFloat fltIn1;
tFloat fltIn2;
tFloat fltIn3;
tFloat fltIn4;
tFloat fltOut;

void main(void)
{
  // input1 value = 0.25
  fltIn1 = (tFloat)0.25;

  // input2 value = 0.15
  fltIn2 = (tFloat)0.15;

  // input3 value = 0.35
  fltIn3 = (tFloat)0.35;

  // input4 value = 0.45
  fltIn4 = (tFloat)0.45;

  // output should be 0.195
  fltOut = MLIB_VMac_FLT(fltIn1,fltIn2,fltIn3,fltIn4);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // output should be 0.195
  fltOut = MLIB_VMac(fltIn1,fltIn2,fltIn3,fltIn4, FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output should be 0.195
  fltOut = MLIB_VMac(fltIn1,fltIn2,fltIn3,fltIn4);
}
```

2.94 Function MLIB_VScale

This function returns input vector scaled by second input value.

2.94.1 Description

This inline function returns a vector multiplied by a scalar.

If the scale is defined as α and the input vector as

$$\vec{u} = (u_1, u_2)$$

Equation MLIB_VScale_Eq1

then the output of the function is calculated by the following simple equation:

$$\alpha \cdot \vec{u} = (\alpha \cdot u_1, \alpha \cdot u_2)$$

Equation MLIB_VScale_Eq2

2.94.2 Re-entrancy

The function is re-entrant.

2.94.3 Function MLIB_VScale_F32

2.94.3.1 Declaration

```
INLINE void MLIB_VScale_F32(SWLIBS_2Syst_F32 *const f32OutVec,
const SWLIBS_2Syst_F32 *const f32InVec, tFrac32 f32InScale);
```

2.94.3.2 Arguments

Table 433. MLIB_VScale_F32 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F32 *const	f32OutVec	**output**	Scaled vector.
const SWLIBS_2Syst_F32 *const	f32InVec	**input**	Input vector to be scaled.
tFrac32	f32InScale	**input**	Scaling coefficient.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.94.3.3 Implementation details

The first input as well as output value are considered as vectors defined by two 32-bit fractional coordinates stored in SWLIBS_2Syst_F32 structure. Input vector is scaled by second scalar input in 32-bit fractional format. If the output value should be outside the [-1,1) interval, the value is saturated.

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.94.3.4 Code Example:

```
#include "mlib.h"

SWLIBS_2Syst_F32 f32InVec;
tFrac32 f32InScale;
SWLIBS_2Syst_F32 f32Out;

void main(void)
{
  // input vector = (0.5,0.5)
  f32InVec.f32Arg1 = FRAC32(0.5);
  f32InVec.f32Arg2 = FRAC32(0.5);
  // input scale = 0.5
  f32InScale = FRAC32(0.5);

  // Both f32Out.f32Arg1 and f32Out.f32Arg2 should be FRAC32(0.25) = 0x20000000
  MLIB_VScale_F32(&f32Out, &f32InVec, f32InScale);

  // Both f32Out.f32Arg1 and f32Out.f32Arg2 should be FRAC32(0.25) = 0x20000000
  MLIB_VScale(&f32Out, &f32InVec, f32InScale, F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // Both f32Out.f32Arg1 and f32Out.f32Arg2 should be FRAC32(0.25) = 0x20000000
  MLIB_VScale(&f32Out, &f32InVec, f32InScale);
}
```

2.94.4 Function MLIB_VScale_F16

2.94.4.1 Declaration

```
INLINE void MLIB_VScale_F16(SWLIBS_2Syst_F16 *const f16OutVec,
const SWLIBS_2Syst_F16 *const f16InVec, tFrac16 f16InScale);
```

2.94.4.2 Arguments

Table 434. MLIB_VScale_F16 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *const	f16OutVec	**output**	Scaled vector.
const SWLIBS_2Syst_F16 *const	f16InVec	**input**	Input vector to be scaled.
tFrac16	f16InScale	**input**	Scaling coefficient.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.94.4.3 Implementation details

The first input as well as output value are considered as vectors defined by two 16-bit fractional coordinates stored in SWLIBS_2Syst_F16 structure. Input vector is scaled by second scalar input in 16-bit fractional format. If the output value should be outside the [-1,1) interval, the value is saturated.

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.94.4.4 Code Example:

```
#include "mlib.h"

SWLIBS_2Syst_F16 f16InVec;
tFrac16 f16InScale;
SWLIBS_2Syst_F16 f16Out;

void main(void)
{
  // input vector 1 = (0.5,0.5)
  f16InVec.f16Arg1 = FRAC16(0.5);
  f16InVec.f16Arg2 = FRAC16(0.5);
  // input scale = 0.5
  f16InScale = FRAC16(0.5);

  // Both f16Out.f16Arg1 and f16Out.f16Arg2 should be FRAC16(0.25) = 0x2000
  MLIB_VScale_F16(&f16Out, &f16InVec, f16InScale);

  // Both f16Out.f16Arg1 and f16Out.f16Arg2 should be FRAC16(0.25) = 0x2000
  MLIB_VScale(&f16Out, &f16InVec, f16InScale, F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // Both f16Out.f16Arg1 and f16Out.f16Arg2 should be FRAC16(0.25) = 0x2000
  MLIB_VScale(&f16Out, &f16InVec, f16InScale);
}
```

2.94.5 Function MLIB_VScale_FLT

2.94.5.1 Declaration

```
INLINE void MLIB_VScale_FLT(SWLIBS_2Syst_FLT *const fltOutVec,
const SWLIBS_2Syst_FLT *const fltInVec, tFloat fltInScale);
```

2.94.5.2 Arguments

Table 435. MLIB_VScale_FLT arguments

Type	Name	Direction	Description
SWLIBS_2Syst_FLT *const	fltOutVec	**output**	Scaled vector.
const SWLIBS_2Syst_FLT *const	fltInVec	**input**	Input vector to be scaled.
tFloat	fltInScale	**input**	Scaling coefficient.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.94.5.3 Implementation details

The first input as well as output value are considered as vectors defined by two single precision floating-point coordinates stored in SWLIBS_2Syst_FLT structure. Input vector is scaled by second scalar input in single precision floating-point format.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

This function is implemented as inline and thus is not ANSI-C compliant.

2.94.5.4 Code Example

```
#include "mlib.h"

SWLIBS_2Syst_FLT fltInVec;
tFloat fltInScale;
SWLIBS_2Syst_FLT fltOut;

void main(void)
{
  // input vector 1 = (0.5,0.5)
  fltInVec.fltArg1 = 0.5;
  fltInVec.fltArg2 = 0.5;
  // input scale = 0.5
  fltInScale = 0.5;

  // Both fltOut.fltArg1 and fltOut.fltArg2 should be 0.25
  MLIB_VScale_FLT(&fltOut, &fltInVec, fltInScale);

  // Both fltOut.fltArg1 and fltOut.fltArg2 should be 0.25
  MLIB_VScale(&fltOut, &fltInVec, fltInScale, FLT);

  // ##############################################################
  // Available only if single precision floating-point
  // implementation selected as default
  // ##############################################################

  // Both fltOut.fltArg1 and fltOut.fltArg2 should be 0.25
  MLIB_VScale(&fltOut, &fltInVec, fltInScale);
}
```

2.95 Function MLIB_VSub

This function returns vector substract of two input vectors.

2.95.1 Description

This inline function returns the vector substract of two input vectors.

If the input vectors are defined as

$$\vec{u} = (u_1, u_2)$$
$$\vec{v} = (v_1, v_2)$$

© NXP B.V. 2023. All rights reserved.

Equation MLIB_VSub_Eq1

then the output of the function is calculated by the following simple equations:

$$\vec{u} - \vec{v} = \left(u_1 - v_1, u_2 - v_2\right)$$

Equation MLIB_VSub_Eq2

2.95.2 Re-entrancy

The function is re-entrant.

2.95.3 Function MLIB_VSub_F32

2.95.3.1 Declaration

```
INLINE void MLIB_VSub_F32(SWLIBS_2Syst_F32 *const f32Out, const
SWLIBS_2Syst_F32 *const f32In1, const SWLIBS_2Syst_F32 *const
f32In2);
```

2.95.3.2 Arguments

Table 436. MLIB_VSub_F32 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F32 *const	f32Out	**output**	Subtraction of two input vectors.
const SWLIBS_2Syst_F32 *const	f32In1	**input**	Vector representing first operand.
const SWLIBS_2Syst_F32 *const	f32In2	**input**	Vector representing second operand.

2.95.3.3 Implementation details

The input values as well as output value is considered as vectors defined by two 32-bit fractional coordinates stored in SWLIBS_2Syst_F32 structure.

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.95.3.4 Code Example:

```
#include "mlib.h"

SWLIBS_2Syst_F32 f32In1, f32In2;
SWLIBS_2Syst_F32 f32Out;

void main(void)
{
  // input vector 1 = (0.5,0.5)
  f32In1.f32Arg1 = FRAC32(0.5);
  f32In1.f32Arg2 = FRAC32(0.5);
  // input vector 2 = (0.25,0.25)
  f32In2.f32Arg1 = FRAC32(0.25);
  f32In2.f32Arg2 = FRAC32(0.25);

  // Both f32Out.f32Arg1 and f32Out.f32Arg2 should be FRAC32(0.25) = 0x20000000
  MLIB_VSub_F32(&f32Out, &f32In1, &f32In2);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // Both f32Out.f32Arg1 and f32Out.f32Arg2 should be FRAC32(0.25) = 0x20000000
  MLIB_VSub(&f32Out, &f32In1, &f32In2, F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // Both f32Out.f32Arg1 and f32Out.f32Arg2 should be FRAC32(0.25) = 0x20000000
  MLIB_VSub(&f32Out, &f32In1, &f32In2);
}
```

2.95.4 Function MLIB_VSub_F16

2.95.4.1 Declaration

```
INLINE void MLIB_VSub_F16(SWLIBS_2Syst_F16 *const f16Out, const
SWLIBS_2Syst_F16 *const f16In1, const SWLIBS_2Syst_F16 *const
f16In2);
```

2.95.4.2 Arguments

Table 437. MLIB_VSub_F16 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *const	f16Out	**output**	Subtraction of two input vectors.
const SWLIBS_2Syst_F16 *const	f16In1	**input**	Vector representing first operand.
const SWLIBS_2Syst_F16 *const	f16In2	**input**	Vector representing second operand.

2.95.4.3 Implementation details

The input values as well as output value is considered as vectors defined by two 16-bit fractional coordinates stored in SWLIBS_2Syst_F16 structure.

Note: *This function is implemented as inline and thus is not ANSI-C compliant.*

2.95.4.4 Code Example:

```
#include "mlib.h"

SWLIBS_2Syst_F16 f16In1, f16In2;
SWLIBS_2Syst_F16 f16Out;

void main(void)
{
  // input vector 1 = (0.5,0.5)
  f16In1.f16Arg1 = FRAC16(0.5);
  f16In1.f16Arg2 = FRAC16(0.5);
  // input vector 2 = (0.25,0.25)
  f16In2.f16Arg1 = FRAC16(0.25);
  f16In2.f16Arg2 = FRAC16(0.25);

  // Both f16Out.f16Arg1 and f16Out.f16Arg2 should be FRAC16(0.25) = 0x2000
  MLIB_VSub_F16(&f16Out, &f16In1, &f16In2);
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  // Both f16Out.f16Arg1 and f16Out.f16Arg2 should be FRAC16(0.25) = 0x2000
  MLIB_VSub(&f16Out, &f16In1, &f16In2, F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // Both f16Out.f16Arg1 and f16Out.f16Arg2 should be FRAC16(0.25) = 0x2000
  MLIB_VSub(&f16Out, &f16In1, &f16In2);
}
```

2.95.5 Function MLIB_VSub_FLT

2.95.5.1 Declaration

```
INLINE void MLIB_VSub_FLT(SWLIBS_2Syst_FLT *const fltOut, const
SWLIBS_2Syst_FLT *const fltIn1, const SWLIBS_2Syst_FLT *const
fltIn2);
```

2.95.5.2 Arguments

Table 438. MLIB_VSub_FLT arguments

Type	Name	Direction	Description
SWLIBS_2Syst_FLT *const	fltOut	**output**	Subtraction of two input vectors.
const SWLIBS_2Syst_FLT *const	fltIn1	**input**	Vector representing first operand.
const SWLIBS_2Syst_FLT *const	fltIn2	**input**	Vector representing second operand.

2.95.5.3 Implementation details

The input values as well as output value is considered as vectors defined by two single precision floating-point coordinates stored in SWLIBS_2Syst_FLT structure.

Note: *The function may raise floating-point exceptions (invalid operation, overflow, underflow, inexact, input denormal). The floating-point unit must be enabled in CPACR register to prevent NOCP UsageFault exception.*

This function is implemented as inline and thus is not ANSI-C compliant.

2.95.5.4 Code Example

```
#include "mlib.h"

SWLIBS_2Syst_FLT fltIn1, fltIn2;
SWLIBS_2Syst_FLT fltOut;

void main(void)
{
  // input vector 1 = (0.5,0.5)
  fltIn1.fltArg1 = 0.5;
  fltIn1.fltArg2 = 0.5;
  // input vector 2 = (0.25,0.25)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

```
  fltIn2.fltArg1 = 0.25;
  fltIn2.fltArg2 = 0.25;

  // Both fltOut.fltArg1 and fltOut.fltArg2 should be 0.25
  MLIB_VSub_FLT(&fltOut, &fltIn1, &fltIn2);

  // Both fltOut.fltArg1 and fltOut.fltArg2 should be 0.25
  MLIB_VSub(&fltOut, &fltIn1, &fltIn2, FLT);

  // ##############################################################
  // Available only if single precision floating-point
  // implementation selected as default
  // ##############################################################

  // Both fltOut.fltArg1 and fltOut.fltArg2 should be 0.25
  MLIB_VSub(&fltOut, &fltIn1, &fltIn2);
}
```

2.96 Function SWLIBS_GetVersion

This function returns the information about AMMCLIB version.

2.96.1 Declaration

```
const SWLIBS_VERSION_T * SWLIBS_GetVersion(void);
```

2.96.2 Return

The function returns the information about the version of Motor Control Library Set.

2.96.3 Description

The function returns the information about the version of Motor Control Library Set. The information are structured as follows:

- Motor Control Library Set identification code
- Motor Control Library Set version code
- Motor Control Library Set supported implementation code

2.96.4 Reentrancy

The function is reentrant.

3 Typedefs

3.1 Typedef index

Table 439. Quick typedefs reference

Type	Name	Description
`typedef double`	`tDouble`	double precision float type
`typedef float`	`tFloat`	single precision float type
`typedef tS16`	`tFrac16`	16-bit signed fractional Q1.15 type
`typedef tS32`	`tFrac32`	32-bit Q1.31 type

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 439. Quick typedefs reference*...continued*

Type	Name	Description
`typedef signed short`	`tS16`	signed 16-bit integer type
`typedef signed long`	`tS32`	signed 32-bit integer type
`typedef signed long long`	`tS64`	signed 64-bit integer type
`typedef signed char`	`tS8`	signed 8-bit integer type
`typedef unsigned short`	`tU16`	unsigned 16-bit integer type
`typedef unsigned long`	`tU32`	unsigned 32-bit integer type
`typedef unsigned long long`	`tU64`	unsigned 64-bit integer type
`typedef unsigned char`	`tU8`	unsigned 8-bit integer type

4 Enums

This section describes in details the enums available in AMMCLib.

4.1 AMCLIB_ESTIMBJINIT_RET_T

Enum type of the AMCLIB_EstimBJInit return value.

4.1.1 Source File

```
#include <AMCLIB_EstimBJ.h>
```

4.1.2 Enumerated Type Members

Table 440. AMCLIB_ESTIMBJINIT_RET_T members description

Name	Value	Description
ESTIMBJ_RET_INIT_OK	0	Initialization successful.
ESTIMBJ_RET_INIT_ERROR	1	Invalid inputs.

4.2 AMCLIB_ESTIMBJ_RET_T

Enum type of the AMCLIB_EstimBJ return value.

4.2.1 Source File

```
#include <AMCLIB_EstimBJ.h>
```

4.2.2 Enumerated Type Members

Table 441. AMCLIB_ESTIMBJ_RET_T members description

Name	Value	Description
ESTIMBJ_RET_IN_PROGRESS	0	Parameter estimation is in progress. The AMCLIB_EstimBJ must be called again in the next sampling period.
ESTIMBJ_RET_DONE	1	Parameter estimation has finished.
ESTIMBJ_RET_ERROR	2	Parameter estimation has failed.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

4.3 AMCLIB_ESTIMBJ_STATE_T

Enum type of the AMCLIB_EstimBJ internal state.

4.3.1 Source File

```
#include <AMCLIB_EstimBJ.h>
```

4.3.2 Enumerated Type Members

Table 442. AMCLIB_ESTIMBJ_STATE_T members description

Name	Value	Description
ESTIMBJ_STATE_ UNINITIALIZED	0	BJ estimator is not initialized.
ESTIMBJ_STATE_ ROTOR_ALIGN	*compiler default*	Rotor alignment.
ESTIMBJ_STATE_OL_ STARTUP	*compiler default*	Open loop startup.
ESTIMBJ_STATE_ FLUX_ESTIM	*compiler default*	Closed loop control.
ESTIMBJ_STATE_ PARAM_EST_MEAS	*compiler default*	Measurement of meachanical parameters.
ESTIMBJ_STATE_ DONE	*compiler default*	Measurement finished.
ESTIMBJ_STATE_ ERROR	*compiler default*	Failure.

4.4 AMCLIB_ESTIMRLINIT_RET_T

Enum type of the AMCLIB_EstimRLInit return value.

4.4.1 Source File

```
#include <AMCLIB_EstimRL.h>
```

4.4.2 Enumerated Type Members

Table 443. AMCLIB_ESTIMRLINIT_RET_T members description

Name	Value	Description
RET_INIT_OK	0	Initialization successful.
RET_INIT_ERROR	1	Invalid inputs.

4.5 AMCLIB_ESTIMRL_RET_T

Enum type of the AMCLIB_EstimRL return value.

4.5.1 Source File

```
#include <AMCLIB_EstimRL.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

4.5.2 Enumerated Type Members

Table 444. AMCLIB_ESTIMRL_RET_T members description

Name	Value	Description
RET_IN_PROGRESS	0	Parameter estimation is in progress. The AMCLIB_EstimRL must be called again in the next sampling period.
RET_DONE	1	Parameter estimation has finished.
RET_ERROR	2	Parameter estimation has failed.

4.6 AMCLIB_ESTIMRL_STATE_T

Enum type of the AMCLIB_EstimRL internal state.

4.6.1 Source File

```
#include <AMCLIB_EstimRL.h>
```

4.6.2 Enumerated Type Members

Table 445. AMCLIB_ESTIMRL_STATE_T members description

Name	Value	Description
STATE_ UNINITIALIZED	0	RL estimator is not initialized.
STATE_MEAS_LD	*compiler default*	Measurement of d-axis inductance.
STATE_MEAS_LQ	*compiler default*	Measurement of q-axis inductance.
STATE_LD_TO_LQ	*compiler default*	Transition between axes.
STATE_ POSTPROCESS_LD	*compiler default*	Postprocessing of accumulated values for d-axis measurement.
STATE_ POSTPROCESS_LQ	*compiler default*	Postprocessing of accumulated values for q-axis measurement.
STATE_UNWIND	*compiler default*	Gradual turn-off of the DC current at the end of measurement.
STATE_DONE	*compiler default*	Measurement finished.
STATE_ERROR	*compiler default*	Failure.

4.7 AMCLIB_WINDMILLING_RET_T

Enum type of the AMCLIB_Windmilling return value.

4.7.1 Source File

```
#include <AMCLIB_Windmilling.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

4.7.2 Enumerated Type Members

Table 446. AMCLIB_WINDMILLING_RET_T members description

Name	Value	Description
UNDECIDED	0	AMCLIB_Windmilling has not yet decided whether the motor is spinning or not. The AMCLIB_Windmilling must be called again in the next sampling period.
SPINNING	1	The motor is spinning.
STOPPED	2	The motor is stopped.

4.8 tBool

Enum representing basic boolean type.

4.8.1 Source File

```
#include <SWLIBS_Typedefs.h>
```

4.8.2 Enumerated Type Members

Table 447. tBool members description

Name	Value	Description
FALSE	0	Boolean type FALSE constant
TRUE	1	Boolean type TRUE constant

5 Compound data types

This section describes in details the compound data types definitions available in AMMCLib.

5.1 AMCLIB_BEMF_OBSRV_DQ_T_F16

Observer configuration structure.

5.1.1 Source File

```
#include <AMCLIB_BemfObsrvDQ.h>
```

5.1.2 Compound Type Members

Table 448. AMCLIB_BEMF_OBSRV_DQ_T_F16 members description

Type	Name	Description
SWLIBS_2Syst_F16	pEObsrv	Estimated BEMF - D/Q.
SWLIBS_2Syst_F32	pIObsrv	Estimated current - D/Q.
GFLIB_CONTROLLER_PIAW_R_T_F16	pParamD	Observer parameters for D-axis controller.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 448. AMCLIB_BEMF_OBSRV_DQ_T_F16 members description...*continued*

Type	Name	Description
GFLIB_CONTROLLER_PIAW_R_T_F16	pParamQ	Observer parameters for Q-axis controller.
SWLIBS_2Syst_F32	pIObsrvIn_1L	Inputs of RL circuit at step k-1 - low word
SWLIBS_2Syst_F16	pIObsrvIn_1H	Inputs of RL circuit at step k-1 - high word
tFrac16	f16IGain	Scaled RL circuit constant, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac16	f16UGain	Scaled voltage cross-coupling constant, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac16	f16WIGain	Scaled angular velocity cross-coupling constant, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac16	f16EGain	Scaled back-EMF cross-coupling constant, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tS16	s16Shift	Scaling bitwise shift applied to all cross-coupling constants, integer format in the range [-14, 14]. Function accuracy guaranteed only for range [-1, 1].

5.2 AMCLIB_BEMF_OBSRV_DQ_T_F32

Observer configuration structure.

5.2.1 Source File

```
#include <AMCLIB_BemfObsrvDQ.h>
```

5.2.2 Compound Type Members

Table 449. AMCLIB_BEMF_OBSRV_DQ_T_F32 members description

Type	Name	Description
SWLIBS_2Syst_F32	pEObsrv	Estimated BEMF - D/Q.
SWLIBS_2Syst_F32	pIObsrv	Estimated current - D/Q.
GFLIB_CONTROLLER_PIAW_R_T_F32	pParamD	Observer parameters for D-axis controller.
GFLIB_CONTROLLER_PIAW_R_T_F32	pParamQ	Observer parameters for Q-axis controller.
SWLIBS_2Syst_F32	pIObsrvIn_1L	Inputs of RL circuit at step k-1 - low word

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 449. AMCLIB_BEMF_OBSRV_DQ_T_F32 members description...*continued*

Type	Name	Description
SWLIBS_2Syst_F16	pIObsrvIn_1H	Inputs of RL circuit at step k-1 - high word
tFrac32	f32IGain	Scaled RL circuit constant, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32UGain	Scaled voltage cross-coupling constant, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32WIGain	Scaled angular velocity cross-coupling constant, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32EGain	Scaled back-EMF cross-coupling constant, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tS16	s16Shift	Scaling bitwise shift applied to all cross-coupling constants, integer format in the range [-30, 30]. Function accuracy guaranteed only for range [-1, 1].

5.3 AMCLIB_BEMF_OBSRV_DQ_T_FLT

Observer configuration structure.

5.3.1 Source File

```
#include <AMCLIB_BemfObsrvDQ.h>
```

5.3.2 Compound Type Members

Table 450. AMCLIB_BEMF_OBSRV_DQ_T_FLT members description

Type	Name	Description
SWLIBS_2Syst_FLT	pEObsrv	Estimated BEMF - D/Q.
SWLIBS_2Syst_FLT	pIObsrv	Estimated current - D/Q.
GFLIB_CONTROLLER_PIAW_R_T_FLT	pParamD	Observer parameters for D-axis controller.
GFLIB_CONTROLLER_PIAW_R_T_FLT	pParamQ	Observer parameters for Q-axis controller.
SWLIBS_2Syst_FLT	pIObsrvIn_1	Inputs of RL circuit at step k-1
tFloat	fltIGain	RL circuit constant, single precision floating point format.
tFloat	fltUGain	Voltage cross-coupling constant, single precision floating point format.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 450. AMCLIB_BEMF_OBSRV_DQ_T_FLT members description...*continued*

Type	Name	Description
tFloat	fltWIGain	Angular velocity cross-coupling constant, single precision floating point format.
tFloat	fltEGain	Back-EMF cross-coupling constant, single precision floating point format.

5.4 AMCLIB_CURRENT_LOOP_PIP_T_F16

ADD BRIEF DESCRIPTION.

5.4.1 Source File

```
#include <AMCLIB_CurrentLoopPIp.h>
```

5.4.2 Compound Type Members

Table 451. AMCLIB_CURRENT_LOOP_PIP_T_F16 members description

Type	Name	Description
GFLIB_CONTROLLER_PIAW_P_T_F16	pPIpAWD	D-axis ControllerPIpAW paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_F16	pPIpAWQ	Q-axis ControllerPIpAW paremeters structure.
SWLIBS_2Syst_F16 *	pIDQReq	Pointer to the structure with the required current.
SWLIBS_2Syst_F16 *	pIDQFbck	Pointer to the structure with the feedback current.

5.5 AMCLIB_CURRENT_LOOP_PIP_T_F32

ADD BRIEF DESCRIPTION.

5.5.1 Source File

```
#include <AMCLIB_CurrentLoopPIp.h>
```

5.5.2 Compound Type Members

Table 452. AMCLIB_CURRENT_LOOP_PIP_T_F32 members description

Type	Name	Description
GFLIB_CONTROLLER_PIAW_P_T_F32	pPIpAWD	D-axis ControllerPIpAW paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_F32	pPIpAWQ	Q-axis ControllerPIpAW paremeters structure.
SWLIBS_2Syst_F32 *	pIDQReq	Pointer to the structure with the required current.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 452. AMCLIB_CURRENT_LOOP_PIP_T_F32 members description...*continued*

Type	Name	Description
SWLIBS_2Syst_F32 *	pIDQFbck	Pointer to the structure with the feedback current.

5.6 AMCLIB_CURRENT_LOOP_PIP_T_FLT

ADD BRIEF DESCRIPTION.

5.6.1 Source File

```
#include <AMCLIB_CurrentLoopPIp.h>
```

5.6.2 Compound Type Members

Table 453. AMCLIB_CURRENT_LOOP_PIP_T_FLT members description

Type	Name	Description
GFLIB_CONTROLLER_PIAW_P_T_FLT	pPIpAWD	D-axis ControllerPIpAW paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_FLT	pPIpAWQ	Q-axis ControllerPIpAW paremeters structure.
SWLIBS_2Syst_FLT *	pIDQReq	Pointer to the structure with the required current.
SWLIBS_2Syst_FLT *	pIDQFbck	Pointer to the structure with the feedback current.

5.7 AMCLIB_CURRENT_LOOP_T_F16

CurrentLoop configuration structure.

5.7.1 Source File

```
#include <AMCLIB_CurrentLoop.h>
```

5.7.2 Compound Type Members

Table 454. AMCLIB_CURRENT_LOOP_T_F16 members description

Type	Name	Description
GFLIB_CONTROLLER_PIAW_R_T_F16	pPIrAWD	D-axis ControllerPIrAW paremeters structure.
GFLIB_CONTROLLER_PIAW_R_T_F16	pPIrAWQ	Q-axis ControllerPIrAW paremeters structure.
SWLIBS_2Syst_F16 *	pIDQReq	Pointer to the structure with the required current.
SWLIBS_2Syst_F16 *	pIDQFbck	Pointer to the structure with the feedback current.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.8 AMCLIB_CURRENT_LOOP_T_F32

CurrentLoop configuration structure.

5.8.1 Source File

```
#include <AMCLIB_CurrentLoop.h>
```

5.8.2 Compound Type Members

Table 455. AMCLIB_CURRENT_LOOP_T_F32 members description

Type	Name	Description
GFLIB_CONTROLLER_PIAW_R_T_F32	pPIrAWD	D-axis ControllerPIrAW paremeters structure.
GFLIB_CONTROLLER_PIAW_R_T_F32	pPIrAWQ	Q-axis ControllerPIrAW paremeters structure.
SWLIBS_2Syst_F32 *	pIDQReq	Pointer to the structure with the required current.
SWLIBS_2Syst_F32 *	pIDQFbck	Pointer to the structure with the feedback current.

5.9 AMCLIB_CURRENT_LOOP_T_FLT

CurrentLoop configuration structure.

5.9.1 Source File

```
#include <AMCLIB_CurrentLoop.h>
```

5.9.2 Compound Type Members

Table 456. AMCLIB_CURRENT_LOOP_T_FLT members description

Type	Name	Description
GFLIB_CONTROLLER_PIAW_R_T_FLT	pPIrAWD	D-axis ControllerPIrAW paremeters structure.
GFLIB_CONTROLLER_PIAW_R_T_FLT	pPIrAWQ	Q-axis ControllerPIrAW paremeters structure.
SWLIBS_2Syst_FLT *	pIDQReq	Pointer to the structure with the required current.
SWLIBS_2Syst_FLT *	pIDQFbck	Pointer to the structure with the feedback current.

5.10 AMCLIB_ESTIMBJ_INIT_T_FLT

Initialization parameters of the RL estimator.

© NXP B.V. 2023. All rights reserved.

5.10.1 Source File

```
#include <AMCLIB_EstimBJ.h>
```

5.10.2 Compound Type Members

Table 457. AMCLIB_ESTIMBJ_INIT_T_FLT members description

Type	Name	Description
tU32	u32SamplingFreq	Sampling frequency [1/s].
tFloat	fltIN	Nominal current [A].
tFloat	fltWN	Nominal velocity [rpm].
tFloat	fltPP	Number of pole pairs [-].
tFloat	fltLd	D-axis inductance [H].
tFloat	fltLq	Q-axis inductance [H].
tFloat	fltRs	Stator resistance [Ohm].
tFloat	fltDtVDrop	Dead time voltage drop of the power stage [V].
tFloat	fltAlignTime	Time needed for rotor alignment [s].
tFloat	fltIReqOl	Required d-axis current for open loop startup [A].
tFloat	fltWStepOl	Required speed step for ramp [delta rpm/s]
tFloat	fltDCWReq1	Required speed for measurment point 1 [rpm]
tFloat	fltDCWReq2	Required speed for measurment point 2 [rpm]
tFloat	fltDcPiPropGain	Proportional gain of the speed loops DC controller [-].
tFloat	fltDcPiIntegGain	Inegral gain of the speed loops DC controller [-].
tFloat	fltAcPiPropGain	Proportional gain of the speed loops AC controller [-].
tFloat	fltAcPiIntegGain	Inegral gain of the speed loops AC controller [-].

5.11 AMCLIB_ESTIMBJ_INNERST_SIM_T_FLT

Internal structure for Matlab Simulink rapid accelerator. No user-editable values.

5.11.1 Source File

```
#include <AMCLIB_EstimBJ.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.11.2 Compound Type Members

Table 458. AMCLIB_ESTIMBJ_INNERST_SIM_T_FLT members description

Type	Name	Description
AMCLIB_ESTIMBJ_STATE_T	pState	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tBool	bTwoPointMeas	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tU32	u32AlignTimer	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tU32	u32TimerPreset	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltIReq	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltDCWReq1	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltDCWReq2	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltACWReq	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltWMax	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltPP	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltWOl	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltThOl	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
GFLIB_RAMP_T_FLT	pOLRamp	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
GFLIB_INTEGRATOR_TR_T_F32	pOLInteg	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
GDFLIB_FILTER_MA_T_FLT	pIqMAFilter	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
SWLIBS_2Syst_FLT	pThRealTransform	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
SWLIBS_2Syst_FLT	pIDQRealFbck	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 458. AMCLIB_ESTIMBJ_INNERST_SIM_T_FLT members description...*continued*

Type	Name	Description
tFloat	fltWDcReq	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltWAcMaxReq	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltIqDcReq	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltIqAcReq	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltIqAcMaxReq	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
GDFLIB_FILTER_MA_T_FLT	pWDcReqMAFilter	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
GDFLIB_FILTER_MA_T_FLT	pWAcMaxReqMAFilter	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
GFLIB_CONTROLLER_PIAW_P_T_FLT	pWDcPI	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
GFLIB_CONTROLLER_PIAW_P_T_FLT	pWAcMaxPI	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltWAcMaxErr	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltWDcErr2	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFrac32	f32Theta	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
SWLIBS_2Syst_FLT	pUDQReq	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
GFLIB_CONTROLLER_PIAW_R_T_FLT	pCloopDPI	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
GFLIB_CONTROLLER_PIAW_R_T_FLT	pCloopQPI	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
SWLIBS_2Syst_FLT	pIDQReqCLoop	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
SWLIBS_2Syst_FLT	pIDQFbckCLoop	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 458. AMCLIB_ESTIMBJ_INNERST_SIM_T_FLT members description...*continued*

Type	Name	Description
tFloat	fltThEl	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
SWLIBS_2Syst_FLT	pThTransform	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltDcRaw	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
SWLIBS_2Syst_FLT	pWABAcRaw	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
SWLIBS_2Syst_FLT	pPllSinCosEst	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltDcFilt	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltAcFilt	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltWABEstErr	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
GDFLIB_FILTER_MA_T_FLT	pPllDMAFilter	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
GDFLIB_FILTER_MA_T_FLT	pPllQMAFilter	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
GDFLIB_FILTER_MA_T_FLT	pPllDcMAFilter	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltPllPiPropGain	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFrac32	f32ThAcEst	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFrac32	f32ThAcInc	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
AMCLIB_FLUXESTIMAB_T_FLT	pFluxEstim	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltLd	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltLq	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 458. AMCLIB_ESTIMBJ_INNERST_SIM_T_FLT members description...*continued*

Type	Name	Description
tFloat	fltWEl	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltThErr	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltThEstim	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltWEstim	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
AMCLIB_TRACK_OBSRV_T_FLT	pTrackObsrv	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltPsiRaw	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltPsiFilt	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltDtVDrop	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
GDFLIB_FILTER_MA_T_FLT	pFluxMAFilter	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tU32	u32SSTimeoutCnt	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tU32	u32SSBandCnt	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tU32	u32SSTimeoutTime	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tU32	u32SSTime	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltSSBandHalfMax	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltSSBandHalf	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltInFilt	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
GDFLIB_FILTER_MA_T_FLT	pSSMAFilter	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 458. AMCLIB_ESTIMBJ_INNERST_SIM_T_FLT members description...*continued*

Type	Name	Description
tU16	u16MeasNum	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltMeDcArr	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltWDcArr	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltInertiaArr	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltWDcErr	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltGainFOC	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltInvGainATO	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltPhShiftCorr	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltKt	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltJ	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltA	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.
tFloat	fltB	Internal structure for Matlab Simulink rapid accelerator. No user-editable values.

5.12 AMCLIB_ESTIMBJ_INNERST_T_FLT

Internal state of the RL estimator. No user-editable values.

5.12.1 Source File

```
#include <AMCLIB_EstimBJ.h>
```

5.12.2 Compound Type Members

Table 459. AMCLIB_ESTIMBJ_INNERST_T_FLT members description

Type	Name	Description
AMCLIB_ESTIMBJ_STATE_T	pState	Mechanical parameters estimator state

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 459. AMCLIB_ESTIMBJ_INNERST_T_FLT members description...*continued*

Type	Name	Description
tBool	bTwoPointMeas	Indicator of the advanced mode measurement
tU32	u32AlignTimer	Rotor alignment timer
tU32	u32TimerPreset	Rotor alignment timer
tFloat	fltIReq	Required current in open loop
tFloat	fltDCWReq1	Required DC speed point 1 for measurement
tFloat	fltDCWReq2	Required DC speed point 2 for measurement
tFloat	fltACWReq	Required amplitude of the injected AC speed
tFloat	fltWMax	RMaximal velocty
tFloat	fltPP	Number of pole pairs
tFloat	fltWOl	Open loop velocity
tFloat	fltThOl	Open loop position
GFLIB_RAMP_T_FLT	pOLRamp	Parameter structure for open loop ramp
GFLIB_INTEGRATOR_TR_T_F32	pOLInteg	Parameter structure for open loop integrator
GDFLIB_FILTER_MA_T_FLT	pIqMAFilter	Parameter structure for open loop MA filter
SWLIBS_2Syst_FLT	pThRealTransform	Structure to contain sine and cosine value of real rotor position
SWLIBS_2Syst_FLT	pIDQRealFbck	Real DQ feedback currents
tFloat	fltWDcReq	Required DC speed
tFloat	fltWAcMaxReq	Required amplitude of AC speed
tFloat	fltIqDcReq	Required Q current based on DC speed regulator
tFloat	fltIqAcReq	Required Q current amplitude based on AC speed regulator
tFloat	fltIqAcMaxReq	Required Q current based on AC speed regulator
GDFLIB_FILTER_MA_T_FLT	pWDcReqMAFilter	MA filter for required DC speed
GDFLIB_FILTER_MA_T_FLT	pWAcMaxReqMAFilter	MA filter for required AC speed
GFLIB_CONTROLLER_PIAW_P_T_FLT	pWDcPI	DC speed controller
GFLIB_CONTROLLER_PIAW_P_T_FLT	pWAcMaxPI	AC speed controller
tFloat	fltWAcMaxErr	Input error for AC speed controller

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 459. AMCLIB_ESTIMBJ_INNERST_T_FLT members description...*continued*

Type	Name	Description
tFloat	fltWDcErr2	Input error for DC speed controller
tFrac32	f32Theta	Modulated AC signals phase
SWLIBS_2Syst_FLT	pUDQReq	Required DQ voltage
AMCLIB_CURRENT_LOOP_T_FLT	pCurrentLoop	Parameter structure for current loop
SWLIBS_2Syst_FLT	pIDQReqCLoop	Required DQ current
SWLIBS_2Syst_FLT	pIDQFbckCLoop	Measured DQ current
tFloat	fltThEl	Rotor angular position
SWLIBS_2Syst_FLT	pThTransform	Structure to store sine and cosine of rotor angular posion
tFloat	fltDcRaw	Demodulated DC speed
SWLIBS_2Syst_FLT	pWABAcRaw	Demodulated AC speed
SWLIBS_2Syst_FLT	pPllSinCosEst	Structure to store sine and cosine of PLL's internal phase
tFloat	fltDcFilt	Filtered demodulated DC speed
tFloat	fltAcFilt	Filtered demodulated AC speed
tFloat	fltWABEstErr	PLL error
GDFLIB_FILTER_MA_T_FLT	pPllDMAFilter	MA filter for D component of AC speed
GDFLIB_FILTER_MA_T_FLT	pPllQMAFilter	MA filter for Q component of AC speed
GDFLIB_FILTER_MA_T_FLT	pPllDcMAFilter	MA filter for DC speed
tFloat	fltPllPiPropGain	Proportional gain of PLL's P controller
tFrac32	f32ThAcEst	PLL's internal phase with corretion from P controller
tFrac32	f32ThAcInc	PLL's internal phase
AMCLIB_FLUXESTIMAB_T_FLT	pFluxEstim	Parameter structure for flux estimator
tFloat	fltLd	D inductance of the motor
tFloat	fltLq	Q inductance of the motor
tFloat	fltWEl	State variable to store motor velocity
tFloat	fltThErr	Position error for ATO
tFloat	fltThEstim	Estimated angular position filtered by ATO
tFloat	fltWEstim	Estimated velocity filtered by ATO
AMCLIB_TRACK_OBSRV_T_FLT	pTrackObsrv	Parameter structure for ATO
tFloat	fltPsiRaw	Estimated rotor flux

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 459. AMCLIB_ESTIMBJ_INNERST_T_FLT members description...*continued*

Type	Name	Description
tFloat	fltPsiFilt	Filtered rotor flux
tFloat	fltDtVDrop	Dead time voltage drop
GDFLIB_FILTER_MA_T_FLT	pFluxMAFilter	Parameter structure for MA filter for flux fitering
tU32	u32SSTimeoutCnt	Steady state timeout counter
tU32	u32SSBandCnt	Steady state in band counter
tU32	u32SSTimeoutTime	Steady state timeout
tU32	u32SSTime	Minimal time needed for stable behaviour
tFloat	fltSSBandHalfMax	Half of maximal deviation from mean steady value
tFloat	fltSSBandHalf	Half of allowed deviation from mean steady value
tFloat	fltInFilt	Filtered observed variable in stary state
GDFLIB_FILTER_MA_T_FLT	pSSMAFilter	Statedy state MA filter
tU16	u16MeasNum	Number of current measuement
tFloat	fltMeDcArr	Array to store measuder values of Me
tFloat	fltWDcArr	Array to store measuder values of W
tFloat	fltInertiaArr	Array to store measuder values of J
tFloat	fltWDcErr	DC speed error
tFloat	fltGainFOC	Correction coefficient for FOC gain
tFloat	fltInvGainATO	Correction coefficient for ATO gain
tFloat	fltPhShiftCorr	Correction coefficient for FOC and ATO phase shift

5.13 AMCLIB_ESTIMBJ_T_FLT

Estimator configuration structure.

5.13.1 Source File

```
#include <AMCLIB_EstimBJ.h>
```

5.13.2 Compound Type Members

Table 460. AMCLIB_ESTIMBJ_T_FLT members description

Type	Name	Description
tFloat	fltKt	Estimated torque constant [N·m/A].
tFloat	fltJ	Estimated moment of inertia [kg·m^2].

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 460. AMCLIB_ESTIMBJ_T_FLT members description...*continued*

Type	Name	Description
tFloat	fltA	Estimated static friction torque [N·m]. In "basic" mode returns zero.
tFloat	fltB	Estimated friction coefficient [N·m·s/rad]. In "basic" mode returns zero.
AMCLIB_ESTIMBJ_INNERST_T_FLT	pInnerState	Internal state of the RL estimator. No user-editable values.

5.14 AMCLIB_ESTIMRL_INIT_T_FLT

Initialization parameters of the RL estimator.

5.14.1 Source File

```
#include <AMCLIB_EstimRL.h>
```

5.14.2 Compound Type Members

Table 461. AMCLIB_ESTIMRL_INIT_T_FLT members description

Type	Name	Description
tFloat	fltIDcMax	Maximum DC current [A].
tU16	u16LdqNumMeas	Number of measured d-axis DC current levels. If this parameter is not 1, the function will perform automatic measurement for a range of DC currents. Otherwise, only one measurement will be performed.
tFloat	fltIDcLd	DC current used for Ld measurement [A].
tFloat	fltIDcLq	DC current used for Lq measurement [A].
tFloat	fltIDcNegMax	Maximum allowed negative d-axis DC current [A]. Note that too large negative DC current can cause permanent damage to the motor. Setting fltIDcNegMax to a safe value prevents irreversible demagnetization of the PMSM's magnets. The value of fltIDcNegMax must be negative or zero.

5.15 AMCLIB_ESTIMRL_INNERST_T_FLT

Internal state of the RL estimator. No user-editable values.

5.15.1 Source File

```
#include <AMCLIB_EstimRL.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.15.2 Compound Type Members

Table 462. AMCLIB_ESTIMRL_INNERST_T_FLT members description

Type	Name	Description
AMCLIB_ESTIMRL_STATE_T	pState	RL estimator state
tFrac32	f32FAc	Normalized fraction of the AC current frequency to be integrated in each iteration
tFloat	fltIDcMax	Maximum DC of current
tFloat	fltIDcNegMax	Maximum allowed negative DC current
tFloat	fltIDcMaxInv	Inverse of the maximum DC current
tFloat	fltIAcReq	Amplitude of the required AC current
tFloat	fltIAcReqInv	Inverse of the amplitude of the required AC current
tFloat	fltIDcDReq	Required DC current in D axis
tFloat	fltIDcDReqk_1	Required DC current in previous step in D axis
tFloat	fltIDcQReq	Required DC current in Q axis
tFloat	fltIDcQReqk_1	Required DC current in previous step in Q axis
tFloat	fltIDcReqStep	Measurement step of the required DC current
tFloat	fltIAlBeEstErr	Current estimation error
SWLIBS_2Syst_FLT	pIAlBeAc	Alpha/beta components of the estimated AC current
SWLIBS_2Syst_FLT	pSinCosEst	Sine/Cosine of the estimated current phase
SWLIBS_2Syst_FLT	pIDQAcRaw	d/q components of the estimated AC current
SWLIBS_2Syst_FLT	pIDQAcFilt	Filtered d/q components of the estimated AC current
GDFLIB_FILTER_MA_T_FLT	pDAxisFilter	Parameters structure of the MA filter in the d axis in the PLL
GDFLIB_FILTER_MA_T_FLT	pQAxisFilter	Parameters structure of pDcFilter in the PLL
GDFLIB_FILTER_MA_T_FLT	pDcFilter	Parameters structure of the MA filter filtering the DC current in PLL
GDFLIB_FILTER_MA_T_FLT	pStdFilter	Parameters structure of the MA filter filtering the standard deviation of current noise
tFloat	fltPLLPropGain	Gain of the P controller in the PLL

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 462. AMCLIB_ESTIMRL_INNERST_T_FLT members description...*continued*

Type	Name	Description
tFrac32	f32ThAc	High-frequency AC voltage phase; <-PI, PI) range normalized into <-1, 1)
tFrac32	f32ThEst	Estimated AC current phase; <-PI, PI) range normalized into <-1, 1)
tFloat	fltIDcD	Estimated DC current in D axis
tFloat	fltIDcQ	Estimated DC current in Q axis
tFloat	fltIDcDAvg	Averaging accumulator for the estimated DC current in D axis
tFloat	fltIDcDAvgk_1	Averaged estimated DC current in D axis in the last step
tFloat	fltPhComp	Compensation of phase error due to sampling lag
tS32	s32ConvDetState	Convergence detector state
tS32	s32ConvDetStateLimL	Convergence detector lower limit
tS32	s32ConvDetStateLimH	Convergence detector coarse/fine tracking threshold
tU32	u32ConvDetToutCnt	Convergence detector timeout counter
tU32	u32ConvDetToutReload	Convergence detector counter reload value
tU32	u32AvgCnt	Noise averaging counter
tU32	u32AvgN	Noise averaging window length
tFloat	fltAvgNInv	Reciprocal of the noise averaging inverse window length
tFloat	fltAvgNInvMaxDelta	Difference of reciprocals of the noise averaging inverse window length (max/nominal)
tU32	u32AvgCntMax	Maximum length of the noise averaging window
tFloat	fltPhAvg	Averaging accumulator for phase shift
tFloat	fltIAcAvg	Averaging accumulator for AC current
tFloat	fltUAcAvg	Averaging accumulator for AC voltage
tFloat	fltAvgMul	Averaging window length multiplier
tFloat	fltAvgMin	Averaging window minimum length
tU16	u16LdqNumMeas	Number of Ld/Lq table measurements
tU32	u32TblCounter	Ld/Lq table counter
tU8	u8RotorFixed	0 == rotor is not mechanically fixed, 1 == rotor is mechanically fixed

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 462. AMCLIB_ESTIMRL_INNERST_T_FLT members description...*continued*

Type	Name	Description
tFloat	fltIDcRampCoef	DC current ramp coefficient
tFloat	fltIDcRampCoefMax	Maximum DC current ramp coefficient
tFloat	fltIDcDRampAcc	DC current ramp accumulator in D axis
tFloat	fltIDcQRampAcc	DC current ramp accumulator in Q axis
tFloat	fltUDcDAcc	DC voltage integrator in D axis
tFloat	fltUDcDAccAvg	Averaging accumulator for DC voltage in D axis
tFloat	fltUDcDAccAvgk_1	Averaged DC voltage in D axis in previous step
tFloat	fltUDcQAcc	DC voltage integrator in Q axis
tFloat	fltUAcAcc	AC voltage integrator
tFloat	fltUDcKi	DC voltage integrator gain
tFloat	fltUAcKi	AC voltage integrator gain
tFloat	fltUDcKiNominal	Nominal DC voltage integrator gain
tFloat	fltUAcKiNominal	Nominal AC voltage integrator gain
tU16	u16RampSteady	Indicator of the steady state of the voltage controller ramp
tU16	u16DcbLimitFlag	DC bus voltage limitation indicator
tU16	u16DcbLimitFlagk_1	DC bus voltage limitation indicator in previous step
tU16	u16DcbLimitCnt	DC bus voltage limitation counter
tFloat	fltPhaseShift	Voltage/current phase shift
tFloat	fltUdcHalf	Minimum DC bus voltage, halved and divided by modulation index
tFloat	fltFsInv	Inverse of the sampling frequency
tFloat	fltImpConst	Constant used for calculation of impedance
tFloat	fltLdk_1	Estimated d-axis inductance in previous step
tFloat	fltLqk_1	Estimated q-axis inductance in previous step
tFloat	fltIDcLd	DC current level to use for measurement of scalar Ld
tFloat	fltIDcLq	DC current level to use for measurement of scalar Lq
tFloat	fltUdtAcc	Dead time voltage drop averaging accumulator
tFloat	fltRAcc	Resistance averaging accumulator

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 462. AMCLIB_ESTIMRL_INNERST_T_FLT members description...*continued*

Type	Name	Description
tU16	u16LdqNumMeasR	Number of measurements of R
tFloat	fltUDcBusMax	Maximum observed DC bus voltage

5.16 AMCLIB_ESTIMRL_RUN_T_FLT

Run-time parameters of the RL estimator.

5.16.1 Source File

```
#include <AMCLIB_EstimRL.h>
```

5.16.2 Compound Type Members

Table 463. AMCLIB_ESTIMRL_RUN_T_FLT members description

Type	Name	Description
tFloat *	pLdqTable	Pointer to a table with measured values. The table has 3 rows and u16LdqNumMeas columns stored in column-major order. Parameter u16LdqNumMeas of the init function AMCLIB_EstimRLInit_FLT determines the number of used DC current levels. The array pointed to by pLdqTable must provide room for 3*u16LdqNumMeas 32-bit values. The first row of the table contains the DC current levels [A], the second row contains the estimated d-axis inductances, and the third row contains the estimated q-axis inductances. The inductances are calculated in [H].
tFloat	fltIDcDReq	Required DC current in the d-axis [A]
tFloat	fltIDcQReq	Required DC current in the q-axis [A]
tFloat	fltIAcReq	Amplitude of the required AC current [A]
tU16	u16FAc	AC current frequency [Hz]
tU8	u8LdqSwitch	Switches between Ld (u8LdqSwitch = 0) and Lq (u8LdqSwitch = 1) measurement

5.17 AMCLIB_ESTIMRL_T_FLT

Estimator configuration structure.

5.17.1 Source File

```
#include <AMCLIB_EstimRL.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.17.2 Compound Type Members

Table 464. AMCLIB_ESTIMRL_T_FLT members description

Type	Name	Description
tFloat	fltLd	Estimated d-axis inductance at zero DC current [H].
tFloat	fltLq	Estimated q-axis inductance at maximal DC current [H].
tFloat	fltR	Estimated resistance [Ohm].
tFloat	fltUdt	Estimated dead time voltage drop of the power stage [V].
AMCLIB_ESTIMRL_INNERST_T_FLT	pInnerState	Internal state of the RL estimator. No user-editable values.

5.18 AMCLIB_FLUXESTIMAB_T_F16

Structure containing coefficients for calculation of the flux estimator.

5.18.1 Source File

```
#include <AMCLIB_FluxEstimAB.h>
```

5.18.2 Compound Type Members

Table 465. AMCLIB_FLUXESTIMAB_T_F16 members description

Type	Name	Description
tFrac16	f16KRs	Coefficient K_{Rs} derived from stator resistance.
tFrac16	f16KLs	Coefficient K_{Ls} derived from stator inductance.
tU16	u16NShift	Scaling coefficient.

5.19 AMCLIB_FLUXESTIMAB_T_F32

Structure containing coefficients for calculation of the flux estimator.

5.19.1 Source File

```
#include <AMCLIB_FluxEstimAB.h>
```

5.19.2 Compound Type Members

Table 466. AMCLIB_FLUXESTIMAB_T_F32 members description

Type	Name	Description
tFrac32	f32KRs	Coefficient K_{Rs} derived from stator resistance.
tFrac32	f32KLs	Coefficient K_{Ls} derived from stator inductance.
tU16	u16NShift	Scaling coefficient.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.20 AMCLIB_FLUXESTIMAB_T_FLT

Structure containing coefficients for calculation of the flux estimator.

5.20.1 Source File

```
#include <AMCLIB_FluxEstimAB.h>
```

5.20.2 Compound Type Members

Table 467. AMCLIB_FLUXESTIMAB_T_FLT members description

Type	Name	Description
tFloat	fltRs	Stator resistance [Ohm].
tFloat	fltLs	Stator inductance [H].

5.21 AMCLIB_FW_DEBUG_T_F16

FW configuration structure with debugging information.

5.21.1 Source File

```
#include <AMCLIB_FW.h>
```

5.21.2 Compound Type Members

Table 468. AMCLIB_FW_DEBUG_T_F16 members description

Type	Name	Description
GDFLIB_FILTER_MA_T_F16	pFilterFW	Field weakening angle FilterMA paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_F16	pPIpAWFW	Field weakening angle ControllerPIpAW paremeters structure.
tFrac16 *	pIQFbck	Pointer to the q-axis component of the feedback current in the two-phase rotational orthogonal system (d-q).
tFrac16 *	pUQReq	Pointer to the q-axis component of the required voltage in the two-phase rotational orthogonal system (d-q).
tFrac16 *	pUQLim	Pointer to the q-axis voltage controller upper limit (should point to the AMCLIB_CURRENT_LOOP_T_F16 structure element pPIrAWQ.f16UpperLimit). The limit must be a positive value.
tFrac16	f16IQErrSign	FW - Current deviation after sign elimination.
tFrac16	f16IQErr	FW - Current deviation.
tFrac16	f16FWErr	FW - Field weakening feedback control variable.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 468. AMCLIB_FW_DEBUG_T_F16 members description...*continued*

Type	Name	Description
tFrac16	f16UQErr	FW - Voltage deviation.
tFrac16	f16FWErrFilt	FW - Filtered field weakening feedback control variable.
tFrac16	f16FWAngle	FW - Field weakening angle.
tFrac16	f16FWSin	FW - Q-axis unity current phasor component.
tFrac16	f16FWCos	FW - D-axis unity current phasor component.

5.22 AMCLIB_FW_DEBUG_T_F32

FW configuration structure with debugging information.

5.22.1 Source File

```
#include <AMCLIB_FW.h>
```

5.22.2 Compound Type Members

Table 469. AMCLIB_FW_DEBUG_T_F32 members description

Type	Name	Description
GDFLIB_FILTER_MA_T_F32	pFilterFW	Field weakening angle FilterMA paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_F32	pPIpAWFW	Field weakening angle ControllerPIpAW paremeters structure.
tFrac32 *	pIQFbck	Pointer to the q-axis component of the feedback current in the two-phase rotational orthogonal system (d-q).
tFrac32 *	pUQReq	Pointer to the q-axis component of the required voltage in the two-phase rotational orthogonal system (d-q).
tFrac32 *	pUQLim	Pointer to the q-axis voltage controller upper limit. Must be a positive value.
tFrac32	f32IQErrSign	FW - Current deviation after sign elimination.
tFrac32	f32IQErr	FW - Current deviation.
tFrac32	f32FWErr	FW - Field weakening feedback control variable.
tFrac32	f32UQErr	FW - Voltage deviation.
tFrac32	f32FWErrFilt	FW - Filtered field weakening feedback control variable.
tFrac32	f32FWAngle	FW - Field weakening angle.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 469. AMCLIB_FW_DEBUG_T_F32 members description...*continued*

Type	Name	Description
tFrac32	f32FWSin	FW - Q-axis unity current phasor component.
tFrac32	f32FWCos	FW - D-axis unity current phasor component.

5.23 AMCLIB_FW_DEBUG_T_FLT

FW configuration structure with debugging information.

5.23.1 Source File

```
#include <AMCLIB_FW.h>
```

5.23.2 Compound Type Members

Table 470. AMCLIB_FW_DEBUG_T_FLT members description

Type	Name	Description
GDFLIB_FILTER_MA_T_FLT	pFilterFW	Field weakening angle FilterMA paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_FLT	pPIpAWFW	Field weakening angle ControllerPIpAW paremeters structure.
tFloat *	pIQFbck	Pointer to the q-axis component of the feedback current in the two-phase rotational orthogonal system (d-q).
tFloat *	pUQReq	Pointer to the q-axis component of the required voltage in the two-phase rotational orthogonal system (d-q).
tFloat *	pUQLim	Pointer to the q-axis voltage controller upper limit (should point to the AMCLIB_CURRENT_LOOP_T_FLT structure element pPIrAWQ.fltUpperLimit). The limit must be a positive value.
tFloat	fltUmaxDivImax	Maximal coil voltage divided by the maximal coil current.
tFloat	fltIQErrSign	FW - Current deviation after sign elimination.
tFloat	fltIQErr	FW - Current deviation.
tFloat	fltFWErr	FW - Field weakening feedback control variable.
tFloat	fltUQErr	FW - Voltage deviation.
tFloat	fltFWErrFilt	FW - Filtered field weakening feedback control variable.
tFloat	fltFWAngle	FW - Field weakening angle.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 470. AMCLIB_FW_DEBUG_T_FLT members description...*continued*

Type	Name	Description
tFloat	fltFWSin	FW - Q-axis unity current phasor component.
tFloat	fltFWCos	FW - D-axis unity current phasor component.

5.24 AMCLIB_FW_SPEED_LOOP_DEBUG_T_F16

FWSpeedLoop configuration structure with debugging information.

5.24.1 Source File

```
#include <AMCLIB_FWSpeedLoop.h>
```

5.24.2 Compound Type Members

Table 471. AMCLIB_FW_SPEED_LOOP_DEBUG_T_F16 members description

Type	Name	Description
GDFLIB_FILTER_MA_T_F16	pFilterW	Velocity FilterMA paremeters structure.
GDFLIB_FILTER_MA_T_F16	pFilterFW	Field weakening angle FilterMA paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_F16	pPIpAWQ	Q-axis ControllerPIpAW paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_F16	pPIpAWFW	Field weakening angle ControllerPIpAW paremeters structure.
GFLIB_RAMP_T_F32	pRamp	Speed ramp paremeters structure.
tFrac16 *	pIQFbck	Pointer to the q-axis component of the feedback current in the two-phase rotational orthogonal system (d-q).
tFrac16 *	pUQReq	Pointer to the q-axis component of the required voltage in the two-phase rotational orthogonal system (d-q).
tFrac16 *	pUQLim	Pointer to the q-axis voltage controller upper limit (should point to the AMCLIB_CURRENT_LOOP_T_F16 structure element pPIrAWQ.f16UpperLimit). The limit must be a positive value.
tFrac16	f16WReqFilt	SpeedLoop - Filtered value of the required angular velocity.
tFrac16	f16WErr	SpeedLoop - Velocity deviation.
tFrac16	f16IDQReqAmp	SpeedLoop - Filtered value of the measured angular velocity.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 471. AMCLIB_FW_SPEED_LOOP_DEBUG_T_F16 members description...*continued*

Type	Name	Description
tFrac16	f16WFbckFilt	SpeedLoop - Required current amplitude.
tFrac16	f16IQErrSign	FW - Current deviation after sign elimination.
tFrac16	f16FWErr	FW - Current deviation.
tFrac16	f16IQErr	FW - Field weakening feedback control variable.
tFrac16	f16UQErr	FW - Voltage deviation.
tFrac16	f16FWErrFilt	FW - Filtered field weakening feedback control variable.
tFrac16	f16FWAngle	FW - Field weakening angle.
tFrac16	f16FWSin	FW - Q-axis unity current phasor component.
tFrac16	f16FWCos	FW - D-axis unity current phasor component.

5.25 AMCLIB_FW_SPEED_LOOP_DEBUG_T_F32

FWSpeedLoop configuration structure with debugging information.

5.25.1 Source File

```
#include <AMCLIB_FWSpeedLoop.h>
```

5.25.2 Compound Type Members

Table 472. AMCLIB_FW_SPEED_LOOP_DEBUG_T_F32 members description

Type	Name	Description
GDFLIB_FILTER_MA_T_F32	pFilterW	Velocity FilterMA paremeters structure.
GDFLIB_FILTER_MA_T_F32	pFilterFW	Field weakening angle FilterMA paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_F32	pPIpAWQ	Q-axis ControllerPIpAW paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_F32	pPIpAWFW	Field weakening angle ControllerPIpAW paremeters structure.
GFLIB_RAMP_T_F32	pRamp	Speed ramp paremeters structure.
tFrac32 *	pIQFbck	Pointer to the q-axis component of the feedback current in the two-phase rotational orthogonal system (d-q).
tFrac32 *	pUQReq	Pointer to the q-axis component of the required voltage in the two-phase rotational orthogonal system (d-q).

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 472. AMCLIB_FW_SPEED_LOOP_DEBUG_T_F32 members description...*continued*

Type	Name	Description
tFrac32 *	pUQLim	Pointer to the q-axis voltage controller upper limit (should point to the AMCLIB_CURRENT_LOOP_T_F32 structure element pPIrAWQ.f32UpperLimit). The limit must be a positive value.
tFrac32	f32WReqFilt	SpeedLoop - Filtered value of the required angular velocity.
tFrac32	f32WErr	SpeedLoop - Velocity deviation.
tFrac32	f32WFbckFilt	SpeedLoop - Filtered value of the measured angular velocity.
tFrac32	f32IDQReqAmp	SpeedLoop - Required current amplitude.
tFrac32	f32IQErrSign	FW - Current deviation after sign elimination.
tFrac32	f32FWErr	FW - Current deviation.
tFrac32	f32IQErr	FW - Field weakening feedback control variable.
tFrac32	f32UQErr	FW - Voltage deviation.
tFrac32	f32FWErrFilt	FW - Filtered field weakening feedback control variable.
tFrac32	f32FWAngle	FW - Field weakening angle.
tFrac32	f32FWSin	FW - Q-axis unity current phasor component.
tFrac32	f32FWCos	FW - D-axis unity current phasor component.

5.26 AMCLIB_FW_SPEED_LOOP_DEBUG_T_FLT

FWSpeedLoop configuration structure with debugging information.

5.26.1 Source File

```
#include <AMCLIB_FWSpeedLoop.h>
```

5.26.2 Compound Type Members

Table 473. AMCLIB_FW_SPEED_LOOP_DEBUG_T_FLT members description

Type	Name	Description
GDFLIB_FILTER_MA_T_FLT	pFilterW	Velocity FilterMA paremeters structure.
GDFLIB_FILTER_MA_T_FLT	pFilterFW	Field weakening angle FilterMA paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_FLT	pPIpAWQ	Q-axis ControllerPIpAW paremeters structure.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 473. AMCLIB_FW_SPEED_LOOP_DEBUG_T_FLT members description...*continued*

Type	Name	Description
GFLIB_CONTROLLER_PIAW_P_T_FLT	pPIpAWFW	Field weakening angle ControllerPIpAW paremeters structure.
GFLIB_RAMP_T_FLT	pRamp	Speed ramp paremeters structure.
tFloat *	pIQFbck	Pointer to the q-axis component of the feedback current in the two-phase rotational orthogonal system (d-q).
tFloat *	pUQReq	Pointer to the q-axis component of the required voltage in the two-phase rotational orthogonal system (d-q).
tFloat *	pUQLim	Pointer to the q-axis voltage controller upper limit (should point to the AMCLIB_CURRENT_LOOP_T_FLT structure element pPIrAWQ.fltUpperLimit). The limit must be a positive value.
tFloat	fltUmaxDivImax	Maximal coil voltage divided by the maximal coil current.
tFloat	fltWReqFilt	SpeedLoop - Filtered value of the required angular velocity.
tFloat	fltWErr	SpeedLoop - Velocity deviation.
tFloat	fltIDQReqAmp	SpeedLoop - Filtered value of the measured angular velocity.
tFloat	fltWFbckFilt	SpeedLoop - Required current amplitude.
tFloat	fltIQErrSign	FW - Current deviation after sign elimination.
tFloat	fltFWErr	FW - Current deviation.
tFloat	fltIQErr	FW - Field weakening feedback control variable.
tFloat	fltUQErr	FW - Voltage deviation.
tFloat	fltFWErrFilt	FW - Filtered field weakening feedback control variable.
tFloat	fltFWAngle	FW - Field weakening angle.
tFloat	fltFWSin	FW - Q-axis unity current phasor component.
tFloat	fltFWCos	FW - D-axis unity current phasor component.

5.27 AMCLIB_FW_SPEED_LOOP_T_F16

FWSpeedLoop configuration structure.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.27.1 Source File

```
#include <AMCLIB_FWSpeedLoop.h>
```

5.27.2 Compound Type Members

Table 474. AMCLIB_FW_SPEED_LOOP_T_F16 members description

Type	Name	Description
GDFLIB_FILTER_MA_T_F16	pFilterW	Velocity FilterMA paremeters structure.
GDFLIB_FILTER_MA_T_F16	pFilterFW	Field weakening angle FilterMA paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_F16	pPIpAWQ	Q-axis ControllerPIpAW paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_F16	pPIpAWFW	Field weakening angle ControllerPIpAW paremeters structure.
GFLIB_RAMP_T_F32	pRamp	Speed ramp paremeters structure.
tFrac16 *	pIQFbck	Pointer to the q-axis component of the feedback current in the two-phase rotational orthogonal system (d-q).
tFrac16 *	pUQReq	Pointer to the q-axis component of the required voltage in the two-phase rotational orthogonal system (d-q).
tFrac16 *	pUQLim	Pointer to the q-axis voltage controller upper limit (should point to the AMCLIB_CURRENT_LOOP_T_F16 structure element pPIrAWQ.f16UpperLimit). The limit must be a positive value.

5.28 AMCLIB_FW_SPEED_LOOP_T_F32

FWSpeedLoop configuration structure.

5.28.1 Source File

```
#include <AMCLIB_FWSpeedLoop.h>
```

5.28.2 Compound Type Members

Table 475. AMCLIB_FW_SPEED_LOOP_T_F32 members description

Type	Name	Description
GDFLIB_FILTER_MA_T_F32	pFilterW	Velocity FilterMA paremeters structure.
GDFLIB_FILTER_MA_T_F32	pFilterFW	Field weakening angle FilterMA paremeters structure.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 475. AMCLIB_FW_SPEED_LOOP_T_F32 members description...*continued*

Type	Name	Description
GFLIB_CONTROLLER_PIAW_P_T_F32	pPIpAWQ	Q-axis ControllerPIpAW paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_F32	pPIpAWFW	Field weakening angle ControllerPIpAW paremeters structure.
GFLIB_RAMP_T_F32	pRamp	Speed ramp paremeters structure.
tFrac32 *	pIQFbck	Pointer to the q-axis component of the feedback current in the two-phase rotational orthogonal system (d-q).
tFrac32 *	pUQReq	Pointer to the q-axis component of the required voltage in the two-phase rotational orthogonal system (d-q).
tFrac32 *	pUQLim	Pointer to the q-axis voltage controller upper limit (should point to the AMCLIB_CURRENT_LOOP_T_F32 structure element pPIrAWQ.f32UpperLimit). The limit must be a positive value.

5.29 AMCLIB_FW_SPEED_LOOP_T_FLT

FWSpeedLoop configuration structure.

5.29.1 Source File

```
#include <AMCLIB_FWSpeedLoop.h>
```

5.29.2 Compound Type Members

Table 476. AMCLIB_FW_SPEED_LOOP_T_FLT members description

Type	Name	Description
GDFLIB_FILTER_MA_T_FLT	pFilterW	Velocity FilterMA paremeters structure.
GDFLIB_FILTER_MA_T_FLT	pFilterFW	Field weakening angle FilterMA paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_FLT	pPIpAWQ	Q-axis ControllerPIpAW paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_FLT	pPIpAWFW	Field weakening angle ControllerPIpAW paremeters structure.
GFLIB_RAMP_T_FLT	pRamp	Speed ramp paremeters structure.
tFloat *	pIQFbck	Pointer to the q-axis component of the feedback current in the two-phase rotational orthogonal system (d-q).

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 476. AMCLIB_FW_SPEED_LOOP_T_FLT members description...*continued*

Type	Name	Description
tFloat *	pUQReq	Pointer to the q-axis component of the required voltage in the two-phase rotational orthogonal system (d-q).
tFloat *	pUQLim	Pointer to the q-axis voltage controller upper limit (should point to the AMCLIB_CURRENT_LOOP_T_FLT structure element pPIrAWQ.fltUpperLimit). The limit must be a positive value.
tFloat	fltUmaxDivImax	Maximal coil voltage divided by the maximal coil current.

5.30 AMCLIB_FW_T_F16

FW configuration structure.

5.30.1 Source File

```
#include <AMCLIB_FW.h>
```

5.30.2 Compound Type Members

Table 477. AMCLIB_FW_T_F16 members description

Type	Name	Description
GDFLIB_FILTER_MA_T_F16	pFilterFW	Field weakening angle FilterMA paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_F16	pPIpAWFW	Field weakening angle ControllerPIpAW paremeters structure.
tFrac16 *	pIQFbck	Pointer to the q-axis component of the feedback current in the two-phase rotational orthogonal system (d-q).
tFrac16 *	pUQReq	Pointer to the q-axis component of the required voltage in the two-phase rotational orthogonal system (d-q).
tFrac16 *	pUQLim	Pointer to the q-axis voltage controller upper limit. Must be a positive value.

5.31 AMCLIB_FW_T_F32

FW configuration structure.

5.31.1 Source File

```
#include <AMCLIB_FW.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.31.2 Compound Type Members

Table 478. AMCLIB_FW_T_F32 members description

Type	Name	Description
GDFLIB_FILTER_MA_T_F32	pFilterFW	Field weakening angle FilterMA paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_F32	pPIpAWFW	Field weakening angle ControllerPIpAW paremeters structure.
tFrac32 *	pIQFbck	Pointer to the q-axis component of the feedback current in the two-phase rotational orthogonal system (d-q).
tFrac32 *	pUQReq	Pointer to the q-axis component of the required voltage in the two-phase rotational orthogonal system (d-q).
tFrac32 *	pUQLim	Pointer to the q-axis voltage controller upper limit. Must be a positive value.

5.32 AMCLIB_FW_T_FLT

FW configuration structure.

5.32.1 Source File

```
#include <AMCLIB_FW.h>
```

5.32.2 Compound Type Members

Table 479. AMCLIB_FW_T_FLT members description

Type	Name	Description
GDFLIB_FILTER_MA_T_FLT	pFilterFW	Field weakening angle FilterMA paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_FLT	pPIpAWFW	Field weakening angle ControllerPIpAW paremeters structure.
tFloat *	pIQFbck	Pointer to the q-axis component of the feedback current in the two-phase rotational orthogonal system (d-q).
tFloat *	pUQReq	Pointer to the q-axis component of the required voltage in the two-phase rotational orthogonal system (d-q).

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 479. AMCLIB_FW_T_FLT members description...*continued*

Type	Name	Description
tFloat *	pUQLim	Pointer to the q-axis voltage controller upper limit (should point to the AMCLIB_CURRENT_LOOP_T_FLT structure element pPIrAWQ.fltUpperLimit). The limit must be a positive value.
tFloat	fltUmaxDivImax	Maximal coil voltage divided by the maximal coil current.

5.33 AMCLIB_SPEED_LOOP_DEBUG_T_F16

SpeedLoop configuration structure with debugging information.

5.33.1 Source File

```
#include <AMCLIB_SpeedLoop.h>
```

5.33.2 Compound Type Members

Table 480. AMCLIB_SPEED_LOOP_DEBUG_T_F16 members description

Type	Name	Description
GDFLIB_FILTER_MA_T_F16	pFilterW	Velocity FilterMA paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_F16	pPIpAWQ	Velocity ControllerPIpAW paremeters structure.
GFLIB_RAMP_T_F32	pRamp	Speed ramp paremeters structure.
tFrac16	f16WReqFilt	SpeedLoop - Filtered value of the required angular velocity.
tFrac16	f16WErr	SpeedLoop - Velocity deviation.
tFrac16	f16WFbckFilt	SpeedLoop - Filtered value of the measured angular velocity.

5.34 AMCLIB_SPEED_LOOP_DEBUG_T_F32

SpeedLoop configuration structure with debugging information.

5.34.1 Source File

```
#include <AMCLIB_SpeedLoop.h>
```

5.34.2 Compound Type Members

Table 481. AMCLIB_SPEED_LOOP_DEBUG_T_F32 members description

Type	Name	Description
GDFLIB_FILTER_MA_T_F32	pFilterW	Velocity FilterMA paremeters structure.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 481. AMCLIB_SPEED_LOOP_DEBUG_T_F32 members description...*continued*

Type	Name	Description
GFLIB_CONTROLLER_PIAW_P_T_F32	pPIpAWQ	Velocity ControllerPIpAW paremeters structure.
GFLIB_RAMP_T_F32	pRamp	Speed ramp paremeters structure.
tFrac32	f32WReqFilt	SpeedLoop - Filtered value of the required angular velocity.
tFrac32	f32WErr	SpeedLoop - Velocity deviation.
tFrac32	f32WFbckFilt	SpeedLoop - Filtered value of the measured angular velocity.

5.35 AMCLIB_SPEED_LOOP_DEBUG_T_FLT

SpeedLoop configuration structure with debugging information.

5.35.1 Source File

```
#include <AMCLIB_SpeedLoop.h>
```

5.35.2 Compound Type Members

Table 482. AMCLIB_SPEED_LOOP_DEBUG_T_FLT members description

Type	Name	Description
GDFLIB_FILTER_MA_T_FLT	pFilterW	Velocity FilterMA paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_FLT	pPIpAWQ	Velocity ControllerPIpAW paremeters structure.
GFLIB_RAMP_T_FLT	pRamp	Speed ramp paremeters structure.
tFloat	fltWReqFilt	SpeedLoop - Filtered value of the required angular velocity.
tFloat	fltWErr	SpeedLoop - Velocity deviation.
tFloat	fltWFbckFilt	SpeedLoop - Filtered value of the measured angular velocity.

5.36 AMCLIB_SPEED_LOOP_T_F16

SpeedLoop configuration structure.

5.36.1 Source File

```
#include <AMCLIB_SpeedLoop.h>
```

5.36.2 Compound Type Members

Table 483. AMCLIB_SPEED_LOOP_T_F16 members description

Type	Name	Description
GDFLIB_FILTER_MA_T_F16	pFilterW	Velocity FilterMA paremeters structure.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 483. AMCLIB_SPEED_LOOP_T_F16 members description...*continued*

Type	Name	Description
GFLIB_CONTROLLER_PIAW_P_T_F16	pPIpAWQ	Velocity ControllerPIpAW paremeters structure.
GFLIB_RAMP_T_F32	pRamp	Speed ramp paremeters structure.

5.37 AMCLIB_SPEED_LOOP_T_F32

SpeedLoop configuration structure.

5.37.1 Source File

```
#include <AMCLIB_SpeedLoop.h>
```

5.37.2 Compound Type Members

Table 484. AMCLIB_SPEED_LOOP_T_F32 members description

Type	Name	Description
GDFLIB_FILTER_MA_T_F32	pFilterW	Velocity FilterMA paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_F32	pPIpAWQ	Velocity ControllerPIpAW paremeters structure.
GFLIB_RAMP_T_F32	pRamp	Speed ramp paremeters structure.

5.38 AMCLIB_SPEED_LOOP_T_FLT

SpeedLoop configuration structure.

5.38.1 Source File

```
#include <AMCLIB_SpeedLoop.h>
```

5.38.2 Compound Type Members

Table 485. AMCLIB_SPEED_LOOP_T_FLT members description

Type	Name	Description
GDFLIB_FILTER_MA_T_FLT	pFilterW	Velocity FilterMA paremeters structure.
GFLIB_CONTROLLER_PIAW_P_T_FLT	pPIpAWQ	Velocity ControllerPIpAW paremeters structure.
GFLIB_RAMP_T_FLT	pRamp	Speed ramp paremeters structure.

5.39 AMCLIB_TRACK_OBSRV_T_F16

Structure containing the estimated position, estimated velocity and the algorithm parameters.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.39.1 Source File

```
#include <AMCLIB_TrackObsrv.h>
```

5.39.2 Compound Type Members

Table 486. AMCLIB_TRACK_OBSRV_T_F16 members description

Type	Name	Description
GFLIB_CONTROLLER_PIAW_R_T_F16	pParamPI	Observer PIrAW controller parameters.
GFLIB_INTEGRATOR_TR_T_F16	pParamInteg	Observer integrator parameters.

5.40 AMCLIB_TRACK_OBSRV_T_F32

Structure containing the estimated position, estimated velocity and the algorithm parameters.

5.40.1 Source File

```
#include <AMCLIB_TrackObsrv.h>
```

5.40.2 Compound Type Members

Table 487. AMCLIB_TRACK_OBSRV_T_F32 members description

Type	Name	Description
GFLIB_CONTROLLER_PIAW_R_T_F32	pParamPI	Observer PIrAW controller parameters.
GFLIB_INTEGRATOR_TR_T_F32	pParamInteg	Observer integrator parameters.

5.41 AMCLIB_TRACK_OBSRV_T_FLT

Structure containing the estimated position, estimated velocity and the algorithm parameters.

5.41.1 Source File

```
#include <AMCLIB_TrackObsrv.h>
```

5.41.2 Compound Type Members

Table 488. AMCLIB_TRACK_OBSRV_T_FLT members description

Type	Name	Description
GFLIB_CONTROLLER_PIAW_R_T_FLT	pParamPI	Observer PIrAW controller parameters.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 488. AMCLIB_TRACK_OBSRV_T_FLT members description...*continued*

Type	Name	Description
GFLIB_INTEGRATOR_TR_T_FLT	pParamInteg	Observer integrator parameters.

5.42 AMCLIB_WINDMILLING_T_F16

Observer configuration structure.

5.42.1 Source File

```
#include <AMCLIB_Windmilling.h>
```

5.42.2 Compound Type Members

Table 489. AMCLIB_WINDMILLING_T_F16 members description

Type	Name	Description
tFrac16	f16ADCComp	A/D converter error compensation constant. This parameter is initialized by AMCLIB_WindmillingInit_F16 based on user's input.
AMCLIB_TRACK_OBSRV_T_F16	pParamATO	PI controller and integrator parameters and state. These parameters must be configured by the user before the first call of AMCLIB_Windmilling_F16.
GDFLIB_FILTER_MA_T_F16	pDetMA	PLL lock detector internal state. Do not change. This sub-structure and all its parameters are automatically initialized by AMCLIB_WindmillingInit_F16.
tU8	u8DetNoiseCnt	PLL lock detector internal state. Do not change. This parameter is automatically initialized by AMCLIB_WindmillingInit_F16.
tU8	u8DetSpin	PLL lock detector internal state. Do not change. This parameter is automatically initialized by AMCLIB_WindmillingInit_F16.
tU16	u16DetHystCnt	PLL lock detector internal state. Do not change. This parameter is automatically initialized by AMCLIB_WindmillingInit_F16.
tU16	u16DetStopCnt	PLL lock detector internal state. Do not change. This parameter is automatically initialized by AMCLIB_WindmillingInit_F16.

5.43 AMCLIB_WINDMILLING_T_F32

Observer configuration structure.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.43.1 Source File

```
#include <AMCLIB_Windmilling.h>
```

5.43.2 Compound Type Members

Table 490. AMCLIB_WINDMILLING_T_F32 members description

Type	Name	Description
tFrac32	f32ADCComp	A/D converter error compensation constant. This parameter is initialized by AMCLIB_WindmillingInit_F32 based on user's input.
AMCLIB_TRACK_OBSRV_T_F32	pParamATO	PI controller and integrator parameters and state. These parameters must be configured by the user before the first call of AMCLIB_Windmilling_F32.
GDFLIB_FILTER_MA_T_F32	pDetMA	PLL lock detector internal state. Do not change. This sub-structure and all its parameters are automatically initialized by AMCLIB_WindmillingInit_F32.
tU8	u8DetNoiseCnt	PLL lock detector internal state. Do not change. This parameter is automatically initialized by AMCLIB_WindmillingInit_F32.
tU8	u8DetSpin	PLL lock detector internal state. Do not change. This parameter is automatically initialized by AMCLIB_WindmillingInit_F32.
tU16	u16DetHystCnt	PLL lock detector internal state. Do not change. This parameter is automatically initialized by AMCLIB_WindmillingInit_F32.
tU16	u16DetStopCnt	PLL lock detector internal state. Do not change. This parameter is automatically initialized by AMCLIB_WindmillingInit_F32.

5.44 AMCLIB_WINDMILLING_T_FLT

Observer configuration structure.

5.44.1 Source File

```
#include <AMCLIB_Windmilling.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.44.2 Compound Type Members

Table 491. AMCLIB_WINDMILLING_T_FLT members description

Type	Name	Description
tFloat	fltADCComp	A/D converter error compensation constant. This parameter is initialized by AMCLIB_WindmillingInit_FLT based on user's input.
AMCLIB_TRACK_OBSRV_T_FLT	pParamATO	PI controller and integrator parameters and state. These parameters must be configured by the user before the first call of AMCLIB_Windmilling_FLT.
GDFLIB_FILTER_MA_T_FLT	pDetMA	PLL lock detector internal state. Do not change. This sub-structure and all its parameters are automatically initialized by AMCLIB_WindmillingInit_FLT.
tU8	u8DetNoiseCnt	PLL lock detector internal state. Do not change. This parameter is automatically initialized by AMCLIB_WindmillingInit_FLT.
tU8	u8DetSpin	PLL lock detector internal state. Do not change. This parameter is automatically initialized by AMCLIB_WindmillingInit_FLT.
tU16	u16DetHystCnt	PLL lock detector internal state. Do not change. This parameter is automatically initialized by AMCLIB_WindmillingInit_FLT.
tU16	u16DetStopCnt	PLL lock detector internal state. Do not change. This parameter is automatically initialized by AMCLIB_WindmillingInit_FLT.

5.45 GDFLIB_FILTER_IIR1_COEFF_T_F16

Sub-structure containing filter coefficients.

5.45.1 Source File

```
#include <GDFLIB_FilterIIR1.h>
```

5.45.2 Compound Type Members

Table 492. GDFLIB_FILTER_IIR1_COEFF_T_F16 members description

Type	Name	Description
tFrac16	f16B0	B0 coefficient of an IIR1 filter, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 492. GDFLIB_FILTER_IIR1_COEFF_T_F16 members description...*continued*

Type	Name	Description
tFrac16	f16B1	B1 coefficient of an IIR1 filter, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac16	f16A1	A1 coefficient of an IIR1 filter, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).

5.46 GDFLIB_FILTER_IIR1_COEFF_T_F32

Sub-structure containing filter coefficients.

5.46.1 Source File

```
#include <GDFLIB_FilterIIR1.h>
```

5.46.2 Compound Type Members

Table 493. GDFLIB_FILTER_IIR1_COEFF_T_F32 members description

Type	Name	Description
tFrac32	f32B0	B0 coefficient of an IIR1 filter, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32B1	B1 coefficient of an IIR1 filter, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32A1	A1 coefficient of an IIR1 filter, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).

5.47 GDFLIB_FILTER_IIR1_COEFF_T_FLT

Sub-structure containing filter coefficients.

5.47.1 Source File

```
#include <GDFLIB_FilterIIR1.h>
```

5.47.2 Compound Type Members

Table 494. GDFLIB_FILTER_IIR1_COEFF_T_FLT members description

Type	Name	Description
tFloat	fltB0	B0 coefficient of an IIR1 filter. The parameter is in full range single precision floating point format.
tFloat	fltB1	B1 coefficient of an IIR1 filter. The parameter is in full range single precision floating point format.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 494. GDFLIB_FILTER_IIR1_COEFF_T_FLT members description...*continued*

Type	Name	Description
tFloat	fltA1	A1 coefficient of an IIR1 filter. The parameter is in full range single precision floating point format.

5.48 GDFLIB_FILTER_IIR1_T_F16

Structure containing filter buffer and coefficients.

5.48.1 Source File

```
#include <GDFLIB_FilterIIR1.h>
```

5.48.2 Compound Type Members

Table 495. GDFLIB_FILTER_IIR1_T_F16 members description

Type	Name	Description
GDFLIB_FILTER_IIR1_COEFF_T_F16	trFiltCoeff	Sub-structure containing filter coefficients.
tFrac16	f16FiltBufferX	Input buffer of an IIR1 filter, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac32	f32FiltBufferY	Internal accumulator buffer, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).

5.49 GDFLIB_FILTER_IIR1_T_F32

Structure containing filter buffer and coefficients.

5.49.1 Source File

```
#include <GDFLIB_FilterIIR1.h>
```

5.49.2 Compound Type Members

Table 496. GDFLIB_FILTER_IIR1_T_F32 members description

Type	Name	Description
GDFLIB_FILTER_IIR1_COEFF_T_F32	trFiltCoeff	Sub-structure containing filter coefficients.
tFrac32	f32FiltBufferX	Input buffer of an IIR1 filter, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32FiltBufferY	Internal accumulator buffer, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).

5.50 GDFLIB_FILTER_IIR1_T_FLT

Structure containing filter buffer and coefficients.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.50.1 Source File

```
#include <GDFLIB_FilterIIR1.h>
```

5.50.2 Compound Type Members

Table 497. GDFLIB_FILTER_IIR1_T_FLT members description

Type	Name	Description
GDFLIB_FILTER_IIR1_COEFF_T_FLT	trFiltCoeff	Sub-structure containing filter coefficients.
tFloat	fltFiltBufferX	Input buffer of an IIR1 filter. The input values are in full range single precision floating point format.
tFloat	fltFiltBufferY	Internal accumulator buffer. The values are in full range single precision floating point format.

5.51 GDFLIB_FILTER_IIR2_COEFF_T_F16

Sub-structure containing filter coefficients.

5.51.1 Source File

```
#include <GDFLIB_FilterIIR2.h>
```

5.51.2 Compound Type Members

Table 498. GDFLIB_FILTER_IIR2_COEFF_T_F16 members description

Type	Name	Description
tFrac16	f16B0	B0 coefficient of an IIR2 filter, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac16	f16B1	B1 coefficient of an IIR2 filter, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac16	f16B2	B2 coefficient of an IIR2 filter, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac16	f16A1	A1 coefficient of an IIR2 filter, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac16	f16A2	A2 coefficient of an IIR2 filter, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).

5.52 GDFLIB_FILTER_IIR2_COEFF_T_F32

Sub-structure containing filter coefficients.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.52.1 Source File

```
#include <GDFLIB_FilterIIR2.h>
```

5.52.2 Compound Type Members

Table 499. GDFLIB_FILTER_IIR2_COEFF_T_F32 members description

Type	Name	Description
tFrac32	f32B0	B0 coefficient of an IIR2 filter, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32B1	B1 coefficient of an IIR2 filter, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32B2	B2 coefficient of an IIR2 filter, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32A1	A1 coefficient of an IIR2 filter, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32A2	A2 coefficient of an IIR2 filter, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).

5.53 GDFLIB_FILTER_IIR2_COEFF_T_FLT

Sub-structure containing filter coefficients.

5.53.1 Source File

```
#include <GDFLIB_FilterIIR2.h>
```

5.53.2 Compound Type Members

Table 500. GDFLIB_FILTER_IIR2_COEFF_T_FLT members description

Type	Name	Description
tFloat	fltB0	B0 coefficient of an IIR2 filter. The parameter is in full range single precision floating point format.
tFloat	fltB1	B1 coefficient of an IIR2 filter. The parameter is in full range single precision floating point format.
tFloat	fltB2	B2 coefficient of an IIR2 filter. The parameter is in full range single precision floating point format.
tFloat	fltA1	A1 coefficient of an IIR2 filter. The parameter is in full range single precision floating point format.
tFloat	fltA2	A2 coefficient of an IIR2 filter. The parameter is in full range single precision floating point format.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.54 GDFLIB_FILTER_IIR2_T_F16

Structure containing filter buffer and coefficients.

5.54.1 Source File

```
#include <GDFLIB_FilterIIR2.h>
```

5.54.2 Compound Type Members

Table 501. GDFLIB_FILTER_IIR2_T_F16 members description

Type	Name	Description
GDFLIB_FILTER_IIR2_COEFF_T_F16	trFiltCoeff	Sub-structure containing filter coefficients.
tFrac16	f16FiltBufferX	Input buffer of an IIR2 filter, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac32	f32FiltBufferY	Internal accumulator buffer, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).

5.55 GDFLIB_FILTER_IIR2_T_F32

Structure containing filter buffer and coefficients.

5.55.1 Source File

```
#include <GDFLIB_FilterIIR2.h>
```

5.55.2 Compound Type Members

Table 502. GDFLIB_FILTER_IIR2_T_F32 members description

Type	Name	Description
GDFLIB_FILTER_IIR2_COEFF_T_F32	trFiltCoeff	Sub-structure containing filter coefficients.
tFrac32	f32FiltBufferX	Input buffer of an IIR2 filter, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32FiltBufferY	Internal accumulator buffer, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).

5.56 GDFLIB_FILTER_IIR2_T_FLT

Structure containing filter buffer and coefficients.

5.56.1 Source File

```
#include <GDFLIB_FilterIIR2.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.56.2 Compound Type Members

Table 503. GDFLIB_FILTER_IIR2_T_FLT members description

Type	Name	Description
GDFLIB_FILTER_IIR2_COEFF_T_FLT	trFiltCoeff	Sub-structure containing filter coefficients.
tFloat	fltFiltBufferX	Input buffer of an IIR2 filter. The input values are in full range single precision floating point format.
tFloat	fltFiltBufferY	Internal accumulator buffer. The values are in full range single precision floating point format.

5.57 GDFLIB_FILTER_MA_T_F16

Structure containing filter buffer and coefficients.

5.57.1 Source File

```
#include <GDFLIB_FilterMA.h>
```

5.57.2 Compound Type Members

Table 504. GDFLIB_FILTER_MA_T_F16 members description

Type	Name	Description
tFrac32	f32Acc	State variable - filter accumulator.
tU16	u16NSamples	Recalculated smoothing factor [0, 15].

5.58 GDFLIB_FILTER_MA_T_F32

Structure containing filter buffer and coefficients.

5.58.1 Source File

```
#include <GDFLIB_FilterMA.h>
```

5.58.2 Compound Type Members

Table 505. GDFLIB_FILTER_MA_T_F32 members description

Type	Name	Description
tFrac32	f32Acc	State variable - filter accumulator.
tU16	u16NSamples	Recalculated smoothing factor [0, 31].

5.59 GDFLIB_FILTER_MA_T_FLT

Structure containing filter buffer and coefficients.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.59.1 Source File

```
#include <GDFLIB_FilterMA.h>
```

5.59.2 Compound Type Members

Table 506. GDFLIB_FILTER_MA_T_FLT members description

Type	Name	Description
tFloat	fltAcc	State variable - filter accumulator.
tFloat	fltLambda	Smoothing factor [0, 1].

5.60 GDFLIB_FILTERFIR_PARAM_T_F16

Structure containing parameters of the filter.

5.60.1 Source File

```
#include <GDFLIB_FilterFIR.h>
```

5.60.2 Compound Type Members

Table 507. GDFLIB_FILTERFIR_PARAM_T_F16 members description

Type	Name	Description
tU16	u16Order	FIR filter order, must be in the interval [1, 32767].
const tFrac16 *	pCoefBuf	FIR filter coefficients buffer. The array stores (u16Order + 1) filter coefficients. The array must be aligned to a 32-bit boundary and there must be 2 readable bytes after the last element of the array.

5.61 GDFLIB_FILTERFIR_PARAM_T_F32

Structure containing parameters of the filter.

5.61.1 Source File

```
#include <GDFLIB_FilterFIR.h>
```

5.61.2 Compound Type Members

Table 508. GDFLIB_FILTERFIR_PARAM_T_F32 members description

Type	Name	Description
tU32	u32Order	FIR filter order, must be in the interval [1, 32767].
const tFrac32 *	pCoefBuf	FIR filter coefficients buffer. The array stores (u32Order + 1) filter coefficients.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.62 GDFLIB_FILTERFIR_PARAM_T_FLT

Structure containing parameters of the filter.

5.62.1 Source File

```
#include <GDFLIB_FilterFIR.h>
```

5.62.2 Compound Type Members

Table 509. GDFLIB_FILTERFIR_PARAM_T_FLT members description

Type	Name	Description
tU32	u32Order	FIR filter order, must be in the interval [1, 32767].
const tFloat *	pCoefBuf	FIR filter coefficients buffer. The array stores (u32Order + 1) filter coefficients.

5.63 GDFLIB_FILTERFIR_STATE_T_F16

Structure containing the current state of the filter.

5.63.1 Source File

```
#include <GDFLIB_FilterFIR.h>
```

5.63.2 Compound Type Members

Table 510. GDFLIB_FILTERFIR_STATE_T_F16 members description

Type	Name	Description
tU16	u16Idx	Input buffer index.
tFrac16 *	pInBuf	Pointer to the input buffer. The array stores (u16Order + 1) samples. The array must be aligned to a 32-bit boundary and there must be 2 readable bytes after the last element of the array.

5.64 GDFLIB_FILTERFIR_STATE_T_F32

Structure containing the current state of the filter.

5.64.1 Source File

```
#include <GDFLIB_FilterFIR.h>
```

5.64.2 Compound Type Members

Table 511. GDFLIB_FILTERFIR_STATE_T_F32 members description

Type	Name	Description
tU32	u32Idx	Input buffer index.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 511. GDFLIB_FILTERFIR_STATE_T_F32 members description...*continued*

Type	Name	Description
tFrac32 *	pInBuf	Pointer to the input buffer. The array stores (u32Order + 1) samples.

5.65 GDFLIB_FILTERFIR_STATE_T_FLT

Structure containing the current state of the filter.

5.65.1 Source File

```
#include <GDFLIB_FilterFIR.h>
```

5.65.2 Compound Type Members

Table 512. GDFLIB_FILTERFIR_STATE_T_FLT members description

Type	Name	Description
tU32	u32Idx	Input buffer index.
tFloat *	pInBuf	Pointer to the input buffer. The array stores (u32Order + 1) samples.

5.66 GFLIB_ACOS_T_F16

Default approximation coefficients datatype for arccosine approximation.

5.66.1 Source File

```
#include <GFLIB_Acos.h>
```

5.66.2 Description

Output of arccos(f16In) for interval [0, 1) of the input ratio is divided into two sub-sectors. Polynomial approximation is done using a 5th order polynomial, for each sub-sector respectively. Five coefficients for a single sub-interval are stored in GFLIB_ACOS_TAYLOR_COEF_T_F16 structure.

5.66.3 Compound Type Members

Table 513. GFLIB_ACOS_T_F16 members description

Type	Name	Description
GFLIB_ACOS_TAYLOR_COEF_T_F16	GFLIB_ACOS_SECTOR	Array of two elements for storing two sub-arrays (each sub-array contains five 16-bit coefficients) for all sub-intervals.

5.67 GFLIB_ACOS_T_F32

Default approximation coefficients datatype for arccosine approximation.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.67.1 Source File

```
#include <GFLIB_Acos.h>
```

5.67.2 Description

Output of arccos(f32In) for interval [0, 1) of the input ratio is divided into three sub-sectors. Polynomial approximation is done using a 5th order polynomial, for each sub-sector respectively. Five coefficients for a single sub-interval are stored in GFLIB_ACOS_TAYLOR_COEF_T_F32 structure.

5.67.3 Compound Type Members

Table 514. GFLIB_ACOS_T_F32 members description

Type	Name	Description
GFLIB_ACOS_TAYLOR_COEF_T_F32	GFLIB_ACOS_SECTOR	Array of two elements for storing three sub-arrays (each sub-array contains five 32-bit coefficients) for all sub-intervals.

5.68 GFLIB_ACOS_T_FLT

Default approximation coefficients datatype for arccosine approximation.

5.68.1 Source File

```
#include <GFLIB_Acos.h>
```

5.68.2 Description

The polynomial approximation is done using the square root function of the input parameter. The essential assumption of this approach is a fast H/W based calculation of the square root operation. The approximation polynomial coefficients assume an input argument in the [- π, π] interval.

5.68.3 Compound Type Members

Table 515. GFLIB_ACOS_T_FLT members description

Type	Name	Description
const tFloat	fltA	Array of approximation coefficients.

5.69 GFLIB_ACOS_TAYLOR_COEF_T_F16

Array of approximation coefficients for piece-wise polynomial.

5.69.1 Source File

```
#include <GFLIB_Acos.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.69.2 Description

Output of arccos(f16In) for interval [0, 1) of the input ratio is divided into two sub-sectors. Polynomial approximation is done using a 5th order polynomial, for each sub-sector respectively. Five coefficients for a single sub-interval are stored in this (GFLIB_ACOS_TAYLOR_COEF_T_F16) structure.

5.69.3 Compound Type Members

Table 516. GFLIB_ACOS_TAYLOR_COEF_T_F16 members description

Type	Name	Description
const tFrac16	f16A	Array of five 16-bit elements for storing coefficients of the piece-wise polynomial.

5.70 GFLIB_ACOS_TAYLOR_COEF_T_F32

Array of approximation coefficients for piece-wise polynomial.

5.70.1 Source File

```
#include <GFLIB_Acos.h>
```

5.70.2 Description

Output of arccos(f32In) for interval [0, 1) of the input ratio is divided into three sub-sectors. Polynomial approximation is done using a 5th order polynomial, for each sub-sector respectively. Five coefficients for a single sub-interval are stored in this (GFLIB_ACOS_TAYLOR_COEF_T_F32) structure.

5.70.3 Compound Type Members

Table 517. GFLIB_ACOS_TAYLOR_COEF_T_F32 members description

Type	Name	Description
const tFrac32	f32A	Array of five 32-bit elements for storing coefficients of the piece-wise polynomial.

5.71 GFLIB_ASIN_T_F16

Default approximation coefficients datatype for arcsine approximation.

5.71.1 Source File

```
#include <GFLIB_Asin.h>
```

5.71.2 Description

Output of arcsin(f16In) for interval [0, 1) of the input ratio is divided into two sub-sectors. Polynomial approximation is done using a 5th order polynomial, for each sub-sector respectively. Five coefficients for a single sub-interval are stored in GFLIB_ASIN_TAYLOR_COEF_T_F16 structure.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.71.3 Compound Type Members

Table 518. GFLIB_ASIN_T_F16 members description

Type	Name	Description
GFLIB_ASIN_TAYLOR_COEF_T_F16	GFLIB_ASIN_SECTOR	Default approximation coefficients datatype for arcsine approximation.

5.72 GFLIB_ASIN_T_F32

Default approximation coefficients datatype for arcsine approximation.

5.72.1 Source File

```
#include <GFLIB_Asin.h>
```

5.72.2 Description

Output of arcsin(f32In) for interval [0, 1) of the input ratio is divided into three sub-sectors. Polynomial approximation is done using a 5th order polynomial, for each sub-sector respectively. Five coefficients for a single sub-interval are stored in GFLIB_ASIN_TAYLOR_COEF_T_F32 structure.

5.72.3 Compound Type Members

Table 519. GFLIB_ASIN_T_F32 members description

Type	Name	Description
GFLIB_ASIN_TAYLOR_COEF_T_F32	GFLIB_ASIN_SECTOR	Default approximation coefficients datatype for arcsine approximation.

5.73 GFLIB_ASIN_T_FLT

Default approximation coefficients datatype for arcsine approximation.

5.73.1 Source File

```
#include <GFLIB_Asin.h>
```

5.73.2 Description

The polynomial approximation is done using the square root function of the input parameter. The essential assumption of this approach is a fast H/W based calculation of the square root operation. The approximation polynomial coefficients assume an input argument in the [- π, π] interval.

5.73.3 Compound Type Members

Table 520. GFLIB_ASIN_T_FLT members description

Type	Name	Description
const tFloat	fltA	Default approximation coefficients datatype for arcsine approximation.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.74 GFLIB_ASIN_TAYLOR_COEF_T_F16

Array of approximation coefficients for piece-wise polynomial.

5.74.1 Source File

```
#include <GFLIB_Asin.h>
```

5.74.2 Description

Output of arcsin(f16In) for interval [0, 1) of the input ratio is divided into two sub-sectors. Polynomial approximation is done using a 5th order polynomial, for each sub-sector respectively. Five coefficients for a single sub-interval are stored in this (GFLIB_ASIN_TAYLOR_COEF_T_F16) structure.

5.74.3 Compound Type Members

Table 521. GFLIB_ASIN_TAYLOR_COEF_T_F16 members description

Type	Name	Description
const tFrac16	f16A	Array of approximation coefficients for piece-wise polynomial.

5.75 GFLIB_ASIN_TAYLOR_COEF_T_F32

Array of approximation coefficients for piece-wise polynomial.

5.75.1 Source File

```
#include <GFLIB_Asin.h>
```

5.75.2 Description

Output of arcsin(f32In) for interval [0, 1) of the input ratio is divided into three sub-sectors. Polynomial approximation is done using a 5th order polynomial, for each sub-sector respectively. Five coefficients for a single sub-interval are stored in this (GFLIB_ASIN_TAYLOR_COEF_T_F32) structure.

5.75.3 Compound Type Members

Table 522. GFLIB_ASIN_TAYLOR_COEF_T_F32 members description

Type	Name	Description
const tFrac32	f32A	Array of five 32-bit elements for storing coefficients of the piece-wise polynomial.

5.76 GFLIB_ATAN_T_F16

Structure containing eight sub-structures with polynomial coefficients to cover all sub-intervals.

5.76.1 Source File

```
#include <GFLIB_Atan.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.76.2 Description

Output of arctan(f16In) for interval [0, 1) of the input ratio is divided into eight sub-sectors. Polynomial approximation is done using a 3rd order polynomial, for each sub-sector respectively. Eight arrays, each including three polynomial coefficients for each sub-interval, are stored in this (GFLIB_ATAN_T_F16) structure.

5.76.3 Compound Type Members

Table 523. GFLIB_ATAN_T_F16 members description

Type	Name	Description
const GFLIB_ATAN_TAYLOR_COEF_T_F16	GFLIB_ATAN_SECTOR	Structure containing eight sub-structures with polynomial coefficients to cover all sub-intervals.

5.77 GFLIB_ATAN_T_F32

Structure containing eight sub-structures with polynomial coefficients to cover all sub-intervals.

5.77.1 Source File

```
#include <GFLIB_Atan.h>
```

5.77.2 Description

Output of arctan(f32In) for interval [0, 1) of the input ratio is divided into eight sub-sectors. Polynomial approximation is done using a 3rd order polynomial, for each sub-sector respectively. Eight arrays, each including three polynomial coefficients for each sub-interval, are stored in this (GFLIB_ATAN_T_F32) structure.

5.77.3 Compound Type Members

Table 524. GFLIB_ATAN_T_F32 members description

Type	Name	Description
const GFLIB_ATAN_TAYLOR_COEF_T_F32	GFLIB_ATAN_SECTOR	Structure containing eight sub-structures with polynomial coefficients to cover all sub-intervals.

5.78 GFLIB_ATAN_T_FLT

Structure containing the approximation coefficients.

5.78.1 Source File

```
#include <GFLIB_Atan.h>
```

5.78.2 Description

The approximation uses the rational polynomial approximation which is based on the division of two rational polynomials. The essential assumption of this approach is the

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

fast calculation of a two floating point values division. The approximation polynomial coefficients assume the input argument in the interval (- 2^{128}, 2^{128}).

5.78.3 Compound Type Members

Table 525. GFLIB_ATAN_T_FLT members description

Type	Name	Description
const tFloat	fltA	Structure containing the approximation coefficients.

5.79 GFLIB_ATAN_TAYLOR_COEF_T_F16

Array of polynomial approximation coefficients for one sub-interval.

5.79.1 Source File

```
#include <GFLIB_Atan.h>
```

5.79.2 Description

Output of arctan(f16In) for interval [0, 1) of the input ratio is divided into eight sub-sectors. Polynomial approximation is done using a 3rd order polynomial, for each sub-sector respectively. Three coefficients for a single sub-interval are stored in this (GFLIB_ATAN_TAYLOR_COEF_T_F16) structure.

5.79.3 Compound Type Members

Table 526. GFLIB_ATAN_TAYLOR_COEF_T_F16 members description

Type	Name	Description
const tFrac16	f16A	Array of polynomial approximation coefficients for one sub-interval.

5.80 GFLIB_ATAN_TAYLOR_COEF_T_F32

Array of minimax polynomial approximation coefficients for one sub-interval.

5.80.1 Source File

```
#include <GFLIB_Atan.h>
```

5.80.2 Description

Output of arctan(f32In) for interval [0, 1) of the input ratio is divided into eight sub-sectors. Minimax polynomial approximation is done using a 3rd order polynomial, for each sub-sector respectively. Three coefficients for a single sub-interval are stored in this (GFLIB_ATAN_TAYLOR_COEF_T_F32) structure.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.80.3 Compound Type Members

Table 527. GFLIB_ATAN_TAYLOR_COEF_T_F32 members description

Type	Name	Description
const tFrac32	f32A	Array of minimax polynomial approximation coefficients for one sub-interval.

5.81 GFLIB_ATANYXSHIFTED_T_F16

Structure containing the parameter for the AtanYXShifted function.

5.81.1 Source File

```
#include <GFLIB_AtanYXShifted.h>
```

5.81.2 Compound Type Members

Table 528. GFLIB_ATANYXSHIFTED_T_F16 members description

Type	Name	Description
tFrac16	f16Ky	Multiplication coefficient for the y-signal.
tFrac16	f16Kx	Multiplication coefficient for the x-signal.
tS16	s16Ny	Scaling coefficient for the y-signal.
tS16	s16Nx	Scaling coefficient for the x-signal.
tFrac16	f16ThetaAdj	Adjusting angle.

5.82 GFLIB_ATANYXSHIFTED_T_F32

Structure containing the parameter for the AtanYXShifted function.

5.82.1 Source File

```
#include <GFLIB_AtanYXShifted.h>
```

5.82.2 Compound Type Members

Table 529. GFLIB_ATANYXSHIFTED_T_F32 members description

Type	Name	Description
tFrac32	f32Ky	Multiplication coefficient for the y-signal.
tFrac32	f32Kx	Multiplication coefficient for the x-signal.
tS32	s32Ny	Scaling coefficient for the y-signal.
tS32	s32Nx	Scaling coefficient for the x-signal.
tFrac32	f32ThetaAdj	Adjusting angle.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.83 GFLIB_ATANYXSHIFTED_T_FLT

Structure containing the parameter for the AtanYXShifted function.

5.83.1 Source File

```
#include <GFLIB_AtanYXShifted.h>
```

5.83.2 Compound Type Members

Table 530. GFLIB_ATANYXSHIFTED_T_FLT members description

Type	Name	Description
tFloat	fltKy	Multiplication coefficient for the y-signal.
tFloat	fltKx	Multiplication coefficient for the x-signal.
tFloat	fltThetaAdj	Adjusting angle.

5.84 GFLIB_CONTROLLER_PI_P_T_F16

Structure containing parameters and states of the parallel form PI controller.

5.84.1 Source File

```
#include <GFLIB_ControllerPIp.h>
```

5.84.2 Compound Type Members

Table 531. GFLIB_CONTROLLER_PI_P_T_F16 members description

Type	Name	Description
tFrac16	f16PropGain	Proportional Gain, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac16	f16IntegGain	Integral Gain, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tS16	s16PropGainShift	Proportional Gain Shift, integer format [-15, 15].
tS16	s16IntegGainShift	Integral Gain Shift, integer format [-15, 15].
tFrac32	f32IntegPartK_1	State variable integral part at step k-1.
tFrac16	f16InK_1	State variable input error at step k-1.

5.85 GFLIB_CONTROLLER_PI_P_T_F32

Structure containing parameters and states of the parallel form PI controller.

5.85.1 Source File

```
#include <GFLIB_ControllerPIp.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.85.2 Compound Type Members

Table 532. GFLIB_CONTROLLER_PI_P_T_F32 members description

Type	Name	Description
tFrac32	f32PropGain	Proportional Gain, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32IntegGain	Integral Gain, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tS16	s16PropGainShift	Proportional Gain Shift, integer format [-31, 31].
tS16	s16IntegGainShift	Integral Gain Shift, integer format [-31, 31].
tFrac32	f32IntegPartK_1	State variable integral part at step k-1.
tFrac32	f32InK_1	State variable input error at step k-1.

5.86 GFLIB_CONTROLLER_PI_P_T_FLT

Structure containing parameters and states of the parallel form PI controller.

5.86.1 Source File

```
#include <GFLIB_ControllerPIp.h>
```

5.86.2 Compound Type Members

Table 533. GFLIB_CONTROLLER_PI_P_T_FLT members description

Type	Name	Description
tFloat	fltPropGain	Proportional Gain, single precision floating point format.
tFloat	fltIntegGain	Integral Gain, single precision floating point format.
tFloat	fltIntegPartK_1	State variable integral part at step k-1, single precision floating point format.
tFloat	fltInK_1	State variable input error at step k-1, single precision floating point format.

5.87 GFLIB_CONTROLLER_PI_R_T_F16

Structure containing parameters and states of the recurrent form PI controller.

5.87.1 Source File

```
#include <GFLIB_ControllerPIr.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.87.2 Compound Type Members

Table 534. GFLIB_CONTROLLER_PI_R_T_F16 members description

Type	Name	Description
tFrac16	f16CC1sc	CC1 coefficient, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac16	f16CC2sc	CC2 coefficient, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac32	f32Acc	State variable - internal controller accumulator.
tFrac16	f16InErrK1	State variable - controller input from the previous calculation step.
tU16	u16NShift	Scaling bitwise shift applied to the controller coefficients, integer format [0, 15].

5.88 GFLIB_CONTROLLER_PI_R_T_F32

Structure containing parameters and states of the recurrent form PI controller.

5.88.1 Source File

```
#include <GFLIB_ControllerPIr.h>
```

5.88.2 Compound Type Members

Table 535. GFLIB_CONTROLLER_PI_R_T_F32 members description

Type	Name	Description
tFrac32	f32CC1sc	CC1 coefficient, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32CC2sc	CC2 coefficient, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32Acc	State variable - internal controller accumulator.
tFrac32	f32InErrK1	State variable - controller input from the previous calculation step.
tU16	u16NShift	Scaling bitwise shift applied to the controller coefficients, integer format [0, 31].

5.89 GFLIB_CONTROLLER_PI_R_T_FLT

Structure containing parameters and states of the recurrent form PI controller.

5.89.1 Source File

```
#include <GFLIB_ControllerPIr.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.89.2 Compound Type Members

Table 536. GFLIB_CONTROLLER_PI_R_T_FLT members description

Type	Name	Description
tFloat	fltCC1sc	CC1 coefficient, single precision floating point format.
tFloat	fltCC2sc	CC2 coefficient, single precision floating point format.
tFloat	fltAcc	State variable - internal controller accumulator, single precision floating point format.
tFloat	fltInErrK1	State variable - controller input from the previous calculation step, single precision floating point format.

5.90 GFLIB_CONTROLLER_PIAW_P_T_F16

Structure containing parameters and states of the parallel form PI controller with anti-windup.

5.90.1 Source File

```
#include <GFLIB_ControllerPIpAW.h>
```

5.90.2 Compound Type Members

Table 537. GFLIB_CONTROLLER_PIAW_P_T_F16 members description

Type	Name	Description
tFrac16	f16PropGain	Proportional Gain, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac16	f16IntegGain	Integral Gain, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tS16	s16PropGainShift	Proportional Gain Shift, integer format [-15, 15].
tS16	s16IntegGainShift	Integral Gain Shift, integer format [-15, 15].
tFrac16	f16LowerLimit	Lower Limit of the controller, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac16	f16UpperLimit	Upper Limit of the controller, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac32	f32IntegPartK_1	State variable integral part at step k-1.
tFrac16	f16InK_1	State variable input error at step k-1.
tU16	u16LimitFlag	Limitation flag, if set to 1, the controller output has reached either the UpperLimit or LowerLimit.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.91 GFLIB_CONTROLLER_PIAW_P_T_F32

Structure containing parameters and states of the parallel form PI controller with anti-windup.

5.91.1 Source File

```
#include <GFLIB_ControllerPIpAW.h>
```

5.91.2 Compound Type Members

Table 538. GFLIB_CONTROLLER_PIAW_P_T_F32 members description

Type	Name	Description
tFrac32	f32PropGain	Proportional Gain, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32IntegGain	Integral Gain, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tS16	s16PropGainShift	Proportional Gain Shift, integer format [-31, 31].
tS16	s16IntegGainShift	Integral Gain Shift, integer format [-31, 31].
tFrac32	f32LowerLimit	Lower Limit of the controller, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32UpperLimit	Upper Limit of the controller, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32IntegPartK_1	State variable integral part at step k-1.
tFrac32	f32InK_1	State variable input error at step k-1.
tU16	u16LimitFlag	Limitation flag, if set to 1, the controller output has reached either the UpperLimit or LowerLimit.

5.92 GFLIB_CONTROLLER_PIAW_P_T_FLT

Structure containing parameters and states of the parallel form PI controller with anti-windup.

5.92.1 Source File

```
#include <GFLIB_ControllerPIpAW.h>
```

5.92.2 Compound Type Members

Table 539. GFLIB_CONTROLLER_PIAW_P_T_FLT members description

Type	Name	Description
tFloat	fltPropGain	Proportional Gain, single precision floating point format.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 539. GFLIB_CONTROLLER_PIAW_P_T_FLT members description...*continued*

Type	Name	Description
tFloat	fltIntegGain	Integral Gain, single precision floating point format.
tFloat	fltLowerLimit	Lower Limit of the controller, single precision floating point format.
tFloat	fltUpperLimit	Upper Limit of the controller, single precision floating point format.
tFloat	fltIntegPartK_1	State variable integral part at step k-1, single precision floating point format.
tFloat	fltInK_1	State variable input error at step k-1, single precision floating point format.
tU16	u16LimitFlag	Limitation flag, if set to 1, the controller output has reached either the UpperLimit or LowerLimit.

5.93 GFLIB_CONTROLLER_PIAW_R_T_F16

Structure containing parameters and states of the recurrent form PI controller with anti-windup.

5.93.1 Source File

```
#include <GFLIB_ControllerPIrAW.h>
```

5.93.2 Compound Type Members

Table 540. GFLIB_CONTROLLER_PIAW_R_T_F16 members description

Type	Name	Description
tFrac16	f16CC1sc	CC1 coefficient, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac16	f16CC2sc	CC2 coefficient, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac32	f32Acc	State variable - internal controller accumulator.
tFrac16	f16InErrK1	State variable - controller input from the previous calculation step.
tFrac16	f16UpperLimit	Upper Limit of the controller, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac16	f16LowerLimit	Lower Limit of the controller, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tU16	u16NShift	Scaling bitwise shift applied to the controller coefficients, integer format [0, 15].

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.94 GFLIB_CONTROLLER_PIAW_R_T_F32

Structure containing parameters and states of the recurrent form PI controller with anti-windup.

5.94.1 Source File

```
#include <GFLIB_ControllerPIrAW.h>
```

5.94.2 Compound Type Members

Table 541. GFLIB_CONTROLLER_PIAW_R_T_F32 members description

Type	Name	Description
tFrac32	f32CC1sc	CC1 coefficient, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32CC2sc	CC2 coefficient, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32Acc	State variable - internal controller accumulator.
tFrac32	f32InErrK1	State variable - controller input from the previous calculation step.
tFrac32	f32UpperLimit	Upper Limit of the controller, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32LowerLimit	Lower Limit of the controller, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tU16	u16NShift	Scaling bitwise shift applied to the controller coefficients, integer format [0, 31].

5.95 GFLIB_CONTROLLER_PIAW_R_T_FLT

Structure containing parameters and states of the recurrent form PI controller with anti-windup.

5.95.1 Source File

```
#include <GFLIB_ControllerPIrAW.h>
```

5.95.2 Compound Type Members

Table 542. GFLIB_CONTROLLER_PIAW_R_T_FLT members description

Type	Name	Description
tFloat	fltCC1sc	CC1 coefficient, single precision floating point format.
tFloat	fltCC2sc	CC2 coefficient, single precision floating point format.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 542. GFLIB_CONTROLLER_PIAW_R_T_FLT members description...*continued*

Type	Name	Description
tFloat	fltAcc	State variable - internal controller accumulator, single precision floating point format.
tFloat	fltInErrK1	State variable - controller input from the previous calculation step, single precision floating point format.
tFloat	fltUpperLimit	Upper Limit of the controller, single precision floating point format.
tFloat	fltLowerLimit	Lower Limit of the controller, single precision floating point format.

5.96 GFLIB_CONTROLLER_PID_P_AW_T_F16

Structure containing parameters and states of the parallel form PI controller.

5.96.1 Source File

```
#include <GFLIB_ControllerPIDpAW.h>
```

5.96.2 Compound Type Members

Table 543. GFLIB_CONTROLLER_PID_P_AW_T_F16 members description

Type	Name	Description
tFrac16	f16PropGain	Proportional Gain, fractional format normalized to fit into (0, 2^{15}-1).
tFrac16	f16IntegGain	Integral Gain, fractional format normalized to fit into (0, 2^{15}-1).
tFrac16	f16DerivGain	Derivative Gain, fractional format normalized to fit into (0, 2^{15}-1).
tFrac16	f16FiltCoef	Derivative term filter coefficient, fractional format in range (0, 2^{15}-1).
tS16	s16PropGainShift	Proportional Gain Shift, integer format [-15, 15].
tS16	s16IntegGainShift	Integral Gain Shift, integer format [-15, 15].
tS16	s16DerivGainShift	Derivative Gain Shift, integer format [-15, 15].
tFrac16	f16LowerLimit	Lower Limit of the controller, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac16	f16UpperLimit	Upper Limit of the controller, fractional format normalized to fit into (- 2^{15}, 2^{15}-1).
tFrac32	f32IntegPartK_1	State variable integral part at step k-1.
tFrac16	f16DerivPartK_1	State variable integral part at step k-1.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 543. GFLIB_CONTROLLER_PID_P_AW_T_F16 members description...*continued*

Type	Name	Description
tFrac16	f16InK_1	State variable input error at step k-1.
tU16	u16LimitFlag	Limitation flag, if set to 1, the controller output has reached either the UpperLimit or LowerLimit.

5.97 GFLIB_CONTROLLER_PID_P_AW_T_F32

Structure containing parameters and states of the PID controller.

5.97.1 Source File

```
#include <GFLIB_ControllerPIDpAW.h>
```

5.97.2 Compound Type Members

Table 544. GFLIB_CONTROLLER_PID_P_AW_T_F32 members description

Type	Name	Description
tFrac32	f32PropGain	Proportional Gain, fractional format normalized to fit into (0, 2^{31}-1).
tFrac32	f32IntegGain	Integral Gain, fractional format normalized to fit into (0, 2^{31}-1).
tFrac32	f32DerivGain	Derivative Gain, fractional format normalized to fit into (0, 2^{31}-1).
tFrac32	f32FiltCoef	Derivative term filter coefficient, fractional format in range (0, 2^{31}-1).
tS16	s16PropGainShift	Proportional Gain Shift, integer format [-31, 31].
tS16	s16IntegGainShift	Integral Gain Shift, integer format [-31, 31].
tS16	s16DerivGainShift	Derivative Gain Shift, integer format [-31, 31].
tFrac32	f32LowerLimit	Lower Limit of the controller, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32UpperLimit	Upper Limit of the controller, fractional format normalized to fit into (- 2^{31}, 2^{31}-1).
tFrac32	f32IntegPartK_1	State variable integral part at step k-1.
tFrac32	f32DerivPartK_1	State variable derivative part filter at step k-1.
tFrac32	f32InK_1	State variable input error at step k-1.
tU16	u16LimitFlag	Limitation flag, if set to 1, the controller output has reached either the UpperLimit or LowerLimit.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.98 GFLIB_CONTROLLER_PID_P_AW_T_FLT

Structure containing parameters and states of the parallel form PI controller.

5.98.1 Source File

```
#include <GFLIB_ControllerPIDpAW.h>
```

5.98.2 Compound Type Members

Table 545. GFLIB_CONTROLLER_PID_P_AW_T_FLT members description

Type	Name	Description
tFloat	fltPropGain	Proportional Gain, positive number in single precision floating point format.
tFloat	fltIntegGain	Integral Gain, positive number in single precision floating point format.
tFloat	fltDerivGain	Derivative Gain, positive number in single precision floating point format.
tFloat	fltFiltCoef	Derivative term filter coefficient, single precision floating point format.
tFloat	fltLowerLimit	Lower Limit of the controller, single precision floating point format.
tFloat	fltUpperLimit	Upper Limit of the controller, single precision floating point format.
tFloat	fltIntegPartK_1	State variable integral part at step k-1, single precision floating point format.
tFloat	fltDerivPartK_1	State variable integral part at step k-1, single precision floating point format.
tFloat	fltInK_1	State variable input error at step k-1, single precision floating point format.
tU16	u16LimitFlag	Limitation flag, if set to 1, the controller output has reached either the UpperLimit or LowerLimit.

5.99 GFLIB_COS_T_F16

Array of four 16-bit elements for storing coefficients of the Taylor polynomial.

5.99.1 Source File

```
#include <GFLIB_Cos.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.99.2 Compound Type Members

Table 546. GFLIB_COS_T_F16 members description

Type	Name	Description
tFrac16	f16A	Array of four 16-bit elements for storing coefficients of the Taylor polynomial.

5.100 GFLIB_COS_T_F32

Array of five 32-bit elements for storing coefficients of the Taylor polynomial.

5.100.1 Source File

```
#include <GFLIB_Cos.h>
```

5.100.2 Compound Type Members

Table 547. GFLIB_COS_T_F32 members description

Type	Name	Description
tFrac32	f32A	Array of five 32-bit elements for storing coefficients of the Taylor polynomial.

5.101 GFLIB_COS_T_FLT

Array of three single precision floating point elements for storing coefficients of the floating point optimized minimax approximation polynomial.

5.101.1 Source File

```
#include <GFLIB_Cos.h>
```

5.101.2 Compound Type Members

Table 548. GFLIB_COS_T_FLT members description

Type	Name	Description
tFloat	fltA	Array of three single precision floating point elements for storing coefficients of the floating point optimized minimax approximation polynomial.

5.102 GFLIB_HYST_T_F16

Structure containing parameters and states for the hysteresis function.

5.102.1 Source File

```
#include <GFLIB_Hyst.h>
```


© NXP B.V. 2023. All rights reserved.

5.102.2 Compound Type Members

Table 549. GFLIB_HYST_T_F16 members description

Type	Name	Description
tFrac16	f16HystOn	Value determining the upper threshold.
tFrac16	f16HystOff	Value determining the lower threshold.
tFrac16	f16OutValOn	Value of the output when input is higher than the upper threshold.
tFrac16	f16OutValOff	Value of the output when input is lower than the lower threshold.
tFrac16	f16OutState	Actual state of the output.

5.103 GFLIB_HYST_T_F32

Structure containing parameters and states for the hysteresis function.

5.103.1 Source File

```
#include <GFLIB_Hyst.h>
```

5.103.2 Compound Type Members

Table 550. GFLIB_HYST_T_F32 members description

Type	Name	Description
tFrac32	f32HystOn	Value determining the upper threshold.
tFrac32	f32HystOff	Value determining the lower threshold.
tFrac32	f32OutValOn	Value of the output when input is higher than the upper threshold.
tFrac32	f32OutValOff	Value of the output when input is lower than the lower threshold.
tFrac32	f32OutState	Actual state of the output.

5.104 GFLIB_HYST_T_FLT

Structure containing parameters and states for the hysteresis function.

5.104.1 Source File

```
#include <GFLIB_Hyst.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.104.2 Compound Type Members

Table 551. GFLIB_HYST_T_FLT members description

Type	Name	Description
tFloat	fltHystOn	Value determining the upper threshold.
tFloat	fltHystOff	Value determining the lower threshold.
tFloat	fltOutValOn	Value of the output when input is higher than the upper threshold.
tFloat	fltOutValOff	Value of the output when input is lower than the lower threshold.
tFloat	fltOutState	Actual state of the output.

5.105 GFLIB_INTEGRATOR_TR_T_F16

Structure containing integrator parameters and coefficients.

5.105.1 Source File

```
#include <GFLIB_IntegratorTR.h>
```

5.105.2 Compound Type Members

Table 552. GFLIB_INTEGRATOR_TR_T_F16 members description

Type	Name	Description
tFrac32	f32State	State variable - integrator state value.
tFrac16	f16InK1	State variable - input value in step k-1.
tFrac16	f16C1	Integrator coefficient = (E_{MAX}/ T_s) (U_{MAX}*2)*($2^{-u16NShift}$).
tU16	u16NShift	Scaling bitwise shift applied to the integrator coefficient f16C1, integer format [0, 15].

5.106 GFLIB_INTEGRATOR_TR_T_F32

Structure containing integrator parameters and coefficients.

5.106.1 Source File

```
#include <GFLIB_IntegratorTR.h>
```

5.106.2 Compound Type Members

Table 553. GFLIB_INTEGRATOR_TR_T_F32 members description

Type	Name	Description
tFrac32	f32State	State variable - integrator state value.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 553. GFLIB_INTEGRATOR_TR_T_F32 members description...*continued*

Type	Name	Description
tFrac32	f32InK1	State variable - input value in step k-1.
tFrac32	f32C1	Integrator coefficient = (E_{MAX}/ T_s) (U_{MAX}*2)*($2^{-u16NShift}$).
tU16	u16NShift	Scaling bitwise shift applied to the integrator coefficient f32C1, integer format [0, 15].

5.107 GFLIB_INTEGRATOR_TR_T_FLT

Structure containing integrator parameters and coefficients.

5.107.1 Source File

```
#include <GFLIB_IntegratorTR.h>
```

5.107.2 Compound Type Members

Table 554. GFLIB_INTEGRATOR_TR_T_FLT members description

Type	Name	Description
tFloat	fltState	State variable - integrator state value, single precision floating point format.
tFloat	fltInK1	State variable - input value in step k-1, single precision floating point format.
tFloat	fltC1	Integrator coefficient, single precision floating point format.

5.108 GFLIB_LIMIT_T_F16

Structure containing the limits.

5.108.1 Source File

```
#include <GFLIB_Limit.h>
```

5.108.2 Compound Type Members

Table 555. GFLIB_LIMIT_T_F16 members description

Type	Name	Description
tFrac16	f16LowerLimit	Value determining the lower limit threshold.
tFrac16	f16UpperLimit	Value determining the upper limit threshold.

5.109 GFLIB_LIMIT_T_F32

Structure containing the limits.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.109.1 Source File

```
#include <GFLIB_Limit.h>
```

5.109.2 Compound Type Members

Table 556. GFLIB_LIMIT_T_F32 members description

Type	Name	Description
tFrac32	f32LowerLimit	Value determining the lower limit threshold.
tFrac32	f32UpperLimit	Value determining the upper limit threshold.

5.110 GFLIB_LIMIT_T_FLT

Structure containing the limits.

5.110.1 Source File

```
#include <GFLIB_Limit.h>
```

5.110.2 Compound Type Members

Table 557. GFLIB_LIMIT_T_FLT members description

Type	Name	Description
tFloat	fltLowerLimit	Value determining the lower limit threshold.
tFloat	fltUpperLimit	Value determining the upper limit threshold.

5.111 GFLIB_LOG10_T_FLT

Array of single precision floating point elements for storing the coefficients of the floating point log10 approximation polynomial.

5.111.1 Source File

```
#include <GFLIB_Log10.h>
```

5.111.2 Compound Type Members

Table 558. GFLIB_LOG10_T_FLT members description

Type	Name	Description
tFloat	fltA	Array of single precision floating point elements for storing the coefficients of the floating point log10 approximation polynomial.

5.112 GFLIB_LOWERLIMIT_T_F16

Structure containing the lower limit.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.112.1 Source File

```
#include <GFLIB_LowerLimit.h>
```

5.112.2 Compound Type Members

Table 559. GFLIB_LOWERLIMIT_T_F16 members description

Type	Name	Description
tFrac16	f16LowerLimit	Value determining the lower limit threshold.

5.113 GFLIB_LOWERLIMIT_T_F32

Structure containing the lower limit.

5.113.1 Source File

```
#include <GFLIB_LowerLimit.h>
```

5.113.2 Compound Type Members

Table 560. GFLIB_LOWERLIMIT_T_F32 members description

Type	Name	Description
tFrac32	f32LowerLimit	Value determining the lower limit threshold.

5.114 GFLIB_LOWERLIMIT_T_FLT

Structure containing the lower limit.

5.114.1 Source File

```
#include <GFLIB_LowerLimit.h>
```

5.114.2 Compound Type Members

Table 561. GFLIB_LOWERLIMIT_T_FLT members description

Type	Name	Description
tFloat	fltLowerLimit	Value determining the lower limit threshold.

5.115 GFLIB_LUT1D_T_F16

Structure containing 1D look-up table parameters.

5.115.1 Source File

```
#include <GFLIB_Lut1D.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.115.2 Compound Type Members

Table 562. GFLIB_LUT1D_T_F16 members description

Type	Name	Description
tU16	u16ShamOffset	u16ShamOffset = log2(N - 1), where N is the table length, e.g. {N = 3, u16ShamOffset = 1}, {N = 5, u16ShamOffset = 2}, {N = 9, u16ShamOffset = 3}, {N = 17, u16ShamOffset = 4}, etc.
const tFrac16 *	pf16Table	Pointer to a look-up table holding 16-bit fractional ordinate values at evenly spaced intervals.

5.116 GFLIB_LUT1D_T_F32

Structure containing 1D look-up table parameters.

5.116.1 Source File

```
#include <GFLIB_Lut1D.h>
```

5.116.2 Compound Type Members

Table 563. GFLIB_LUT1D_T_F32 members description

Type	Name	Description
tU32	u32ShamOffset	u32ShamOffset = log2(N - 1), where N is the table length, e.g. {N = 3, u32ShamOffset = 1}, {N = 5, u32ShamOffset = 2}, {N = 9, u32ShamOffset = 3}, {N = 17, u32ShamOffset = 4}, etc.
const tFrac32 *	pf32Table	Pointer to a look-up table holding 32-bit fractional ordinate values at evenly spaced intervals.

5.117 GFLIB_LUT1D_T_FLT

Structure containing 1D look-up table parameters.

5.117.1 Source File

```
#include <GFLIB_Lut1D.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.117.2 Compound Type Members

Table 564. GFLIB_LUT1D_T_FLT members description

Type	Name	Description
tU32	u32ShamOffset	u32ShamOffset = log2(N - 1), where N is the table length, e.g. {N = 3, u32ShamOffset = 1}, {N = 5, u32ShamOffset = 2}, {N = 9, u32ShamOffset = 3}, {N = 17, u32ShamOffset = 4}, etc.
const tFloat *	pfltTable	Pointer to a look-up table holding single precision floating-point ordinate values at evenly spaced intervals.

5.118 GFLIB_LUT2D_T_F16

Structure containing 2D look-up table parameters.

5.118.1 Source File

```
#include <GFLIB_Lut2D.h>
```

5.118.2 Compound Type Members

Table 565. GFLIB_LUT2D_T_F16 members description

Type	Name	Description
tU16	u16ShamOffset1	u16ShamOffset1 = log2(N - 1), where N is number of columns in the table, e.g. {N = 3, u16ShamOffset1 = 1}, {N = 5, u16ShamOffset1 = 2}, {N = 9, u16ShamOffset1 = 3}, {N = 17, u16ShamOffset1 = 4}, etc.
tU16	u16ShamOffset2	u16ShamOffset2 = log2(M - 1), where M is number of rows in the table, e.g. {M = 3, u16ShamOffset2 = 1}, {M = 5, u16ShamOffset2 = 2}, {M = 9, u16ShamOffset2 = 3}, {M = 17, u16ShamOffset2 = 4}, etc.
const tFrac16 *	pf16Table	Pointer to a look-up table holding 16-bit fractional values arranged in a uniform grid. The table is stored in column-major format.

5.119 GFLIB_LUT2D_T_F32

Structure containing 2D look-up table parameters.

5.119.1 Source File

```
#include <GFLIB_Lut2D.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.119.2 Compound Type Members

Table 566. GFLIB_LUT2D_T_F32 members description

Type	Name	Description
tU32	u32ShamOffset1	u32ShamOffset1 = log2(N - 1), where N is number of columns in the table, e.g. {N = 3, u32ShamOffset1 = 1}, {N = 5, u32ShamOffset1 = 2}, {N = 9, u32ShamOffset1 = 3}, {N = 17, u32ShamOffset1 = 4}, etc.
tU32	u32ShamOffset2	u32ShamOffset2 = log2(M - 1), where M is number of rows in the table, e.g. {M = 3, u32ShamOffset2 = 1}, {M = 5, u32ShamOffset2 = 2}, {M = 9, u32ShamOffset2 = 3}, {M = 17, u32ShamOffset2 = 4}, etc.
const tFrac32 *	pf32Table	Pointer to a look-up table holding 32-bit fractional values arranged in a uniform grid. The table is stored in column-major format.

5.120 GFLIB_LUT2D_T_FLT

Structure containing 2D look-up table parameters.

5.120.1 Source File

```
#include <GFLIB_Lut2D.h>
```

5.120.2 Compound Type Members

Table 567. GFLIB_LUT2D_T_FLT members description

Type	Name	Description
tU32	u32ShamOffset1	u32ShamOffset1 = log2(N - 1), where N is number of columns in the table, e.g. {N = 3, u32ShamOffset1 = 1}, {N = 5, u32ShamOffset1 = 2}, {N = 9, u32ShamOffset1 = 3}, {N = 17, u32ShamOffset1 = 4}, etc.
tU32	u32ShamOffset2	u32ShamOffset2 = log2(M - 1), where M is number of rows in the table, e.g. {M = 3, u32ShamOffset2 = 1}, {M = 5, u32ShamOffset2 = 2}, {M = 9, u32ShamOffset2 = 3}, {M = 17, u32ShamOffset2 = 4}, etc.
const tFloat *	pfltTable	Pointer to a look-up table holding single precision floating-point values arranged in a uniform grid. The table is stored in column-major format.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.121 GFLIB_RAMP_T_F16

Structure containing increment/decrement coefficients and state value for the ramp function implemented in GFLIB_Ramp.

5.121.1 Source File

```
#include <GFLIB_Ramp.h>
```

5.121.2 Compound Type Members

Table 568. GFLIB_RAMP_T_F16 members description

Type	Name	Description
tFrac16	f16State	Ramp state value.
tFrac16	f16RampUp	Ramp up increment coefficient.
tFrac16	f16RampDown	Ramp decrement coefficient.

5.122 GFLIB_RAMP_T_F32

Structure containing increment/decrement coefficients and state value for the ramp function implemented in GFLIB_Ramp.

5.122.1 Source File

```
#include <GFLIB_Ramp.h>
```

5.122.2 Compound Type Members

Table 569. GFLIB_RAMP_T_F32 members description

Type	Name	Description
tFrac32	f32State	Ramp state value.
tFrac32	f32RampUp	Ramp up increment coefficient.
tFrac32	f32RampDown	Ramp decrement coefficient.

5.123 GFLIB_RAMP_T_FLT

Structure containing increment/decrement coefficients and state value for the ramp function implemented in GFLIB_Ramp.

5.123.1 Source File

```
#include <GFLIB_Ramp.h>
```

5.123.2 Compound Type Members

Table 570. GFLIB_RAMP_T_FLT members description

Type	Name	Description
tFloat	fltState	Ramp state value.
tFloat	fltRampUp	Ramp up increment coefficient.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 570. GFLIB_RAMP_T_FLT members description...*continued*

Type	Name	Description
tFloat	fltRampDown	Ramp decrement coefficient.

5.124 GFLIB_SIN_T_F16

Array of four 16-bit elements for storing coefficients of the Taylor polynomial.

5.124.1 Source File

```
#include <GFLIB_Sin.h>
```

5.124.2 Compound Type Members

Table 571. GFLIB_SIN_T_F16 members description

Type	Name	Description
tFrac16	f16A	Array of four 16-bit elements for storing coefficients of the Taylor polynomial.

5.125 GFLIB_SIN_T_F32

Array of five 32-bit elements for storing coefficients of the Taylor polynomial.

5.125.1 Source File

```
#include <GFLIB_Sin.h>
```

5.125.2 Compound Type Members

Table 572. GFLIB_SIN_T_F32 members description

Type	Name	Description
tFrac32	f32A	Array of five 32-bit elements for storing coefficients of the Taylor polynomial.

5.126 GFLIB_SIN_T_FLT

Array of three single precision floating point elements for storing coefficients of the floating point optimized minimax approximation polynomial.

5.126.1 Source File

```
#include <GFLIB_Sin.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.126.2 Compound Type Members

Table 573. GFLIB_SIN_T_FLT members description

Type	Name	Description
tFloat	fltA	Array of three single precision floating point elements for storing coefficients of the floating point optimized minimax approximation polynomial.

5.127 GFLIB_SINCOS_T_F16

Array of four 16-bit elements for storing coefficients of the Taylor polynomial.

5.127.1 Source File

```
#include <GFLIB_SinCos.h>
```

5.127.2 Compound Type Members

Table 574. GFLIB_SINCOS_T_F16 members description

Type	Name	Description
tFrac16	f16A	Array of four 16-bit elements for storing coefficients of the Taylor polynomial.

5.128 GFLIB_SINCOS_T_F32

Array of five 32-bit elements for storing coefficients of the Taylor polynomial.

5.128.1 Source File

```
#include <GFLIB_SinCos.h>
```

5.128.2 Compound Type Members

Table 575. GFLIB_SINCOS_T_F32 members description

Type	Name	Description
tFrac32	f32A	Array of five 32-bit elements for storing coefficients of the Taylor polynomial.

5.129 GFLIB_SINCOS_T_FLT

Array of three single precision floating point elements for storing coefficients of the floating point optimized minimax approximation polynomial.

5.129.1 Source File

```
#include <GFLIB_SinCos.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.129.2 Compound Type Members

Table 576. GFLIB_SINCOS_T_FLT members description

Type	Name	Description
tFloat	fltA	Array of three single precision floating point elements for storing coefficients of the floating point optimized minimax approximation polynomial.

5.130 GFLIB_TAN_T_F16

Output of tan(PI * f16In) for interval [0, PI/4) of the input angles is divided into eight sub-sectors. Polynomial approximation is done using a 4th order polynomial, for each sub-sector respectively. Eight arrays, each including four polynomial coefficients for each sub-interval, are stored in this (GFLIB_TAN_T_F16) structure.

5.130.1 Source File

```
#include <GFLIB_Tan.h>
```

5.130.2 Compound Type Members

Table 577. GFLIB_TAN_T_F16 members description

Type	Name	Description
GFLIB_TAN_TAYLOR_COEF_T_F16	GFLIB_TAN_SECTOR	Output of tan(PI * f16In) for interval [0, PI/4) of the input angles is divided into eight sub-sectors. Polynomial approximation is done using a 4th order polynomial, for each sub-sector respectively. Eight arrays, each including four polynomial coefficients for each sub-interval, are stored in this (GFLIB_TAN_T_F16) structure.

5.131 GFLIB_TAN_T_F32

Output of tan(PI * f32In) for interval [0, PI/4) of the input angles is divided into eight sub-sectors. Polynomial approximation is done using a 4th order polynomial, for each sub-sector respectively. Eight arrays, each including four polynomial coefficients for each sub-interval, are stored in this (GFLIB_TAN_T_F32) structure.

5.131.1 Source File

```
#include <GFLIB_Tan.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.131.2 Compound Type Members

Table 578. GFLIB_TAN_T_F32 members description

Type	Name	Description
GFLIB_TAN_TAYLOR_COEF_T_F32	GFLIB_TAN_SECTOR	Output of tan(PI * f32In) for interval [0, PI/4) of the input angles is divided into eight sub-sectors. Polynomial approximation is done using a 4th order polynomial, for each sub-sector respectively. Eight arrays, each including four polynomial coefficients for each sub-interval, are stored in this (GFLIB_TAN_T_F32) structure.

5.132 GFLIB_TAN_T_FLT

Polynomial coefficient for fractional approximation in single precision floating point format.

5.132.1 Source File

```
#include <GFLIB_Tan.h>
```

5.132.2 Compound Type Members

Table 579. GFLIB_TAN_T_FLT members description

Type	Name	Description
tFloat	fltA	Polynomial coefficient for fractional approximation in single precision floating point format.

5.133 GFLIB_TAN_TAYLOR_COEF_T_F16

Structure containing four polynomial coefficients for one sub-interval.

5.133.1 Source File

```
#include <GFLIB_Tan.h>
```

5.133.2 Description

Output of tan(π * f16In) for interval [0, π/4) of the input angles is divided into eight sub-sectors. Polynomial approximation is done using a 4th order polynomial, for each sub-sector respectively. Four coefficients for a single sub-interval are stored in this (GFLIB_TAN_TAYLOR_COEF_T_F16) structure.

5.133.3 Compound Type Members

Table 580. GFLIB_TAN_TAYLOR_COEF_T_F16 members description

Type	Name	Description
const tFrac16	f16A	Structure containing four polynomial coefficients for one sub-interval.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.134 GFLIB_TAN_TAYLOR_COEF_T_F32

Structure containing four polynomial coefficients for one sub-interval.

5.134.1 Source File

```
#include <GFLIB_Tan.h>
```

5.134.2 Description

Output of tan(π * f32In) for interval [0, π/4) of the input angles is divided into eight sub-sectors. Polynomial approximation is done using a 4th order polynomial, for each sub-sector respectively. Four coefficients for a single sub-interval are stored in this (GFLIB_TAN_TAYLOR_COEF_T_F32) structure.

5.134.3 Compound Type Members

Table 581. GFLIB_TAN_TAYLOR_COEF_T_F32 members description

Type	Name	Description
const tFrac32	f32A	Structure containing four polynomial coefficients for one sub-interval.

5.135 GFLIB_UPPERLIMIT_T_F16

Structure containing the upper limit.

5.135.1 Source File

```
#include <GFLIB_UpperLimit.h>
```

5.135.2 Compound Type Members

Table 582. GFLIB_UPPERLIMIT_T_F16 members description

Type	Name	Description
tFrac16	f16UpperLimit	Value determining the upper limit threshold.

5.136 GFLIB_UPPERLIMIT_T_F32

Structure containing the upper limit.

5.136.1 Source File

```
#include <GFLIB_UpperLimit.h>
```

5.136.2 Compound Type Members

Table 583. GFLIB_UPPERLIMIT_T_F32 members description

Type	Name	Description
tFrac32	f32UpperLimit	Value determining the upper limit threshold.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.137 GFLIB_UPPERLIMIT_T_FLT

Structure containing the upper limit.

5.137.1 Source File

```
#include <GFLIB_UpperLimit.h>
```

5.137.2 Compound Type Members

Table 584. GFLIB_UPPERLIMIT_T_FLT members description

Type	Name	Description
tFloat	fltUpperLimit	Value determining the upper limit threshold.

5.138 GFLIB_VECTORLIMIT_T_F16

Structure containing the limit.

5.138.1 Source File

```
#include <GFLIB_VectorLimit.h>
```

5.138.2 Compound Type Members

Table 585. GFLIB_VECTORLIMIT_T_F16 members description

Type	Name	Description
tFrac16	f16Limit	The maximum magnitude of the input vector. The defined magnitude must be positive and equal to or greater than F16_MAX value.

5.139 GFLIB_VECTORLIMIT_T_F32

Structure containing the limit.

5.139.1 Source File

```
#include <GFLIB_VectorLimit.h>
```

5.139.2 Compound Type Members

Table 586. GFLIB_VECTORLIMIT_T_F32 members description

Type	Name	Description
tFrac32	f32Limit	The maximum magnitude of the input vector. The defined magnitude must be positive and equal to or greater than F32_MAX value.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.140 GFLIB_VECTORLIMIT_T_FLT

Structure containing the limit.

5.140.1 Source File

```
#include <GFLIB_VectorLimit.h>
```

5.140.2 Compound Type Members

Table 587. GFLIB_VECTORLIMIT_T_FLT members description

Type	Name	Description
tFloat	fltLimit	The maximum magnitude of the input vector. The defined magnitude must be positive and equal to or greater than FLT_MAX value.

5.141 GFLIB_VLOG10_T_FLT

Array of single precision floating point elements for storing the coefficients of the floating point log10 approximation polynomial.

5.141.1 Source File

```
#include <GFLIB_VLog10.h>
```

5.141.2 Compound Type Members

Table 588. GFLIB_VLOG10_T_FLT members description

Type	Name	Description
tFloat	fltA	Array of single precision floating point elements for storing the coefficients of the floating point log10 approximation polynomial.

5.142 GMCLIB_CLARK_6PH_T_F16

Parameters structure for 6-phase Clark function containing transformation coefficients calculated by GMCLIB_Clark6PhInit function.

5.142.1 Source File

```
#include <GMCLIB_Clark6Ph.h>
```

5.142.2 Compound Type Members

Table 589. GMCLIB_CLARK_6PH_T_F16 members description

Type	Name	Description
tFrac16	f16AlphaD	Transformation coefficient AlphaD
tFrac16	f16AlphaE	Transformation coefficient AlphaE
tFrac16	f16AlphaF	Transformation coefficient AlphaF

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 589. GMCLIB_CLARK_6PH_T_F16 members description...*continued*

Type	Name	Description
tFrac16	f16BetaD	Transformation coefficient BetaD
tFrac16	f16BetaE	Transformation coefficient BetaE
tFrac16	f16BetaF	Transformation coefficient BetaF

5.143 GMCLIB_CLARK_6PH_T_F32

Parameters structure for 6-phase Clark function containing transformation coefficients calculated by GMCLIB_Clark6PhInit function.

5.143.1 Source File

```
#include <GMCLIB_Clark6Ph.h>
```

5.143.2 Compound Type Members

Table 590. GMCLIB_CLARK_6PH_T_F32 members description

Type	Name	Description
tFrac32	f32AlphaD	Transformation coefficient AlphaD
tFrac32	f32AlphaE	Transformation coefficient AlphaE
tFrac32	f32AlphaF	Transformation coefficient AlphaF
tFrac32	f32BetaD	Transformation coefficient BetaD
tFrac32	f32BetaE	Transformation coefficient BetaE
tFrac32	f32BetaF	Transformation coefficient BetaF

5.144 GMCLIB_CLARK_6PH_T_FLT

Parameters structure for 6-phase Clark function containing transformation coefficients calculated by GMCLIB_Clark6PhInit function.

5.144.1 Source File

```
#include <GMCLIB_Clark6Ph.h>
```

5.144.2 Compound Type Members

Table 591. GMCLIB_CLARK_6PH_T_FLT members description

Type	Name	Description
tFloat	fltAlphaD	Transformation coefficient AlphaD
tFloat	fltAlphaE	Transformation coefficient AlphaE
tFloat	fltAlphaF	Transformation coefficient AlphaF
tFloat	fltBetaD	Transformation coefficient BetaD
tFloat	fltBetaE	Transformation coefficient BetaE
tFloat	fltBetaF	Transformation coefficient BetaF

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.145 GMCLIB_CLARKINV_6PH_T_F16

Parameters structure for 6-phase ClarkInv function containing transformation coefficients calculated by GMCLIB_ClarkInv6PhInit function.

5.145.1 Source File

```
#include <GMCLIB_ClarkInv6Ph.h>
```

5.145.2 Compound Type Members

Table 592. GMCLIB_CLARKINV_6PH_T_F16 members description

Type	Name	Description
tFrac16	f16AlphaD	Transformation coefficient AlphaD
tFrac16	f16AlphaE	Transformation coefficient AlphaE
tFrac16	f16AlphaF	Transformation coefficient AlphaF
tFrac16	f16BetaD	Transformation coefficient BetaD
tFrac16	f16BetaE	Transformation coefficient BetaE
tFrac16	f16BetaF	Transformation coefficient BetaF

5.146 GMCLIB_CLARKINV_6PH_T_F32

Parameters structure for 6-phase ClarkInv function containing transformation coefficients calculated by GMCLIB_ClarkInv6PhInit function.

5.146.1 Source File

```
#include <GMCLIB_ClarkInv6Ph.h>
```

5.146.2 Compound Type Members

Table 593. GMCLIB_CLARKINV_6PH_T_F32 members description

Type	Name	Description
tFrac32	f32AlphaD	Transformation coefficient AlphaD
tFrac32	f32AlphaE	Transformation coefficient AlphaE
tFrac32	f32AlphaF	Transformation coefficient AlphaF
tFrac32	f32BetaD	Transformation coefficient BetaD
tFrac32	f32BetaE	Transformation coefficient BetaE
tFrac32	f32BetaF	Transformation coefficient BetaF

5.147 GMCLIB_CLARKINV_6PH_T_FLT

Parameters structure for 6-phase ClarkInv function containing transformation coefficients calculated by GMCLIB_ClarkInv6PhInit function.

5.147.1 Source File

```
#include <GMCLIB_ClarkInv6Ph.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.147.2 Compound Type Members

Table 594. GMCLIB_CLARKINV_6PH_T_FLT members description

Type	Name	Description
tFloat	fltAlphaD	Transformation coefficient AlphaD
tFloat	fltAlphaE	Transformation coefficient AlphaE
tFloat	fltAlphaF	Transformation coefficient AlphaF
tFloat	fltBetaD	Transformation coefficient BetaD
tFloat	fltBetaE	Transformation coefficient BetaE
tFloat	fltBetaF	Transformation coefficient BetaF

5.148 GMCLIB_DECOUPLINGPMSM_T_F16

Structure containing coefficients for calculation of the decoupling.

5.148.1 Source File

```
#include <GMCLIB_DecouplingPMSM.h>
```

5.148.2 Compound Type Members

Table 595. GMCLIB_DECOUPLINGPMSM_T_F16 members description

Type	Name	Description
tFrac16	f16Kd	Coefficient k_{df}.
tS16	s16KdShift	Scaling coefficient k_{d_shift}.
tFrac16	f16Kq	Coefficient k_{qf}.
tS16	s16KqShift	Scaling coefficient k_{q_shift}.

5.149 GMCLIB_DECOUPLINGPMSM_T_F32

Structure containing coefficients for calculation of the decoupling.

5.149.1 Source File

```
#include <GMCLIB_DecouplingPMSM.h>
```

5.149.2 Compound Type Members

Table 596. GMCLIB_DECOUPLINGPMSM_T_F32 members description

Type	Name	Description
tFrac32	f32Kd	Coefficient k_{df}.
tS16	s16KdShift	Scaling coefficient k_{d_shift}.
tFrac32	f32Kq	Coefficient k_{qf}.
tS16	s16KqShift	Scaling coefficient k_{q_shift}.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.150 GMCLIB_DECOUPLINGPMSM_T_FLT

Structure containing coefficients for calculation of the decoupling.

5.150.1 Source File

```
#include <GMCLIB_DecouplingPMSM.h>
```

5.150.2 Compound Type Members

Table 597. GMCLIB_DECOUPLINGPMSM_T_FLT members description

Type	Name	Description
tFloat	fltLD	L_D inductance [H].
tFloat	fltLQ	L_Q inductance [H].

5.151 GMCLIB_DOUBLESWITCHING_3PH_EDGES_T_F16

Structure of PWM edges for each phase of the three-phase system.

5.151.1 Source File

```
#include <GMCLIB_DoubleSwitching.h>
```

5.151.2 Compound Type Members

Table 598. GMCLIB_DOUBLESWITCHING_3PH_EDGES_T_F16 members description

Type	Name	Description
GMCLIB_DOUBLESWITCHING_EDGES_T_F16	pPhA	PWM edges for phase A
GMCLIB_DOUBLESWITCHING_EDGES_T_F16	pPhB	PWM edges for phase B
GMCLIB_DOUBLESWITCHING_EDGES_T_F16	pPhC	PWM edges for phase C

5.152 GMCLIB_DOUBLESWITCHING_EDGES_T_F16

Structure of PWM edges for two modulators.

5.152.1 Source File

```
#include <GMCLIB_DoubleSwitching.h>
```

5.152.2 Compound Type Members

Table 599. GMCLIB_DOUBLESWITCHING_EDGES_T_F16 members description

Type	Name	Description
tFrac16	f16Edge1	First edge for the first modulator
tFrac16	f16Edge2	Second edge for the first modulator

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 599. GMCLIB_DOUBLESWITCHING_EDGES_T_F16 members description...*continued*

Type	Name	Description
tFrac16	f16Edge3	First edge for the second modulator
tFrac16	f16Edge4	Second edge for the second modulator

5.153 GMCLIB_DOUBLESWITCHING_T_F16

Configuration structure for double switching algorithm.

5.153.1 Source File

```
#include <GMCLIB_DoubleSwitching.h>
```

5.153.2 Compound Type Members

Table 600. GMCLIB_DOUBLESWITCHING_T_F16 members description

Type	Name	Description
tFrac16	f16MinZeroPulse	The minimal pulse inserted to the middle of the PWM period of double-switching signal. User has to consider at least one deadtime to be included in this value. Value is stored as 16-bit fixed point number.
tFrac16	f16MinSamplingPulse	Represents at least the minimal conversion time of ADC (is a sum of "sample&hold" time and "successive approximation" time). Additional delays such as ADC pre-trigger delay or delays between two subsequent triggers caused by the triggering unit should be included as well. Value is stored as 16-bit fixed point number.
tFrac16	f16SampleNhold	The minimal "sample&hold" time of the ADC represented by 16-bit fixed point value.

5.154 GMCLIB_DOUBLESWITCHING_TRIGGERS_T_F16

Structure of ADC triggers for triggering unit.

5.154.1 Source File

```
#include <GMCLIB_DoubleSwitching.h>
```

5.154.2 Compound Type Members

Table 601. GMCLIB_DOUBLESWITCHING_TRIGGERS_T_F16 members description

Type	Name	Description
tU16	u16Trigger1	First trigger
tU16	u16Trigger2	Second trigger

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 601. GMCLIB_DOUBLESWITCHING_TRIGGERS_T_F16 members description...*continued*

Type	Name	Description
tU16	u16Trigger3	Third trigger
tU16	u16Trigger4	Fourth trigger

5.155 GMCLIB_ELIMDCBUSRIP_T_F16

Structure containing the PWM modulation index and the measured value of the DC bus voltage.

5.155.1 Source File

```
#include <GMCLIB_ElimDcBusRip.h>
```

5.155.2 Compound Type Members

Table 602. GMCLIB_ELIMDCBUSRIP_T_F16 members description

Type	Name	Description
tFrac16	f16ArgDcBusMsr	Measured DC bus voltage.
tFrac16	f16ModIndex	Inverse Modulation Index.

5.156 GMCLIB_ELIMDCBUSRIP_T_F32

Structure containing the PWM modulation index and the measured value of the DC bus voltage.

5.156.1 Source File

```
#include <GMCLIB_ElimDcBusRip.h>
```

5.156.2 Compound Type Members

Table 603. GMCLIB_ELIMDCBUSRIP_T_F32 members description

Type	Name	Description
tFrac32	f32ArgDcBusMsr	Measured DC bus voltage.
tFrac32	f32ModIndex	Inverse Modulation Index.

5.157 GMCLIB_ELIMDCBUSRIP_T_FLT

Structure containing the PWM modulation index and the measured value of the DC bus voltage.

5.157.1 Source File

```
#include <GMCLIB_ElimDcBusRip.h>
```


© NXP B.V. 2023. All rights reserved.

5.157.2 Compound Type Members

Table 604. GMCLIB_ELIMDCBUSRIP_T_FLT members description

Type	Name	Description
tFloat	fltArgDcBusMsr	Measured DC bus voltage.
tFloat	fltModIndex	Inverse Modulation Index.

5.158 GMCLIB_SPREADPWM_T

Structure containing parameters and states of GMCLIB_SpreadPWM.

5.158.1 Source File

```
#include <GMCLIB_SpreadPWM.h>
```

5.158.2 Compound Type Members

Table 605. GMCLIB_SPREADPWM_T members description

Type	Name	Description
tU32	u32Rnd	Generated 32-bit random number. This value is updated in every iteration of GMCLIB_SpreadPWM.
tU16	u16ModMin	PWM counter minimum modulus (corresponds to the highest PWM frequency). This value is calculated in GMCLIB_SpreadPWMInit. Do not edit.
tU16	u16ModMax	PWM counter maximum modulus (corresponds to the lowest PWM frequency). This value is calculated in GMCLIB_SpreadPWMInit. Do not edit.
tU16	u16CntK1	PWM counter modulus in previous iteration. Do not edit.
tFrac16	f16Inc	Internal variable. Do not edit.
tU16	u16DevConst	Internal variable. Do not edit.

5.159 GMCLIB_SVMSTD_6PH_T_F16

Parameters structure for 6-phase SvmStd function containing transformation coefficients calculated by GMCLIB_SvmStd6PhInit function.

5.159.1 Source File

```
#include <GMCLIB_SvmStd6Ph.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.159.2 Compound Type Members

Table 606. GMCLIB_SVMSTD_6PH_T_F16 members description

Type	Name	Description
tFrac16	f16AlphaD	Transformation coefficient AlphaD
tFrac16	f16AlphaE	Transformation coefficient AlphaE
tFrac16	f16AlphaF	Transformation coefficient AlphaF
tFrac16	f16BetaD	Transformation coefficient BetaD
tFrac16	f16BetaE	Transformation coefficient BetaE
tFrac16	f16BetaF	Transformation coefficient BetaF

5.160 GMCLIB_SVMSTD_6PH_T_F32

Parameters structure for 6-phase SvmStd function containing transformation coefficients calculated by GMCLIB_SvmStd6PhInit function.

5.160.1 Source File

```
#include <GMCLIB_SvmStd6Ph.h>
```

5.160.2 Compound Type Members

Table 607. GMCLIB_SVMSTD_6PH_T_F32 members description

Type	Name	Description
tFrac32	f32AlphaD	Transformation coefficient AlphaD
tFrac32	f32AlphaE	Transformation coefficient AlphaE
tFrac32	f32AlphaF	Transformation coefficient AlphaF
tFrac32	f32BetaD	Transformation coefficient BetaD
tFrac32	f32BetaE	Transformation coefficient BetaE
tFrac32	f32BetaF	Transformation coefficient BetaF

5.161 GMCLIB_SVMSTD_6PH_T_FLT

Parameters structure for 6-phase SvmStd function containing transformation coefficients calculated by GMCLIB_SvmStd6PhInit function.

5.161.1 Source File

```
#include <GMCLIB_SvmStd6Ph.h>
```

5.161.2 Compound Type Members

Table 608. GMCLIB_SVMSTD_6PH_T_FLT members description

Type	Name	Description
tFloat	fltAlphaD	Transformation coefficient AlphaD
tFloat	fltAlphaE	Transformation coefficient AlphaE
tFloat	fltAlphaF	Transformation coefficient AlphaF

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 608. GMCLIB_SVMSTD_6PH_T_FLT members description...*continued*

Type	Name	Description
tFloat	fltBetaD	Transformation coefficient BetaD
tFloat	fltBetaE	Transformation coefficient BetaE
tFloat	fltBetaF	Transformation coefficient BetaF

5.162 SWLIBS_2Syst_F16

Array of two standard 16-bit fractional arguments.

5.162.1 Source File

```
#include <SWLIBS_Typedefs.h>
```

5.162.2 Compound Type Members

Table 609. SWLIBS_2Syst_F16 members description

Type	Name	Description
tFrac16	f16Arg1	First argument
tFrac16	f16Arg2	Second argument

5.163 SWLIBS_2Syst_F32

Array of two standard 32-bit fractional arguments.

5.163.1 Source File

```
#include <SWLIBS_Typedefs.h>
```

5.163.2 Compound Type Members

Table 610. SWLIBS_2Syst_F32 members description

Type	Name	Description
tFrac32	f32Arg1	First argument
tFrac32	f32Arg2	Second argument

5.164 SWLIBS_2Syst_FLT

Array of two standard single precision floating point arguments.

5.164.1 Source File

```
#include <SWLIBS_Typedefs.h>
```

5.164.2 Compound Type Members

Table 611. SWLIBS_2Syst_FLT members description

Type	Name	Description
tFloat	fltArg1	First argument

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 611. SWLIBS_2Syst_FLT members description...*continued*

Type	Name	Description
tFloat	fltArg2	Second argument

5.165 SWLIBS_2Syst_U16

Array of two unsigned 16-bit integer arguments.

5.165.1 Source File

```
#include <SWLIBS_Typedefs.h>
```

5.165.2 Compound Type Members

Table 612. SWLIBS_2Syst_U16 members description

Type	Name	Description
tU16	u16Arg1	First argument
tU16	u16Arg2	Second argument

5.166 SWLIBS_2Syst_U32

Array of two unsigned 32-bit integer arguments.

5.166.1 Source File

```
#include <SWLIBS_Typedefs.h>
```

5.166.2 Compound Type Members

Table 613. SWLIBS_2Syst_U32 members description

Type	Name	Description
tU32	u32Arg1	First argument
tU32	u32Arg2	Second argument

5.167 SWLIBS_3Syst_F16

Array of three standard 16-bit fractional arguments.

5.167.1 Source File

```
#include <SWLIBS_Typedefs.h>
```

5.167.2 Compound Type Members

Table 614. SWLIBS_3Syst_F16 members description

Type	Name	Description
tFrac16	f16Arg1	First argument
tFrac16	f16Arg2	Second argument
tFrac16	f16Arg3	Third argument

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.168 SWLIBS_3Syst_F32

Array of three standard 32-bit fractional arguments.

5.168.1 Source File

```
#include <SWLIBS_Typedefs.h>
```

5.168.2 Compound Type Members

Table 615. SWLIBS_3Syst_F32 members description

Type	Name	Description
tFrac32	f32Arg1	First argument
tFrac32	f32Arg2	Second argument
tFrac32	f32Arg3	Third argument

5.169 SWLIBS_3Syst_FLT

Array of three standard single precision floating point arguments.

5.169.1 Source File

```
#include <SWLIBS_Typedefs.h>
```

5.169.2 Compound Type Members

Table 616. SWLIBS_3Syst_FLT members description

Type	Name	Description
tFloat	fltArg1	First argument
tFloat	fltArg2	Second argument
tFloat	fltArg3	Third argument

5.170 SWLIBS_3Syst_U16

Array of three unsigned 16-bit integer arguments.

5.170.1 Source File

```
#include <SWLIBS_Typedefs.h>
```

5.170.2 Compound Type Members

Table 617. SWLIBS_3Syst_U16 members description

Type	Name	Description
tU16	u16Arg1	First argument
tU16	u16Arg2	Second argument
tU16	u16Arg3	Third argument

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.171 SWLIBS_6Syst_F16

Array of six standard 16-bit fractional arguments.

5.171.1 Source File

```
#include <SWLIBS_Typedefs.h>
```

5.171.2 Compound Type Members

Table 618. SWLIBS_6Syst_F16 members description

Type	Name	Description
tFrac16	f16Arg1	First argument
tFrac16	f16Arg2	Second argument
tFrac16	f16Arg3	Third argument
tFrac16	f16Arg4	Fourth argument
tFrac16	f16Arg5	Fifth argument
tFrac16	f16Arg6	Sixth argument

5.172 SWLIBS_6Syst_F32

Array of six standard 32-bit fractional arguments.

5.172.1 Source File

```
#include <SWLIBS_Typedefs.h>
```

5.172.2 Compound Type Members

Table 619. SWLIBS_6Syst_F32 members description

Type	Name	Description
tFrac32	f32Arg1	First argument
tFrac32	f32Arg2	Second argument
tFrac32	f32Arg3	Third argument
tFrac32	f32Arg4	Fourth argument
tFrac32	f32Arg5	Fifth argument
tFrac32	f32Arg6	Sixth argument

5.173 SWLIBS_6Syst_FLT

Array of three standard single precision floating point arguments.

5.173.1 Source File

```
#include <SWLIBS_Typedefs.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.173.2 Compound Type Members

Table 620. SWLIBS_6Syst_FLT members description

Type	Name	Description
tFloat	fltArg1	First argument
tFloat	fltArg2	Second argument
tFloat	fltArg3	Third argument
tFloat	fltArg4	Fourth argument
tFloat	fltArg5	Fifth argument
tFloat	fltArg6	Sixth argument

5.174 SWLIBS_VERSION_T

Motor Control Library Set identification structure.

5.174.1 Source File

```
#include <SWLIBS_Version.h>
```

5.174.2 Compound Type Members

Table 621. SWLIBS_VERSION_T members description

Type	Name	Description
unsigned char	mcId	MCLIB identification code
unsigned char	mcVersion	MCLIB version code
unsigned char	mcImpl	MCLIB supported implementation code

6 Macros

This section describes in details the macro definitions available in AMMCLib.

6.1 AMCLIB_BEMF_OBSRV_DQ_DEFAULT_F32

Default value for AMCLIB_BEMF_OBSRV_DQ_T_F32.

6.1.1 Source File

```
#include <AMCLIB_BemfObsrvDQ.h>
```

6.1.2 Macro Definition

```
#define AMCLIB_BEMF_OBSRV_DQ_DEFAULT_F32 {(tFrac32)0,(tFrac32)0,
\ (tFrac32)0,(tFrac32)0, \ (tFrac32)0,(tFrac32)0,(tFrac32)0,
(tFrac32)0,INT32_MIN,INT32_MAX,(tU16)0, \ (tFrac32)0,(tFrac32)0,
(tFrac32)0,(tFrac32)0,INT32_MIN,INT32_MAX,(tU16)0, \ (tFrac32)0,
(tFrac32)0, \ (tFrac16)0,(tFrac16)0, \ (tFrac32)0,(tFrac32)0,
(tFrac32)0,(tFrac32)0,(tS16)0}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.2 AMCLIB_BEMF_OBSRV_DQ_DEFAULT_F16

Default value for AMCLIB_BEMF_OBSRV_DQ_T_F16.

6.2.1 Source File

```
#include <AMCLIB_BemfObsrvDQ.h>
```

6.2.2 Macro Definition

```
#define AMCLIB_BEMF_OBSRV_DQ_DEFAULT_F16 {(tFrac16)0,(tFrac16)0,
\ (tFrac32)0,(tFrac32)0, \ (tFrac16)0,(tFrac16)0,(tFrac32)0,
(tFrac16)0,INT16_MIN,INT16_MAX,(tU16)0, \ (tFrac16)0,(tFrac16)0,
(tFrac32)0,(tFrac16)0,INT16_MIN,INT16_MAX,(tU16)0, \ (tFrac32)0,
(tFrac32)0, \ (tFrac16)0,(tFrac16)0, \ (tFrac16)0,(tFrac16)0,
(tFrac16)0,(tFrac16)0,(tS16)0}
```

6.3 AMCLIB_BEMF_OBSRV_DQ_DEFAULT_FLT

Default value for AMCLIB_BEMF_OBSRV_DQ_T_FLT.

6.3.1 Source File

```
#include <AMCLIB_BemfObsrvDQ.h>
```

6.3.2 Macro Definition

```
#define AMCLIB_BEMF_OBSRV_DQ_DEFAULT_FLT {(tFloat)0,(tFloat)0,
\ (tFloat)0,(tFloat)0, \ (tFloat)0,(tFloat)0,(tFloat)0,
(tFloat)0,FLOAT_MIN,FLOAT_MAX, \ (tFloat)0,(tFloat)0,(tFloat)0,
(tFloat)0,FLOAT_MIN,FLOAT_MAX, \ (tFloat)0,(tFloat)0, \
(tFloat)0,(tFloat)0,(tFloat)0,(tFloat)0}
```

6.4 AMCLIB_CURRENT_LOOP_DEFAULT_F32

Default value for AMCLIB_CURRENT_LOOP_T_F32.

6.4.1 Source File

```
#include <AMCLIB_CurrentLoop.h>
```

6.4.2 Macro Definition

```
#define AMCLIB_CURRENT_LOOP_DEFAULT_F32 { \ (tFrac32)0,
(tFrac32)0,(tS32)0,(tS32)0,INT32_MIN,INT32_MAX,(tFrac32)0,
(tFrac32)0,(tU16)0, \ (tFrac32)0,(tFrac32)0,(tS32)0,
(tS32)0,INT32_MIN,INT32_MAX,(tFrac32)0,(tFrac32)0,(tU16)0, \
(tFrac32)0,(tFrac32)0, \ (tFrac32)0,(tFrac32)0,}
```

6.5 AMCLIB_CURRENT_LOOP_DEFAULT_F16

Default value for AMCLIB_CURRENT_LOOP_T_F16.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.5.1 Source File

```
#include <AMCLIB_CurrentLoop.h>
```

6.5.2 Macro Definition

```
#define AMCLIB_CURRENT_LOOP_DEFAULT_F16 { \ (tFrac16)0,
(tFrac16)0,(tS16)0,(tS16)0,INT16_MIN,INT16_MAX,(tFrac32)0,
(tFrac16)0,(tU16)0, \ (tFrac16)0,(tFrac16)0,(tS16)0,
(tS16)0,INT16_MIN,INT16_MAX,(tFrac32)0,(tFrac16)0,(tU16)0, \
(tFrac16)0,(tFrac16)0, \ (tFrac16)0,(tFrac16)0,}
```

6.6 AMCLIB_CURRENT_LOOP_DEFAULT_FLT

Default value for AMCLIB_CURRENT_LOOP_T_FLT.

6.6.1 Source File

```
#include <AMCLIB_CurrentLoop.h>
```

6.6.2 Macro Definition

```
#define AMCLIB_CURRENT_LOOP_DEFAULT_FLT { \ (tFloat)0,
(tFloat)0,FLOAT_MIN,FLOAT_MAX,(tFloat)0,(tFloat)0,(tU16)0, \
(tFloat)0,(tFloat)0,FLOAT_MIN,FLOAT_MAX,(tFloat)0,(tFloat)0,
(tU16)0, \ (tFloat)0,(tFloat)0, \ (tFloat)0,(tFloat)0}
```

6.7 ESTIMBJ_INIT_DEFAULT_FLT

Default configuration for AMCLIB_EstimBJInit_FLT. Use this macro to fill the structure pParam in the function API. Arguments must adhere to the data types of AMCLIB_ESTIMBJ_INIT_T_FLT.

6.7.1 Source File

```
#include <AMCLIB_EstimBJ.h>
```

6.7.2 Macro Definition

```
#define ESTIMBJ_INIT_DEFAULT_FLT (AMCLIB_ESTIMBJ_INIT_T_FLT)
{((tU32)fs),((tFloat)In),((tFloat)Wn),((tFloat)pp),
((tFloat)Ld),((tFloat)Lq),((tFloat)Rs),
((tFloat)Udt),0.0f,0.0f,0.0f,0.0f,0.0f,0.0f,0.0f,0.0f,0.0f}
```

6.7.3 Arguments

Table 622. ESTIMBJ_INIT_DEFAULT_FLT arguments

Name	Description
fs	Sampling frequency [1/s].
In	Nominal current of the motor [A].
Wn	Nominal velocity of the motor [rpm].
pp	Number of pole pairs [-].
Ld	D-axis inductance [H].

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Table 622. ESTIMBJ_INIT_DEFAULT_FLT arguments...*continued*

Name	Description
Lq	Q-axis inductance [H].
Rs	Stator resistance [Ohm].
Udt	Dead time voltage drop of the power stage [V].

6.8 ESTIMRL_INIT_DEFAULT_FLT

Default configuration for AMCLIB_EstimRLInit_FLT. Use this macro to fill the structure pParam in the function API. Arguments must adhere to the data types of AMCLIB_ESTIMRL_INIT_T_FLT.

6.8.1 Source File

```
#include <AMCLIB_EstimRL.h>
```

6.8.2 Macro Definition

```
#define ESTIMRL_INIT_DEFAULT_FLT (AMCLIB_ESTIMRL_INIT_T_FLT)
{(fltINominal),0u,0.0f,0.0f,0.0f}
```

6.8.3 Arguments

Table 623. ESTIMRL_INIT_DEFAULT_FLT arguments

Name	Description
fltINominal	Nominal current of the motor [A].

6.9 ESTIMRL_INIT_CFG1_FLT

Custom configuration set 1 for AMCLIB_EstimRLInit_FLT. Use this macro to fill the structure pParam in the function API. Arguments must adhere to the data types of AMCLIB_ESTIMRL_INIT_T_FLT.

6.9.1 Source File

```
#include <AMCLIB_EstimRL.h>
```

6.9.2 Macro Definition

```
#define ESTIMRL_INIT_CFG1_FLT (AMCLIB_ESTIMRL_INIT_T_FLT)
{(fltIDcMax),(u16LdqNumMeas),0.0f,(fltIDcLq),0.0f}
```

6.9.3 Arguments

Table 624. ESTIMRL_INIT_CFG1_FLT arguments

Name	Description
fltIDcMax	Maximum DC current [A].
u16LdqNumMeas	Number of measured DC current levels.
fltIDcLq	DC current [A] for which the q-axis inductance is measured.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.10 ESTIMRL_INIT_CFG2_FLT

Custom configuration set 2 for AMCLIB_EstimRLInit_FLT. Use this macro to fill the structure pParam in the function API. Arguments must adhere to the data types of AMCLIB_ESTIMRL_INIT_T_FLT.

6.10.1 Source File

```
#include <AMCLIB_EstimRL.h>
```

6.10.2 Macro Definition

```
#define ESTIMRL_INIT_CFG2_FLT (AMCLIB_ESTIMRL_INIT_T_FLT)
{(fltIDcMax),(u16LdqNumMeas),(fltIDcLd),(fltIDcLq),
(fltIDcNegMax)}
```

6.10.3 Arguments

Table 625. ESTIMRL_INIT_CFG2_FLT arguments

Name	Description
fltIDcMax	Maximum DC current [A].
u16LdqNumMeas	Number of measured DC current levels.
fltIDcLd	DC current [A] for which the d-axis inductance is measured.
fltIDcLq	DC current [A] for which the q-axis inductance is measured.
fltIDcNegMax	Maximum negative DC current [A].

6.11 ESTIMRL_INIT_CFG3_FLT

Custom configuration set 3 for AMCLIB_EstimRLInit_FLT for fully manual measurement. Use this macro to fill the structure pParam in the function API. All parameters will be specified in the API of the function AMCLIB_EstimRL_FLT during run-time.

6.11.1 Source File

```
#include <AMCLIB_EstimRL.h>
```

6.11.2 Macro Definition

```
#define ESTIMRL_INIT_CFG3_FLT (AMCLIB_ESTIMRL_INIT_T_FLT)
{0.0f,1u,0.0f,0.0f,0.0f}
```

6.12 ESTIMRL_RUN_DEFAULT_FLT

Default configuration for AMCLIB_EstimRL_FLT. Use this macro to fill the structure pParam in the function API.

6.12.1 Source File

```
#include <AMCLIB_EstimRL.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.12.2 Macro Definition

```
#define ESTIMRL_RUN_DEFAULT_FLT (AMCLIB_ESTIMRL_RUN_T_FLT)
{NULL,0.0f,0.0f,0.0f,0u,0u}
```

6.13 ESTIMRL_RUN_CFG1_FLT

Custom configuration set 1 for AMCLIB_EstimRL_FLT. Use this macro to fill the structure pParam in the function API. Arguments must adhere to the data types of AMCLIB_ESTIMRL_RUN_T_FLT.

6.13.1 Source File

```
#include <AMCLIB_EstimRL.h>
```

6.13.2 Macro Definition

```
#define ESTIMRL_RUN_CFG1_FLT (AMCLIB_ESTIMRL_RUN_T_FLT)
{(pLdqTable),0.0f,0.0f,0.0f,0u,0u}
```

6.13.3 Arguments

Table 626. ESTIMRL_RUN_CFG1_FLT arguments

Name	Description
pLdqTable	Pointer to the table of measured inductances. The table has 3 rows and u16LdqNumMeas columns stored in column-major order. Parameter u16LdqNumMeas of the init function AMCLIB_EstimRLInit_FLT determines the number of used DC current levels. The array pointed to by pLdqTable must provide room for 3*u16LdqNumMeas 32-bit values. The first row of the table contains the DC current levels [A], the second row contains the estimated d-axis inductances, and the third row contains the estimated q-axis inductances. The inductances are calculated in [H].

6.14 ESTIMRL_RUN_CFG2_FLT

Custom configuration set 2 for AMCLIB_EstimRL_FLT. Use this macro to fill the structure pParam in the function API. Arguments must adhere to the data types of AMCLIB_ESTIMRL_RUN_T_FLT.

6.14.1 Source File

```
#include <AMCLIB_EstimRL.h>
```

6.14.2 Macro Definition

```
#define ESTIMRL_RUN_CFG2_FLT (AMCLIB_ESTIMRL_RUN_T_FLT)
{(pLdqTable),0.0f,0.0f,0.0f,0u,0u}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.14.3 Arguments

Table 627. ESTIMRL_RUN_CFG2_FLT arguments

Name	Description
pLdqTable	Pointer to the table of measured inductances. The table has 3 rows and u16LdqNumMeas columns stored in column-major order. Parameter u16LdqNumMeas of the init function AMCLIB_EstimRLInit_FLT determines the number of used DC current levels. The array pointed to by pLdqTable must provide room for 3*u16LdqNumMeas 32-bit values. The first row of the table contains the DC current levels [A], the second row contains the estimated d-axis inductances, and the third row contains the estimated q-axis inductances. The inductances are calculated in [H].

6.15 ESTIMRL_RUN_CFG3_FLT

Custom configuration set 3 for AMCLIB_EstimRL_FLT. Use this macro to fill the structure pParam in the function API. Arguments must adhere to the data types of AMCLIB_ESTIMRL_RUN_T_FLT.

6.15.1 Source File

```
#include <AMCLIB_EstimRL.h>
```

6.15.2 Macro Definition

```
#define ESTIMRL_RUN_CFG3_FLT (AMCLIB_ESTIMRL_RUN_T_FLT){NULL,
(fltIDcDReq),(fltIDcQReq),(fltIAcReq),(u16FAc),(u8LdqSwitch)}
```

6.15.3 Arguments

Table 628. ESTIMRL_RUN_CFG3_FLT arguments

Name	Description
fltIDcDReq	Required DC current in the d-axis [A].
fltIDcQReq	Required DC current in the q-axis [A].
fltIAcReq	Required amplitude of the injected high-frequency AC current (default: 10% of IDc) [A].
u16FAc	Required frequency of the injected high-frequency AC current (default: 500) [Hz]. Must not exceed 25% of the sampling frequency.
u8LdqSwitch	Switches between d-axis measurement (0) and q-axis measurement (1).

6.16 AMCLIB_FW_DEFAULT_F32

Default value for AMCLIB_FW_T_F32.

6.16.1 Source File

```
#include <AMCLIB_FW.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.16.2 Macro Definition

```
#define AMCLIB_FW_DEFAULT_F32 {0,0, \ (tFrac32)0,(tFrac32)0,
(tS32)0,(tS32)0,INT32_MIN,INT32_MAX,(tFrac32)0,(tFrac32)0,
(tU16)0, \ (tFrac32 *)0, \ (tFrac32 *)0, \ (tFrac32 *)0}
```

6.17 AMCLIB_FW_DEFAULT_F16

Default value for AMCLIB_FW_T_F16.

6.17.1 Source File

```
#include <AMCLIB_FW.h>
```

6.17.2 Macro Definition

```
#define AMCLIB_FW_DEFAULT_F16 {0,0, \ (tFrac16)0,(tFrac16)0,
(tS16)0,(tS16)0,INT16_MIN,INT16_MAX,(tFrac32)0,(tFrac16)0,
(tU16)0, \ (tFrac16 *)0, \ (tFrac16 *)0, \ (tFrac16 *)0}
```

6.18 AMCLIB_FW_DEFAULT_FLT

Default value for AMCLIB_FW_T_FLT.

6.18.1 Source File

```
#include <AMCLIB_FW.h>
```

6.18.2 Macro Definition

```
#define AMCLIB_FW_DEFAULT_FLT {0,0 \ (tFloat)0,
(tFloat)0,FLOAT_MIN,FLOAT_MAX,(tFloat)0,(tFloat)0,(tU16)0, \
(tFloat *)0, \ (tFloat *)0, \ (tFloat *)0}
```

6.19 AMCLIB_FW_SPEED_LOOP_DEFAULT_F32

Default value for AMCLIB_FW_SPEED_LOOP_T_F32.

6.19.1 Source File

```
#include <AMCLIB_FWSpeedLoop.h>
```

6.19.2 Macro Definition

```
#define AMCLIB_FW_SPEED_LOOP_DEFAULT_F32 {0,0, \ 0,0, \
(tFrac32)0,(tFrac32)0,(tS32)0,(tS32)0,INT32_MIN,INT32_MAX,
(tFrac32)0,(tFrac32)0,(tU16)0, \ (tFrac32)0,(tFrac32)0,(tS32)0,
(tS32)0,INT32_MIN,INT32_MAX,(tFrac32)0,(tFrac32)0,(tU16)0, \
(tFrac32)0,(tFrac32)0,(tFrac32)0, \ (tFrac32 *)0, \ (tFrac32 *)0,
\ (tFrac32 *)0}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.20 AMCLIB_FW_SPEED_LOOP_DEFAULT_F16

Default value for AMCLIB_FW_SPEED_LOOP_T_F16.

6.20.1 Source File

```
#include <AMCLIB_FWSpeedLoop.h>
```

6.20.2 Macro Definition

```
#define AMCLIB_FW_SPEED_LOOP_DEFAULT_F16 {0,0, \ 0,0, \
(tFrac16)0,(tFrac16)0,(tS16)0,(tS16)0,INT16_MIN,INT16_MAX,
(tFrac32)0,(tFrac16)0,(tU16)0, \ (tFrac16)0,(tFrac16)0,(tS16)0,
(tS16)0,INT16_MIN,INT16_MAX,(tFrac32)0,(tFrac16)0,(tU16)0, \
(tFrac32)0,(tFrac32)0,(tFrac32)0, \ (tFrac16 *)0, \ (tFrac16 *)0,
\ (tFrac16 *)0}
```

6.21 AMCLIB_FW_SPEED_LOOP_DEFAULT_FLT

Default value for AMCLIB_FW_SPEED_LOOP_T_FLT.

6.21.1 Source File

```
#include <AMCLIB_FWSpeedLoop.h>
```

6.21.2 Macro Definition

```
#define AMCLIB_FW_SPEED_LOOP_DEFAULT_FLT {0,0 \ 0,0 \ (tFloat)0,
(tFloat)0,FLOAT_MIN,FLOAT_MAX,(tFloat)0,(tFloat)0,(tU16)0, \
(tFloat)0,(tFloat)0,FLOAT_MIN,FLOAT_MAX,(tFloat)0,(tFloat)0,
(tU16)0, \ (tFloat)0,(tFloat)0,(tFloat)0, \ (tFloat *)0, \
(tFloat *)0, \ (tFloat *)0}
```

6.22 AMCLIB_SPEED_LOOP_DEFAULT_F32

Default value for AMCLIB_SPEED_LOOP_T_F32.

6.22.1 Source File

```
#include <AMCLIB_SpeedLoop.h>
```

6.22.2 Macro Definition

```
#define AMCLIB_SPEED_LOOP_DEFAULT_F32 {0,0, \ (tFrac32)0,
(tFrac32)0,(tS32)0,(tS32)0,INT32_MIN,INT32_MAX,(tFrac32)0,
(tFrac32)0,(tU16)0, \ (tFrac32)0,(tFrac32)0,(tFrac32)0}
```

6.23 AMCLIB_SPEED_LOOP_DEFAULT_F16

Default value for AMCLIB_SPEED_LOOP_T_F16.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.23.1 Source File

```
#include <AMCLIB_SpeedLoop.h>
```

6.23.2 Macro Definition

```
#define AMCLIB_SPEED_LOOP_DEFAULT_F16 {0,0, \ (tFrac16)0,
(tFrac16)0,(tS16)0,(tS16)0,INT16_MIN,INT16_MAX,(tFrac32)0,
(tFrac16)0,(tU16)0, \ (tFrac32)0,(tFrac32)0,(tFrac32)0}
```

6.24 AMCLIB_SPEED_LOOP_DEFAULT_FLT

Default value for AMCLIB_SPEED_LOOP_T_FLT.

6.24.1 Source File

```
#include <AMCLIB_SpeedLoop.h>
```

6.24.2 Macro Definition

```
#define AMCLIB_SPEED_LOOP_DEFAULT_FLT {0,0 \ (tFloat)0,
(tFloat)0,FLOAT_MIN,FLOAT_MAX,(tFloat)0,(tFloat)0,(tU16)0, \
(tFloat)0,(tFloat)0,(tFloat)0}
```

6.25 AMCLIB_TRACK_OBSRV_T

Definition of AMCLIB_TRACK_OBSRV_T as alias for AMCLIB_TRACK_OBSRV_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.25.1 Source File

```
#include <AMCLIB_TrackObsrv.h>
```

6.25.2 Macro Definition

```
#define AMCLIB_TRACK_OBSRV_T AMCLIB_TRACK_OBSRV_T_F16
```

6.26 AMCLIB_TRACK_OBSRV_DEFAULT

Definition of AMCLIB_TRACK_OBSRV_DEFAULT as alias for AMCLIB_TRACK_OBSRV_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.26.1 Source File

```
#include <AMCLIB_TrackObsrv.h>
```

6.26.2 Macro Definition

```
#define AMCLIB_TRACK_OBSRV_DEFAULT AMCLIB_TRACK_OBSRV_DEFAULT_F16
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.27 AMCLIB_TRACK_OBSRV_DEFAULT_F32

Default value for AMCLIB_TRACK_OBSRV_T_F32.

6.27.1 Source File

```
#include <AMCLIB_TrackObsrv.h>
```

6.27.2 Macro Definition

```
#define AMCLIB_TRACK_OBSRV_DEFAULT_F32 {(tFrac32)0,(tFrac32)0,
(tFrac32)0,(tFrac32)0,INT32_MIN,INT32_MAX,(tU16)0, \ (tFrac32)0,
(tFrac32)0,(tFrac32)0,(tU16)0}
```

6.28 AMCLIB_TRACK_OBSRV_DEFAULT_F16

Default value for AMCLIB_TRACK_OBSRV_T_F16.

6.28.1 Source File

```
#include <AMCLIB_TrackObsrv.h>
```

6.28.2 Macro Definition

```
#define AMCLIB_TRACK_OBSRV_DEFAULT_F16 {(tFrac16)0,(tFrac16)0,
(tFrac32)0,(tFrac16)0,INT16_MIN,INT16_MAX,(tU16)0, \ (tFrac32)0,
(tFrac16)0,(tFrac16)0,(tU16)0}
```

6.29 AMCLIB_TRACK_OBSRV_DEFAULT_FLT

Default value for AMCLIB_TRACK_OBSRV_T_FLT.

6.29.1 Source File

```
#include <AMCLIB_TrackObsrv.h>
```

6.29.2 Macro Definition

```
#define AMCLIB_TRACK_OBSRV_DEFAULT_FLT {(tFloat)0,(tFloat)0,
(tFloat)0,(tFloat)0,FLOAT_MIN,FLOAT_MAX, \ (tFloat)0,(tFloat)0,
(tFloat)0}
```

6.30 GDFLIB_FILTERFIR_PARAM_T

Definition of alias for GDFLIB_FILTERFIR_PARAM_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.30.1 Source File

```
#include <GDFLIB_FilterFIR.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.30.2 Macro Definition

```
#define GDFLIB_FILTERFIR_PARAM_T GDFLIB_FILTERFIR_PARAM_T_F16
```

6.31 GDFLIB_FILTERFIR_STATE_T

Definition of alias for GDFLIB_FILTERFIR_STATE_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.31.1 Source File

```
#include <GDFLIB_FilterFIR.h>
```

6.31.2 Macro Definition

```
#define GDFLIB_FILTERFIR_STATE_T GDFLIB_FILTERFIR_STATE_T_F16
```

6.32 GDFLIB_FILTER_IIR1_T

Definition of GDFLIB_FILTER_IIR1_T as alias for GDFLIB_FILTER_IIR1_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.32.1 Source File

```
#include <GDFLIB_FilterIIR1.h>
```

6.32.2 Macro Definition

```
#define GDFLIB_FILTER_IIR1_T GDFLIB_FILTER_IIR1_T_F16
```

6.33 GDFLIB_FILTER_IIR1_DEFAULT

Definition of GDFLIB_FILTER_IIR1_DEFAULT as alias for GDFLIB_FILTER_IIR1_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.33.1 Source File

```
#include <GDFLIB_FilterIIR1.h>
```

6.33.2 Macro Definition

```
#define GDFLIB_FILTER_IIR1_DEFAULT GDFLIB_FILTER_IIR1_DEFAULT_F16
```

6.34 GDFLIB_FILTER_IIR1_DEFAULT_F32

Default value for GDFLIB_FILTER_IIR1_T_F32.

6.34.1 Source File

```
#include <GDFLIB_FilterIIR1.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.34.2 Macro Definition

```
#define GDFLIB_FILTER_IIR1_DEFAULT_F32 {{(tFrac32)0,(tFrac32)0,
(tFrac32)0},{(tFrac32)0},{(tFrac32)0}}
```

6.35 GDFLIB_FILTER_IIR1_DEFAULT_F16

Default value for GDFLIB_FILTER_IIR1_T_F16.

6.35.1 Source File

```
#include <GDFLIB_FilterIIR1.h>
```

6.35.2 Macro Definition

```
#define GDFLIB_FILTER_IIR1_DEFAULT_F16 {{(tFrac16)0,(tFrac16)0,
(tFrac16)0},{(tFrac16)0},{(tFrac16)0}}
```

6.36 GDFLIB_FILTER_IIR1_DEFAULT_FLT

Default value for GDFLIB_FILTER_IIR1_T_FLT.

6.36.1 Source File

```
#include <GDFLIB_FilterIIR1.h>
```

6.36.2 Macro Definition

```
#define GDFLIB_FILTER_IIR1_DEFAULT_FLT {{(tFloat)0,(tFloat)0,
(tFloat)0},{(tFloat)0},{(tFloat)0}}
```

6.37 GDFLIB_FILTER_IIR2_T

Definition of GDFLIB_FILTER_IIR2_T as alias for GDFLIB_FILTER_IIR2_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.37.1 Source File

```
#include <GDFLIB_FilterIIR2.h>
```

6.37.2 Macro Definition

```
#define GDFLIB_FILTER_IIR2_T GDFLIB_FILTER_IIR2_T_F16
```

6.38 GDFLIB_FILTER_IIR2_DEFAULT

Definition of GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.38.1 Source File

```
#include <GDFLIB_FilterIIR2.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.38.2 Macro Definition

```
#define GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_F16
```

6.39 GDFLIB_FILTER_IIR2_DEFAULT_F32

Default value for GDFLIB_FILTER_IIR2_T_F32.

6.39.1 Source File

```
#include <GDFLIB_FilterIIR2.h>
```

6.39.2 Macro Definition

```
#define GDFLIB_FILTER_IIR2_DEFAULT_F32 {{(tFrac32)0,(tFrac32)0,
(tFrac32)0,(tFrac32)0,(tFrac32)0},{(tFrac32)0,(tFrac32)0},
{(tFrac32)0,(tFrac32)0}}
```

6.40 GDFLIB_FILTER_IIR2_DEFAULT_F16

Default value for GDFLIB_FILTER_IIR2_T_F16.

6.40.1 Source File

```
#include <GDFLIB_FilterIIR2.h>
```

6.40.2 Macro Definition

```
#define GDFLIB_FILTER_IIR2_DEFAULT_F16 {{(tFrac16)0,(tFrac16)0,
(tFrac16)0,(tFrac16)0,(tFrac16)0},{(tFrac16)0,(tFrac16)0},
{(tFrac16)0,(tFrac16)0}}
```

6.41 GDFLIB_FILTER_IIR2_DEFAULT_FLT

Default value for GDFLIB_FILTER_IIR2_T_FLT.

6.41.1 Source File

```
#include <GDFLIB_FilterIIR2.h>
```

6.41.2 Macro Definition

```
#define GDFLIB_FILTER_IIR2_DEFAULT_FLT {{(tFloat)0,(tFloat)0,
(tFloat)0,(tFloat)0,(tFloat)0},{(tFloat)0,(tFloat)0},{(tFloat)0,
(tFloat)0}}
```

6.42 GDFLIB_FILTER_MA_T

Definition of GDFLIB_FILTER_MA_T as alias for GDFLIB_FILTER_MA_T_F16 datatype in case the 16-bit fractional implementation is selected.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.42.1 Source File

```
#include <GDFLIB_FilterMA.h>
```

6.42.2 Macro Definition

```
#define GDFLIB_FILTER_MA_T GDFLIB_FILTER_MA_T_F16
```

6.43 GDFLIB_FILTER_MA_DEFAULT

Definition of GDFLIB_FILTER_MA_DEFAULT as alias for GDFLIB_FILTER_MA_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.43.1 Source File

```
#include <GDFLIB_FilterMA.h>
```

6.43.2 Macro Definition

```
#define GDFLIB_FILTER_MA_DEFAULT GDFLIB_FILTER_MA_DEFAULT_F16
```

6.44 GDFLIB_FILTER_MA_DEFAULT_F32

Default value for GDFLIB_FILTER_MA_T_F32.

6.44.1 Source File

```
#include <GDFLIB_FilterMA.h>
```

6.44.2 Macro Definition

```
#define GDFLIB_FILTER_MA_DEFAULT_F32 {0,0}
```

6.45 GDFLIB_FILTER_MA_DEFAULT_F16

Default value for GDFLIB_FILTER_MA_T_F16.

6.45.1 Source File

```
#include <GDFLIB_FilterMA.h>
```

6.45.2 Macro Definition

```
#define GDFLIB_FILTER_MA_DEFAULT_F16 {0,0}
```

6.46 GDFLIB_FILTER_MA_DEFAULT_FLT

Default value for GDFLIB_FILTER_MA_T_FLT.

6.46.1 Source File

```
#include <GDFLIB_FilterMA.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.46.2 Macro Definition

```
#define GDFLIB_FILTER_MA_DEFAULT_FLT {0,0}
```

6.47 GFLIB_ACOS_T

Definition of GFLIB_ACOS_T as alias for GFLIB_ACOS_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.47.1 Source File

```
#include <GFLIB_Acos.h>
```

6.47.2 Macro Definition

```
#define GFLIB_ACOS_T GFLIB_ACOS_T_F16
```

6.48 GFLIB_ACOS_DEFAULT

Definition of GFLIB_ACOS_DEFAULT as alias for GFLIB_ACOS_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.48.1 Source File

```
#include <GFLIB_Acos.h>
```

6.48.2 Macro Definition

```
#define GFLIB_ACOS_DEFAULT GFLIB_ACOS_DEFAULT_F16
```

6.49 GFLIB_ACOS_DEFAULT_F32

Default approximation coefficients for GFLIB_Acos_F32 function.

6.49.1 Source File

```
#include <GFLIB_Acos.h>
```

6.49.2 Macro Definition

```
#define GFLIB_ACOS_DEFAULT_F32 &f32gflibAcosCoef
```

6.50 GFLIB_ACOS_DEFAULT_F16

Default approximation coefficients for GFLIB_Acos_F16 function.

6.50.1 Source File

```
#include <GFLIB_Acos.h>
```

6.50.2 Macro Definition

```
#define GFLIB_ACOS_DEFAULT_F16 &f16gflibAcosCoef
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.51 GFLIB_ACOS_DEFAULT_FLT

Default approximation coefficients for GFLIB_Acos_FLT function.

6.51.1 Source File

```
#include <GFLIB_Acos.h>
```

6.51.2 Macro Definition

```
#define GFLIB_ACOS_DEFAULT_FLT &fltgflibAcosCoef
```

6.52 GFLIB_ASIN_FLT_MIN

Floating-point min. input value for computation.

6.52.1 Source File

```
#include <GFLIB_Asin.c>
```

6.52.2 Macro Definition

```
#define GFLIB_ASIN_FLT_MIN ((tFloat)0.005)
```

6.53 GFLIB_ASIN_FLT_INT1

Floating-point min. input value for computation in interval 1.

6.53.1 Source File

```
#include <GFLIB_Asin.c>
```

6.53.2 Macro Definition

```
#define GFLIB_ASIN_FLT_INT1 ((tFloat)0.41)
```

6.54 GFLIB_ASIN_T

Definition of GFLIB_ASIN_T as alias for GFLIB_ASIN_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.54.1 Source File

```
#include <GFLIB_Asin.h>
```

6.54.2 Macro Definition

```
#define GFLIB_ASIN_T GFLIB_ASIN_T_F16
```

6.55 GFLIB_ASIN_DEFAULT

Definition of GFLIB_ASIN_DEFAULT as alias for GFLIB_ASIN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.55.1 Source File

```
#include <GFLIB_Asin.h>
```

6.55.2 Macro Definition

```
#define GFLIB_ASIN_DEFAULT GFLIB_ASIN_DEFAULT_F16
```

6.56 GFLIB_ASIN_DEFAULT_F32

Default approximation coefficients for GFLIB_Asin_F32 function.

6.56.1 Source File

```
#include <GFLIB_Asin.h>
```

6.56.2 Macro Definition

```
#define GFLIB_ASIN_DEFAULT_F32 &f32gflibAsinCoef
```

6.57 GFLIB_ASIN_DEFAULT_F16

Default approximation coefficients for GFLIB_Asin_F16 function.

6.57.1 Source File

```
#include <GFLIB_Asin.h>
```

6.57.2 Macro Definition

```
#define GFLIB_ASIN_DEFAULT_F16 &f16gflibAsinCoef
```

6.58 GFLIB_ASIN_DEFAULT_FLT

Default approximation coefficients for GFLIB_Asin_FLT function.

6.58.1 Source File

```
#include <GFLIB_Asin.h>
```

6.58.2 Macro Definition

```
#define GFLIB_ASIN_DEFAULT_FLT &fltgflibAsinCoef
```

6.59 GFLIB_ATAN_T

Definition of GFLIB_ATAN_T as alias for GFLIB_ATAN_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.59.1 Source File

```
#include <GFLIB_Atan.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.59.2 Macro Definition

```
#define GFLIB_ATAN_T GFLIB_ATAN_T_F16
```

6.60 GFLIB_ATAN_DEFAULT

Definition of GFLIB_ATAN_DEFAULT as alias for GFLIB_ATAN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.60.1 Source File

```
#include <GFLIB_Atan.h>
```

6.60.2 Macro Definition

```
#define GFLIB_ATAN_DEFAULT GFLIB_ATAN_DEFAULT_F16
```

6.61 GFLIB_ATAN_DEFAULT_F32

Default approximation coefficients for GFLIB_Atan_F32 function.

6.61.1 Source File

```
#include <GFLIB_Atan.h>
```

6.61.2 Macro Definition

```
#define GFLIB_ATAN_DEFAULT_F32 &f32gflibAtanCoef
```

6.62 GFLIB_ATAN_DEFAULT_F16

Default approximation coefficients for GFLIB_Atan_F16 function.

6.62.1 Source File

```
#include <GFLIB_Atan.h>
```

6.62.2 Macro Definition

```
#define GFLIB_ATAN_DEFAULT_F16 &f16gflibAtanCoef
```

6.63 GFLIB_ATAN_DEFAULT_FLT

Default approximation coefficients for GFLIB_Atan_FLT function.

6.63.1 Source File

```
#include <GFLIB_Atan.h>
```

6.63.2 Macro Definition

```
#define GFLIB_ATAN_DEFAULT_FLT &fltgflibAtanCoef
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.64 GFLIB_ATANYXSHIFTED_T

Definition of GFLIB_ATANYXSHIFTED_T as alias for GFLIB_ATANYXSHIFTED_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.64.1 Source File

```
#include <GFLIB_AtanYXShifted.h>
```

6.64.2 Macro Definition

```
#define GFLIB_ATANYXSHIFTED_T GFLIB_ATANYXSHIFTED_T_F16
```

6.65 GFLIB_CONTROLLER_PID_P_AW_T

Definition of GFLIB_CONTROLLER_PID_P_AW_T as alias for GFLIB_CONTROLLER_PID_P_AW_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.65.1 Source File

```
#include <GFLIB_ControllerPIDpAW.h>
```

6.65.2 Macro Definition

```
#define GFLIB_CONTROLLER_PID_P_AW_T
GFLIB_CONTROLLER_PID_P_AW_T_F16
```

6.66 GFLIB_CONTROLLER_PID_P_AW_DEFAULT

Definition of GFLIB_CONTROLLER_PID_P_AW_DEFAULT as alias for GFLIB_CONTROLLER_PID_P_AW_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.66.1 Source File

```
#include <GFLIB_ControllerPIDpAW.h>
```

6.66.2 Macro Definition

```
#define GFLIB_CONTROLLER_PID_P_AW_DEFAULT
GFLIB_CONTROLLER_PID_P_AW_DEFAULT_F16
```

6.67 GFLIB_CONTROLLER_PID_P_AW_DEFAULT_F32

Default value for GFLIB_CONTROLLER_PID_P_AW_T_F32.

6.67.1 Source File

```
#include <GFLIB_ControllerPIDpAW.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.67.2 Macro Definition

```
#define GFLIB_CONTROLLER_PID_P_AW_DEFAULT_F32 {(tFrac32)0,
(tFrac32)0,(tFrac32)0,(tFrac32)0,(tS16)0,(tS16)0,(tS16)0,
(tFrac32)0,(tFrac32)0,(tFrac32)0,(tFrac32)0,(tFrac32)0,(tU16)0}
```

6.68 GFLIB_CONTROLLER_PID_P_AW_DEFAULT_F16

Default value for GFLIB_CONTROLLER_PID_P_AW_T_F16.

6.68.1 Source File

```
#include <GFLIB_ControllerPIDpAW.h>
```

6.68.2 Macro Definition

```
#define GFLIB_CONTROLLER_PID_P_AW_DEFAULT_F16 {(tFrac16)0,
(tFrac16)0,(tFrac16)0,(tFrac16)0,(tS16)0,(tS16)0,(tS16)0,
(tFrac16)0,(tFrac16)0,(tFrac32)0,(tFrac16)0,(tFrac16)0,(tU16)0}
```

6.69 GFLIB_CONTROLLER_PID_P_AW_DEFAULT_FLT

Default value for GFLIB_CONTROLLER_PID_P_AW_T_FLT.

6.69.1 Source File

```
#include <GFLIB_ControllerPIDpAW.h>
```

6.69.2 Macro Definition

```
#define GFLIB_CONTROLLER_PID_P_AW_DEFAULT_FLT {(tFloat)0,
(tFloat)0,(tFloat)0,(tFloat)0,(tFloat)0,(tFloat)0,(tFloat)0,
(tFloat)0,(tFloat)0,(tU16)0}
```

6.70 GFLIB_CONTROLLER_PI_P_T

Definition of GFLIB_CONTROLLER_PI_P_T as alias for GFLIB_CONTROLLER_PI_P_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.70.1 Source File

```
#include <GFLIB_ControllerPIp.h>
```

6.70.2 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_T GFLIB_CONTROLLER_PI_P_T_F16
```

6.71 GFLIB_CONTROLLER_PI_P_DEFAULT

Definition of GFLIB_CONTROLLER_PI_P_DEFAULT as alias for GFLIB_CONTROLLER_PI_P_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.71.1 Source File

```
#include <GFLIB_ControllerPIp.h>
```

6.71.2 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_DEFAULT
GFLIB_CONTROLLER_PI_P_DEFAULT_F16
```

6.72 GFLIB_CONTROLLER_PI_P_DEFAULT_F32

Default value for GFLIB_CONTROLLER_PI_P_T_F32.

6.72.1 Source File

```
#include <GFLIB_ControllerPIp.h>
```

6.72.2 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_DEFAULT_F32 {(tFrac32)0,(tFrac32)0,
(tS16)0,(tS16)0,(tFrac32)0,(tFrac32)0}
```

6.73 GFLIB_CONTROLLER_PI_P_DEFAULT_F16

Default value for GFLIB_CONTROLLER_PI_P_T_F16.

6.73.1 Source File

```
#include <GFLIB_ControllerPIp.h>
```

6.73.2 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_DEFAULT_F16 {(tFrac16)0,(tFrac16)0,
(tS16)0,(tS16)0,(tFrac32)0,(tFrac16)0}
```

6.74 GFLIB_CONTROLLER_PI_P_DEFAULT_FLT

Default value for GFLIB_CONTROLLER_PI_P_T_FLT.

6.74.1 Source File

```
#include <GFLIB_ControllerPIp.h>
```

6.74.2 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_DEFAULT_FLT {(tFloat)0,(tFloat)0,
(tFloat)0,(tFloat)0}
```

6.75 GFLIB_CONTROLLER_PIAW_P_T

Definition of GFLIB_CONTROLLER_PIAW_P_T as alias for GFLIB_CONTROLLER_PIAW_P_T_F16 datatype in case the 16-bit fractional implementation is selected.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.75.1 Source File

```
#include <GFLIB_ControllerPIpAW.h>
```

6.75.2 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_T GFLIB_CONTROLLER_PIAW_P_T_F16
```

6.76 GFLIB_CONTROLLER_PIAW_P_DEFAULT

Definition of GFLIB_CONTROLLER_PIAW_P_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.76.1 Source File

```
#include <GFLIB_ControllerPIpAW.h>
```

6.76.2 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_DEFAULT
GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16
```

6.77 GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32

Default value for GFLIB_CONTROLLER_PIAW_P_T_F32.

6.77.1 Source File

```
#include <GFLIB_ControllerPIpAW.h>
```

6.77.2 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32 {(tFrac32)0,
(tFrac32)0,(tS32)0,(tS32)0,INT32_MIN,INT32_MAX,(tFrac32)0,
(tFrac32)0,(tU16)0}
```

6.78 GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16

Default value for GFLIB_CONTROLLER_PIAW_P_T_F16.

6.78.1 Source File

```
#include <GFLIB_ControllerPIpAW.h>
```

6.78.2 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16 {(tFrac16)0,
(tFrac16)0,(tS16)0,(tS16)0,INT16_MIN,INT16_MAX,(tFrac32)0,
(tFrac16)0,(tU16)0}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.79 GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT

Default value for GFLIB_CONTROLLER_PIAW_P_T_FLT.

6.79.1 Source File

```
#include <GFLIB_ControllerPIpAW.h>
```

6.79.2 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT {(tFloat)0,
(tFloat)0,FLOAT_MIN,FLOAT_MAX,(tFloat)0,(tFloat)0,(tU16)0}
```

6.80 GFLIB_CONTROLLER_PI_R_T

Definition of GFLIB_CONTROLLER_PI_R_T as alias for GFLIB_CONTROLLER_PI_R_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.80.1 Source File

```
#include <GFLIB_ControllerPIr.h>
```

6.80.2 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_T GFLIB_CONTROLLER_PI_R_T_F16
```

6.81 GFLIB_CONTROLLER_PI_R_DEFAULT

Definition of GFLIB_CONTROLLER_PI_R_DEFAULT as alias for GFLIB_CONTROLLER_PI_R_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.81.1 Source File

```
#include <GFLIB_ControllerPIr.h>
```

6.81.2 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_DEFAULT
GFLIB_CONTROLLER_PI_R_DEFAULT_F16
```

6.82 GFLIB_CONTROLLER_PI_R_DEFAULT_F32

Default value for GFLIB_CONTROLLER_PI_R_T_F32.

6.82.1 Source File

```
#include <GFLIB_ControllerPIr.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.82.2 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_DEFAULT_F32 {(tFrac32)0,(tFrac32)0,
(tFrac32)0,(tFrac32)0,(tU16)0}
```

6.83 GFLIB_CONTROLLER_PI_R_DEFAULT_F16

Default value for GFLIB_CONTROLLER_PI_R_T_F16.

6.83.1 Source File

```
#include <GFLIB_ControllerPIr.h>
```

6.83.2 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_DEFAULT_F16 {(tFrac16)0,(tFrac16)0,
(tFrac32)0,(tFrac16)0,(tU16)0}
```

6.84 GFLIB_CONTROLLER_PI_R_DEFAULT_FLT

Default value for GFLIB_CONTROLLER_PI_R_T_FLT.

6.84.1 Source File

```
#include <GFLIB_ControllerPIr.h>
```

6.84.2 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_DEFAULT_FLT {(tFloat)0,(tFloat)0,
(tFloat)0,(tFloat)0}
```

6.85 GFLIB_CONTROLLER_PIAW_R_T

Definition of GFLIB_CONTROLLER_PIAW_R_T as alias for GFLIB_CONTROLLER_PIAW_R_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.85.1 Source File

```
#include <GFLIB_ControllerPIrAW.h>
```

6.85.2 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_T GFLIB_CONTROLLER_PIAW_R_T_F16
```

6.86 GFLIB_CONTROLLER_PIAW_R_DEFAULT

Definition of GFLIB_CONTROLLER_PIAW_R_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

© NXP B.V. 2023. All rights reserved.

6.86.1 Source File

```
#include <GFLIB_ControllerPIrAW.h>
```

6.86.2 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_DEFAULT
GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16
```

6.87 GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32

Default value for GFLIB_CONTROLLER_PIAW_R_T_F32.

6.87.1 Source File

```
#include <GFLIB_ControllerPIrAW.h>
```

6.87.2 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32 {(tFrac32)0,
(tFrac32)0,(tFrac32)0,(tFrac32)0,INT32_MIN,INT32_MAX,(tU16)0}
```

6.88 GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16

Default value for GFLIB_CONTROLLER_PIAW_R_T_F16.

6.88.1 Source File

```
#include <GFLIB_ControllerPIrAW.h>
```

6.88.2 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16 {(tFrac16)0,
(tFrac16)0,(tFrac32)0,(tFrac16)0,INT16_MIN,INT16_MAX,(tU16)0}
```

6.89 GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT

Default value for GFLIB_CONTROLLER_PIAW_R_T_FLT.

6.89.1 Source File

```
#include <GFLIB_ControllerPIrAW.h>
```

6.89.2 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT {(tFloat)0,(tFloat)0,
(tFloat)0,(tFloat)0,FLOAT_MIN,FLOAT_MAX}
```

6.90 GFLIB_COS_T

Definition of GFLIB_COS_T as alias for GFLIB_COS_T_F16 datatype in case the 16-bit fractional implementation is selected.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.90.1 Source File

```
#include <GFLIB_Cos.h>
```

6.90.2 Macro Definition

```
#define GFLIB_COS_T GFLIB_COS_T_F16
```

6.91 GFLIB_COS_DEFAULT

Definition of GFLIB_COS_DEFAULT as alias for GFLIB_COS_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.91.1 Source File

```
#include <GFLIB_Cos.h>
```

6.91.2 Macro Definition

```
#define GFLIB_COS_DEFAULT GFLIB_COS_DEFAULT_F16
```

6.92 GFLIB_COS_DEFAULT_F32

Default approximation coefficients for GFLIB_Cos_F32 function.

6.92.1 Source File

```
#include <GFLIB_Cos.h>
```

6.92.2 Macro Definition

```
#define GFLIB_COS_DEFAULT_F32 &f32gflibCosCoef
```

6.93 GFLIB_COS_DEFAULT_F16

Default approximation coefficients for GFLIB_Cos_F32 function.

6.93.1 Source File

```
#include <GFLIB_Cos.h>
```

6.93.2 Macro Definition

```
#define GFLIB_COS_DEFAULT_F16 &f16gflibCosCoef
```

6.94 GFLIB_COS_DEFAULT_FLT

Default approximation coefficients for GFLIB_Cos_FLT function.

6.94.1 Source File

```
#include <GFLIB_Cos.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.94.2 Macro Definition

```
#define GFLIB_COS_DEFAULT_FLT &fltgflibCosCoef
```

6.95 GFLIB_HYST_T

Definition of GFLIB_HYST_T as alias for GFLIB_HYST_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.95.1 Source File

```
#include <GFLIB_Hyst.h>
```

6.95.2 Macro Definition

```
#define GFLIB_HYST_T GFLIB_HYST_T_F16
```

6.96 GFLIB_HYST_DEFAULT

Definition of GFLIB_HYST_DEFAULT as alias for GFLIB_HYST_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.96.1 Source File

```
#include <GFLIB_Hyst.h>
```

6.96.2 Macro Definition

```
#define GFLIB_HYST_DEFAULT GFLIB_HYST_DEFAULT_F16
```

6.97 GFLIB_HYST_DEFAULT_F32

Default value for GFLIB_HYST_T_F32.

6.97.1 Source File

```
#include <GFLIB_Hyst.h>
```

6.97.2 Macro Definition

```
#define GFLIB_HYST_DEFAULT_F32 {(tFrac32)0,(tFrac32)0,(tFrac32)0,
(tFrac32)0,(tFrac32)0}
```

6.98 GFLIB_HYST_DEFAULT_F16

Default value for GFLIB_HYST_T_F16.

6.98.1 Source File

```
#include <GFLIB_Hyst.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.98.2 Macro Definition

```
#define GFLIB_HYST_DEFAULT_F16 {(tFrac16)0,(tFrac16)0,(tFrac16)0,
(tFrac16)0,(tFrac16)0}
```

6.99 GFLIB_HYST_DEFAULT_FLT

Default value for GFLIB_HYST_T_FLT.

6.99.1 Source File

```
#include <GFLIB_Hyst.h>
```

6.99.2 Macro Definition

```
#define GFLIB_HYST_DEFAULT_FLT {(tFloat)0,(tFloat)0,(tFloat)0,
(tFloat)0,(tFloat)0}
```

6.100 GFLIB_INTEGRATOR_TR_T

Definition of GFLIB_INTEGRATOR_TR_T as alias for GFLIB_INTEGRATOR_TR_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.100.1 Source File

```
#include <GFLIB_IntegratorTR.h>
```

6.100.2 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_T GFLIB_INTEGRATOR_TR_T_F16
```

6.101 GFLIB_INTEGRATOR_TR_DEFAULT

Definition of GFLIB_INTEGRATOR_TR_DEFAULT as alias for GFLIB_INTEGRATOR_TR_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.101.1 Source File

```
#include <GFLIB_IntegratorTR.h>
```

6.101.2 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_DEFAULT
GFLIB_INTEGRATOR_TR_DEFAULT_F16
```

6.102 GFLIB_INTEGRATOR_TR_DEFAULT_F32

Default value for GFLIB_INTEGRATOR_TR_T_F32.

6.102.1 Source File

```
#include <GFLIB_IntegratorTR.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.102.2 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_DEFAULT_F32 {(tFrac32)0,(tFrac32)0,
(tFrac32)0,(tU16)0}
```

6.103 GFLIB_INTEGRATOR_TR_DEFAULT_F16

Default value for GFLIB_INTEGRATOR_TR_T_F16.

6.103.1 Source File

```
#include <GFLIB_IntegratorTR.h>
```

6.103.2 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_DEFAULT_F16 {(tFrac32)0,(tFrac16)0,
(tFrac16)0,(tU16)0}
```

6.104 GFLIB_INTEGRATOR_TR_DEFAULT_FLT

Default value for GFLIB_INTEGRATOR_TR_T_FLT.

6.104.1 Source File

```
#include <GFLIB_IntegratorTR.h>
```

6.104.2 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_DEFAULT_FLT {(tFloat)0,(tFloat)0,
(tFloat)0}
```

6.105 GFLIB_LIMIT_T

Definition of GFLIB_LIMIT_T as alias for GFLIB_LIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.105.1 Source File

```
#include <GFLIB_Limit.h>
```

6.105.2 Macro Definition

```
#define GFLIB_LIMIT_T GFLIB_LIMIT_T_F16
```

6.106 GFLIB_LIMIT_DEFAULT

Definition of GFLIB_LIMIT_DEFAULT as alias for GFLIB_LIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.106.1 Source File

```
#include <GFLIB_Limit.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.106.2 Macro Definition

```
#define GFLIB_LIMIT_DEFAULT GFLIB_LIMIT_DEFAULT_F16
```

6.107 GFLIB_LIMIT_DEFAULT_F32

Default value for GFLIB_LIMIT_T_F32.

6.107.1 Source File

```
#include <GFLIB_Limit.h>
```

6.107.2 Macro Definition

```
#define GFLIB_LIMIT_DEFAULT_F32 {INT32_MIN,INT32_MAX}
```

6.108 GFLIB_LIMIT_DEFAULT_F16

Default value for GFLIB_LIMIT_T_F16.

6.108.1 Source File

```
#include <GFLIB_Limit.h>
```

6.108.2 Macro Definition

```
#define GFLIB_LIMIT_DEFAULT_F16 {INT16_MIN,INT16_MAX}
```

6.109 GFLIB_LIMIT_DEFAULT_FLT

Default value for GFLIB_LIMIT_T_FLT.

6.109.1 Source File

```
#include <GFLIB_Limit.h>
```

6.109.2 Macro Definition

```
#define GFLIB_LIMIT_DEFAULT_FLT {FLOAT_MIN,FLOAT_MAX}
```

6.110 GFLIB_LOG10_DEFAULT_FLT

Default approximation coefficients for GFLIB_Log10_FLT function.

6.110.1 Source File

```
#include <GFLIB_Log10.h>
```

6.110.2 Macro Definition

```
#define GFLIB_LOG10_DEFAULT_FLT &fltgflibLog10Coef
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.111 GFLIB_LOWERLIMIT_T

Definition of GFLIB_LOWERLIMIT_T as alias for GFLIB_LOWERLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.111.1 Source File

```
#include <GFLIB_LowerLimit.h>
```

6.111.2 Macro Definition

```
#define GFLIB_LOWERLIMIT_T GFLIB_LOWERLIMIT_T_F16
```

6.112 GFLIB_LOWERLIMIT_DEFAULT

Definition of GFLIB_LOWERLIMIT_DEFAULT as alias for GFLIB_LOWERLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.112.1 Source File

```
#include <GFLIB_LowerLimit.h>
```

6.112.2 Macro Definition

```
#define GFLIB_LOWERLIMIT_DEFAULT GFLIB_LOWERLIMIT_DEFAULT_F16
```

6.113 GFLIB_LOWERLIMIT_DEFAULT_F32

Default value for GFLIB_LOWERLIMIT_T_F32.

6.113.1 Source File

```
#include <GFLIB_LowerLimit.h>
```

6.113.2 Macro Definition

```
#define GFLIB_LOWERLIMIT_DEFAULT_F32 {INT32_MIN}
```

6.114 GFLIB_LOWERLIMIT_DEFAULT_F16

Default value for GFLIB_LOWERLIMIT_T_F16.

6.114.1 Source File

```
#include <GFLIB_LowerLimit.h>
```

6.114.2 Macro Definition

```
#define GFLIB_LOWERLIMIT_DEFAULT_F16 {INT16_MIN}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.115 GFLIB_LOWERLIMIT_DEFAULT_FLT

Default value for GFLIB_LOWERLIMIT_T_FLT.

6.115.1 Source File

```
#include <GFLIB_LowerLimit.h>
```

6.115.2 Macro Definition

```
#define GFLIB_LOWERLIMIT_DEFAULT_FLT {FLOAT_MIN}
```

6.116 GFLIB_LUT1D_T

Definition of GFLIB_LUT1D_T as alias for GFLIB_LUT1D_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.116.1 Source File

```
#include <GFLIB_Lut1D.h>
```

6.116.2 Macro Definition

```
#define GFLIB_LUT1D_T GFLIB_LUT1D_T_F16
```

6.117 GFLIB_LUT1D_DEFAULT

Definition of GFLIB_LUT1D_DEFAULT as alias for GFLIB_LUT1D_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.117.1 Source File

```
#include <GFLIB_Lut1D.h>
```

6.117.2 Macro Definition

```
#define GFLIB_LUT1D_DEFAULT GFLIB_LUT1D_DEFAULT_F16
```

6.118 GFLIB_LUT1D_DEFAULT_F32

Default value for GFLIB_LUT1D_T_F32.

6.118.1 Source File

```
#include <GFLIB_Lut1D.h>
```

6.118.2 Macro Definition

```
#define GFLIB_LUT1D_DEFAULT_F32 {(tU32)0,(tFrac32*)0}
```

6.119 GFLIB_LUT1D_DEFAULT_F16

Default value for GFLIB_LUT1D_T_F16.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.119.1 Source File

```
#include <GFLIB_Lut1D.h>
```

6.119.2 Macro Definition

```
#define GFLIB_LUT1D_DEFAULT_F16 {(tU16)0,(tFrac16*)0}
```

6.120 GFLIB_LUT1D_DEFAULT_FLT

Default value for GFLIB_LUT1D_T_FLT.

6.120.1 Source File

```
#include <GFLIB_Lut1D.h>
```

6.120.2 Macro Definition

```
#define GFLIB_LUT1D_DEFAULT_FLT {(tU32)0,(tFloat*)0}
```

6.121 GFLIB_LUT2D_T

Definition of GFLIB_LUT2D_T as alias for GFLIB_LUT2D_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.121.1 Source File

```
#include <GFLIB_Lut2D.h>
```

6.121.2 Macro Definition

```
#define GFLIB_LUT2D_T GFLIB_LUT2D_T_F16
```

6.122 GFLIB_LUT2D_DEFAULT

Definition of GFLIB_LUT2D_DEFAULT as alias for GFLIB_LUT2D_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.122.1 Source File

```
#include <GFLIB_Lut2D.h>
```

6.122.2 Macro Definition

```
#define GFLIB_LUT2D_DEFAULT GFLIB_LUT2D_DEFAULT_F16
```

6.123 GFLIB_LUT2D_DEFAULT_F32

Default value for GFLIB_LUT2D_T_F32.

6.123.1 Source File

```
#include <GFLIB_Lut2D.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.123.2 Macro Definition

```
#define GFLIB_LUT2D_DEFAULT_F32 {(tU32)0,(tU32)0,(tFrac32*)0}
```

6.124 GFLIB_LUT2D_DEFAULT_F16

Default value for GFLIB_LUT2D_T_F16.

6.124.1 Source File

```
#include <GFLIB_Lut2D.h>
```

6.124.2 Macro Definition

```
#define GFLIB_LUT2D_DEFAULT_F16 {(tU16)0,(tU16)0,(tFrac16*)0}
```

6.125 GFLIB_LUT2D_DEFAULT_FLT

Default value for GFLIB_LUT2D_T_FLT.

6.125.1 Source File

```
#include <GFLIB_Lut2D.h>
```

6.125.2 Macro Definition

```
#define GFLIB_LUT2D_DEFAULT_FLT {(tU32)0,(tU32)0,(tFloat*)0}
```

6.126 GFLIB_RAMP_T

Definition of GFLIB_RAMP_T as alias for GFLIB_RAMP_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.126.1 Source File

```
#include <GFLIB_Ramp.h>
```

6.126.2 Macro Definition

```
#define GFLIB_RAMP_T GFLIB_RAMP_T_F16
```

6.127 GFLIB_RAMP_DEFAULT

Definition of GFLIB_RAMP_DEFAULT as alias for GFLIB_RAMP_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.127.1 Source File

```
#include <GFLIB_Ramp.h>
```

6.127.2 Macro Definition

```
#define GFLIB_RAMP_DEFAULT GFLIB_RAMP_DEFAULT_F16
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.128 GFLIB_RAMP_DEFAULT_F32

Default value for GFLIB_RAMP_T_F32.

6.128.1 Source File

```
#include <GFLIB_Ramp.h>
```

6.128.2 Macro Definition

```
#define GFLIB_RAMP_DEFAULT_F32 {(tFrac32)0,(tFrac32)0,(tFrac32)0}
```

6.129 GFLIB_RAMP_DEFAULT_F16

Default value for GFLIB_RAMP_T_F16.

6.129.1 Source File

```
#include <GFLIB_Ramp.h>
```

6.129.2 Macro Definition

```
#define GFLIB_RAMP_DEFAULT_F16 {(tFrac16)0,(tFrac16)0,(tFrac16)0}
```

6.130 GFLIB_RAMP_DEFAULT_FLT

Default value for GFLIB_RAMP_T_FLT.

6.130.1 Source File

```
#include <GFLIB_Ramp.h>
```

6.130.2 Macro Definition

```
#define GFLIB_RAMP_DEFAULT_FLT {(tFloat)0,(tFloat)0,(tFloat)0}
```

6.131 GFLIB_SIN_FLT_MIN

Floating-point min. input value for computation.

6.131.1 Source File

```
#include <GFLIB_Sin.c>
```

6.131.2 Macro Definition

```
#define GFLIB_SIN_FLT_MIN ((tFloat)1.220703125000000e-04)
```

6.132 GFLIB_SIN_T

Definition of GFLIB_SIN_T as alias for GFLIB_SIN_T_F16 datatype in case the 16-bit fractional implementation is selected.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.132.1 Source File

```
#include <GFLIB_Sin.h>
```

6.132.2 Macro Definition

```
#define GFLIB_SIN_T GFLIB_SIN_T_F16
```

6.133 GFLIB_SIN_DEFAULT

Definition of GFLIB_SIN_DEFAULT as alias for GFLIB_SIN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.133.1 Source File

```
#include <GFLIB_Sin.h>
```

6.133.2 Macro Definition

```
#define GFLIB_SIN_DEFAULT GFLIB_SIN_DEFAULT_F16
```

6.134 GFLIB_SIN_DEFAULT_F32

Default approximation coefficients for GFLIB_Sin_F32 function.

6.134.1 Source File

```
#include <GFLIB_Sin.h>
```

6.134.2 Macro Definition

```
#define GFLIB_SIN_DEFAULT_F32 &f32gflibSinCoef
```

6.135 GFLIB_SIN_DEFAULT_F16

Default approximation coefficients for GFLIB_Sin_F16 function.

6.135.1 Source File

```
#include <GFLIB_Sin.h>
```

6.135.2 Macro Definition

```
#define GFLIB_SIN_DEFAULT_F16 &f16gflibSinCoef
```

6.136 GFLIB_SIN_DEFAULT_FLT

Default approximation coefficients for GFLIB_Sin_FLT function.

6.136.1 Source File

```
#include <GFLIB_Sin.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.136.2 Macro Definition

```
#define GFLIB_SIN_DEFAULT_FLT &fltgflibSinCoef
```

6.137 GFLIB_SINCOS_FLT_MIN

Floating-point min. input value for computation.

6.137.1 Source File

```
#include <GFLIB_SinCos.c>
```

6.137.2 Macro Definition

```
#define GFLIB_SINCOS_FLT_MIN ((tFloat)1.220703125000000e-04)
```

6.138 GFLIB_SINCOS_T

Definition of GFLIB_SINCOS_T as alias for GFLIB_SINCOS_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.138.1 Source File

```
#include <GFLIB_SinCos.h>
```

6.138.2 Macro Definition

```
#define GFLIB_SINCOS_T GFLIB_SINCOS_T_F16
```

6.139 GFLIB_SINCOS_DEFAULT

Definition of GFLIB_SINCOS_DEFAULT as alias for GFLIB_SINCOS_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.139.1 Source File

```
#include <GFLIB_SinCos.h>
```

6.139.2 Macro Definition

```
#define GFLIB_SINCOS_DEFAULT GFLIB_SINCOS_DEFAULT_F16
```

6.140 GFLIB_SINCOS_DEFAULT_F32

Default approximation coefficients for GFLIB_SinCos_F32 function.

6.140.1 Source File

```
#include <GFLIB_SinCos.h>
```

6.140.2 Macro Definition

```
#define GFLIB_SINCOS_DEFAULT_F32 &f32gflibSinCosCoef
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.141 GFLIB_SINCOS_DEFAULT_F16

Default approximation coefficients for GFLIB_SinCos_F16 function.

6.141.1 Source File

```
#include <GFLIB_SinCos.h>
```

6.141.2 Macro Definition

```
#define GFLIB_SINCOS_DEFAULT_F16 &f16gflibSinCosCoef
```

6.142 GFLIB_SINCOS_DEFAULT_FLT

Default approximation coefficients for GFLIB_SinCos_FLT function.

6.142.1 Source File

```
#include <GFLIB_SinCos.h>
```

6.142.2 Macro Definition

```
#define GFLIB_SINCOS_DEFAULT_FLT &fltgflibSinCosCoef
```

6.143 GFLIB_TAN_FLT_MIN

Floating-point min. input value for computation.

6.143.1 Source File

```
#include <GFLIB_Tan.c>
```

6.143.2 Macro Definition

```
#define GFLIB_TAN_FLT_MIN ((tFloat)2.441406250000000e-04)
```

6.144 GFLIB_TAN_FLT_3P4

Floating-point constant 3*PI/4.

6.144.1 Source File

```
#include <GFLIB_Tan.c>
```

6.144.2 Macro Definition

```
#define GFLIB_TAN_FLT_3P4 ((tFloat)2.356194490192345e+00)
```

6.145 GFLIB_TAN_FLT_PI

Floating-point constant PI.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.145.1 Source File

```
#include <GFLIB_Tan.c>
```

6.145.2 Macro Definition

```
#define GFLIB_TAN_FLT_PI ((tFloat)3.141592653589793e+00)
```

6.146 GFLIB_TAN_FLT_CORR1

Floating-point correction constant PI_double - PI_single.

6.146.1 Source File

```
#include <GFLIB_Tan.c>
```

6.146.2 Macro Definition

```
#define GFLIB_TAN_FLT_CORR1 ((tFloat)-8.742278012618954e-08)
```

6.147 GFLIB_TAN_FLT_PI4

Floating-point constant PI/4.

6.147.1 Source File

```
#include <GFLIB_Tan.c>
```

6.147.2 Macro Definition

```
#define GFLIB_TAN_FLT_PI4 ((tFloat)7.853981633974483e-01)
```

6.148 GFLIB_TAN_FLT_PI2

Floating-point constant PI/2.

6.148.1 Source File

```
#include <GFLIB_Tan.c>
```

6.148.2 Macro Definition

```
#define GFLIB_TAN_FLT_PI2 ((tFloat)1.570796326794897e+00)
```

6.149 GFLIB_TAN_FLT_CORR2

Floating-point correction constant PI_double/2 - PI_single/2.

6.149.1 Source File

```
#include <GFLIB_Tan.c>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.149.2 Macro Definition

```
#define GFLIB_TAN_FLT_CORR2 ((tFloat)-4.371139006309477e-08)
```

6.150 GFLIB_TAN_T

Definition of GFLIB_TAN_T as alias for GFLIB_TAN_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.150.1 Source File

```
#include <GFLIB_Tan.h>
```

6.150.2 Macro Definition

```
#define GFLIB_TAN_T GFLIB_TAN_T_F16
```

6.151 GFLIB_TAN_DEFAULT

Definition of GFLIB_TAN_DEFAULT as alias for GFLIB_TAN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.151.1 Source File

```
#include <GFLIB_Tan.h>
```

6.151.2 Macro Definition

```
#define GFLIB_TAN_DEFAULT GFLIB_TAN_DEFAULT_F16
```

6.152 GFLIB_TAN_DEFAULT_F32

Default approximation coefficients for GFLIB_Tan_F32 function.

6.152.1 Source File

```
#include <GFLIB_Tan.h>
```

6.152.2 Macro Definition

```
#define GFLIB_TAN_DEFAULT_F32 &f32gflibTanCoef
```

6.153 GFLIB_TAN_DEFAULT_F16

Default approximation coefficients for GFLIB_Tan_F16 function.

6.153.1 Source File

```
#include <GFLIB_Tan.h>
```

6.153.2 Macro Definition

```
#define GFLIB_TAN_DEFAULT_F16 &f16gflibTanCoef
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.154 GFLIB_TAN_DEFAULT_FLT

Default approximation coefficients for GFLIB_Tan_FLT function.

6.154.1 Source File

```
#include <GFLIB_Tan.h>
```

6.154.2 Macro Definition

```
#define GFLIB_TAN_DEFAULT_FLT &fltgflibTanCoef
```

6.155 GFLIB_UPPERLIMIT_T

Definition of GFLIB_UPPERLIMIT_T as alias for GFLIB_UPPERLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.155.1 Source File

```
#include <GFLIB_UpperLimit.h>
```

6.155.2 Macro Definition

```
#define GFLIB_UPPERLIMIT_T GFLIB_UPPERLIMIT_T_F16
```

6.156 GFLIB_UPPERLIMIT_DEFAULT

Definition of GFLIB_UPPERLIMIT_DEFAULT as alias for GFLIB_UPPERLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.156.1 Source File

```
#include <GFLIB_UpperLimit.h>
```

6.156.2 Macro Definition

```
#define GFLIB_UPPERLIMIT_DEFAULT GFLIB_UPPERLIMIT_DEFAULT_F16
```

6.157 GFLIB_UPPERLIMIT_DEFAULT_F32

Default value for GFLIB_UPPERLIMIT_T_F32.

6.157.1 Source File

```
#include <GFLIB_UpperLimit.h>
```

6.157.2 Macro Definition

```
#define GFLIB_UPPERLIMIT_DEFAULT_F32 {INT32_MAX}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.158 GFLIB_UPPERLIMIT_DEFAULT_F16

Default value for GFLIB_UPPERLIMIT_T_F16.

6.158.1 Source File

```
#include <GFLIB_UpperLimit.h>
```

6.158.2 Macro Definition

```
#define GFLIB_UPPERLIMIT_DEFAULT_F16 {INT16_MAX}
```

6.159 GFLIB_UPPERLIMIT_DEFAULT_FLT

Default value for GFLIB_UPPERLIMIT_T_FLT.

6.159.1 Source File

```
#include <GFLIB_UpperLimit.h>
```

6.159.2 Macro Definition

```
#define GFLIB_UPPERLIMIT_DEFAULT_FLT {FLOAT_MAX}
```

6.160 GFLIB_VECTORLIMIT_T

Definition of GFLIB_VECTORLIMIT_T as alias for GFLIB_VECTORLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.160.1 Source File

```
#include <GFLIB_VectorLimit.h>
```

6.160.2 Macro Definition

```
#define GFLIB_VECTORLIMIT_T GFLIB_VECTORLIMIT_T_F16
```

6.161 GFLIB_VECTORLIMIT_DEFAULT

Definition of GFLIB_VECTORLIMIT_DEFAULT as alias for GFLIB_VECTORLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.161.1 Source File

```
#include <GFLIB_VectorLimit.h>
```

6.161.2 Macro Definition

```
#define GFLIB_VECTORLIMIT_DEFAULT GFLIB_VECTORLIMIT_DEFAULT_F16
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.162 GFLIB_VECTORLIMIT_DEFAULT_F32

Default value for GFLIB_VECTORLIMIT_T_F32.

6.162.1 Source File

```
#include <GFLIB_VectorLimit.h>
```

6.162.2 Macro Definition

```
#define GFLIB_VECTORLIMIT_DEFAULT_F32 {(tFrac32)0}
```

6.163 GFLIB_VECTORLIMIT_DEFAULT_F16

Default value for GFLIBVECTORLIMIT_T_F16.

6.163.1 Source File

```
#include <GFLIB_VectorLimit.h>
```

6.163.2 Macro Definition

```
#define GFLIB_VECTORLIMIT_DEFAULT_F16 {(tFrac16)0}
```

6.164 GFLIB_VECTORLIMIT_DEFAULT_FLT

Default value for GFLIB_VECTORLIMIT_T_FLT.

6.164.1 Source File

```
#include <GFLIB_VectorLimit.h>
```

6.164.2 Macro Definition

```
#define GFLIB_VECTORLIMIT_DEFAULT_FLT {(tFloat)0}
```

6.165 GFLIB_VLOG10_DEFAULT_FLT

Default approximation coefficients for GFLIB_VLog10_FLT function.

6.165.1 Source File

```
#include <GFLIB_VLog10.h>
```

6.165.2 Macro Definition

```
#define GFLIB_VLOG10_DEFAULT_FLT &fltgflibVLog10Coef
```

6.166 GMCLIB_DECOUPLINGPMSM_T

Definition of GMCLIB_DECOUPLINGPMSM_T as alias for GMCLIB_DECOUPLINGPMSM_T_F16 datatype in case the 16-bit fractional implementation is selected.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.166.1 Source File

```
#include <GMCLIB_DecouplingPMSM.h>
```

6.166.2 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_T GMCLIB_DECOUPLINGPMSM_T_F16
```

6.167 GMCLIB_DECOUPLINGPMSM_DEFAULT

Definition of GMCLIB_DECOUPLINGPMSM_DEFAULT as alias for GMCLIB_DECOUPLINGPMSM_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.167.1 Source File

```
#include <GMCLIB_DecouplingPMSM.h>
```

6.167.2 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT
GMCLIB_DECOUPLINGPMSM_DEFAULT_F16
```

6.168 GMCLIB_DECOUPLINGPMSM_DEFAULT_F32

Default value for GMCLIB_DECOUPLINGPMSM_T_F32.

6.168.1 Source File

```
#include <GMCLIB_DecouplingPMSM.h>
```

6.168.2 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT_F32 {(tFrac32)0,(tS16)0,
(tFrac32)0,(tS16)0}
```

6.169 GMCLIB_DECOUPLINGPMSM_DEFAULT_F16

Default value for GMCLIB_DECOUPLINGPMSM_T_F16.

6.169.1 Source File

```
#include <GMCLIB_DecouplingPMSM.h>
```

6.169.2 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT_F16 {(tFrac16)0,(tS16)0,
(tFrac16)0,(tS16)0}
```

6.170 GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT

Default value for GMCLIB_DECOUPLINGPMSM_T_FLT.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.170.1 Source File

```
#include <GMCLIB_DecouplingPMSM.h>
```

6.170.2 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT {(tFloat)0,(tFloat)0}
```

6.171 GMCLIB_ELIMDCBUSRIP_T

Definition of GMCLIB_ELIMDCBUSRIP_T as alias for GMCLIB_ELIMDCBUSRIP_T_F16 datatype in case the 16-bit fractional implementation is selected.

6.171.1 Source File

```
#include <GMCLIB_ElimDcBusRip.h>
```

6.171.2 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_T GMCLIB_ELIMDCBUSRIP_T_F16
```

6.172 GMCLIB_ELIMDCBUSRIP_DEFAULT

Definition of GMCLIB_ELIMDCBUSRIP_DEFAULT as alias for GMCLIB_ELIMDCBUSRIP_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

6.172.1 Source File

```
#include <GMCLIB_ElimDcBusRip.h>
```

6.172.2 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_DEFAULT
GMCLIB_ELIMDCBUSRIP_DEFAULT_F16
```

6.173 GMCLIB_ELIMDCBUSRIP_DEFAULT_F32

Default value for GMCLIB_ELIMDCBUSRIP_T_F32.

6.173.1 Source File

```
#include <GMCLIB_ElimDcBusRip.h>
```

6.173.2 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_DEFAULT_F32 {(tFrac32)0,(tFrac32)0}
```

6.174 GMCLIB_ELIMDCBUSRIP_DEFAULT_F16

Default value for GMCLIB_ELIMDCBUSRIP_T_F16.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.174.1 Source File

```
#include <GMCLIB_ElimDcBusRip.h>
```

6.174.2 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_DEFAULT_F16 {(tFrac16)0,(tFrac16)0}
```

6.175 GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT

Default value for GMCLIB_ELIMDCBUSRIP_T_FLT.

6.175.1 Source File

```
#include <GMCLIB_ElimDcBusRip.h>
```

6.175.2 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT {(tFloat)0,(tFloat)0}
```

6.176 SWLIBS_SUPPORT_F32

Enables/disables support of 32-bit fractional implementation.

6.176.1 Source File

```
#include <SWLIBS_Config.h>
```

6.176.2 Macro Definition

```
#define SWLIBS_SUPPORT_F32 SWLIBS_STD_ON
```

6.177 SWLIBS_SUPPORT_F16

Enables/disables support of 16-bit fractional implementation.

6.177.1 Source File

```
#include <SWLIBS_Config.h>
```

6.177.2 Macro Definition

```
#define SWLIBS_SUPPORT_F16 SWLIBS_STD_ON
```

6.178 SWLIBS_SUPPORT_FLT

Enables/disables support of single precision floating point implementation.

6.178.1 Source File

```
#include <SWLIBS_Config.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.178.2 Macro Definition

```
#define SWLIBS_SUPPORT_FLT SWLIBS_STD_ON
```

6.179 SWLIBS_SUPPORTED_IMPLEMENTATION

Array of supported implementations.

6.179.1 Source File

```
#include <SWLIBS_Config.h>
```

6.179.2 Macro Definition

```
#define SWLIBS_SUPPORTED_IMPLEMENTATION
{SWLIBS_SUPPORT_F32,SWLIBS_SUPPORT_F16,SWLIBS_SUPPORT_FLT,0,0,0,0,0,0}
```

6.180 SFRACT_MIN

Constant representing the maximal negative value of a signed 16-bit fixed point fractional number, floating point representation.

6.180.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.180.2 Macro Definition

```
#define SFRACT_MIN (-1.0)
```

6.181 SFRACT_MAX

Constant representing the maximal positive value of a signed 16-bit fixed point fractional number, floating point representation.

6.181.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.181.2 Macro Definition

```
#define SFRACT_MAX (0.999969482421875)
```

6.182 FRACT_MIN

Constant representing the maximal negative value of signed 32-bit fixed point fractional number, floating point representation.

6.182.1 Source File

```
#include <SWLIBS_Defines.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.182.2 Macro Definition

```
#define FRACT_MIN (-1.0)
```

6.183 FRACT_MAX

Constant representing the maximal positive value of a signed 32-bit fixed point fractional number, floating point representation.

6.183.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.183.2 Macro Definition

```
#define FRACT_MAX (0.99999999953433871269226074218 75)
```

6.184 FRAC32_0_5

Value 0.5 in 32-bit fixed point fractional format.

6.184.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.184.2 Macro Definition

```
#define FRAC32_0_5 ((tFrac32) 0x40000000U)
```

6.185 FRAC16_0_5

Value 0.5 in 16-bit fixed point fractional format.

6.185.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.185.2 Macro Definition

```
#define FRAC16_0_5 ((tFrac16) 0x4000U)
```

6.186 FRAC32_0_25

Value 0.25 in 32-bit fixed point fractional format.

6.186.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.186.2 Macro Definition

```
#define FRAC32_0_25 ((tFrac32) 0x20000000U)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.187 FRAC16_0_25

Value 0.25 in 16-bit fixed point fractional format.

6.187.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.187.2 Macro Definition

```
#define FRAC16_0_25 ((tFrac16) 0x2000U)
```

6.188 UINT16_MAX

Constant representing the maximal positive value of a unsigned 16-bit fixed point integer number, equal to 2^{15} = 0x8000.

6.188.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.188.2 Macro Definition

```
#define UINT16_MAX ((tU16) 0x8000U)
```

6.189 INT16_MAX

Constant representing the maximal positive value of a signed 16-bit fixed point integer number, equal to 2^{15}-1 = 0x7fff.

6.189.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.189.2 Macro Definition

```
#define INT16_MAX ((tS16) 0x7fffU)
```

6.190 INT16_MIN

Constant representing the maximal negative value of a signed 16-bit fixed point integer number, equal to - 2^{15} = 0x8000.

6.190.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.190.2 Macro Definition

```
#define INT16_MIN ((tS16) 0x8000U)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.191 UINT32_MAX

Constant representing the maximal positive value of a unsigned 32-bit fixed point integer number, equal to 2^{31} = 0x80000000.

6.191.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.191.2 Macro Definition

```
#define UINT32_MAX ((tU32) 0x80000000U)
```

6.192 INT32_MAX

Constant representing the maximal positive value of a signed 32-bit fixed point integer number, equal to 2^{31}-1 = 0x7fff ffff.

6.192.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.192.2 Macro Definition

```
#define INT32_MAX ((tS32) 0x7fffffffU)
```

6.193 INT32_MIN

Constant representing the maximal negative value of a signed 32-bit fixed point integer number, equal to - 2^{31} = 0x8000 0000.

6.193.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.193.2 Macro Definition

```
#define INT32_MIN ((tS32) 0x80000000U)
```

6.194 FLOAT_MIN

Constant representing the maximal negative value of the 32-bit float type.

6.194.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.194.2 Macro Definition

```
#define FLOAT_MIN ((tFloat)(-3.4028234e+38F))
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.195 FLOAT_MAX

Constant representing the maximal positive value of the 32-bit float type.

6.195.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.195.2 Macro Definition

```
#define FLOAT_MAX ((tFloat)(3.4028234e+38F))
```

6.196 INT16TOINT32

Type casting - signed 16-bit integer value cast to a signed 32-bit integer.

6.196.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.196.2 Macro Definition

```
#define INT16TOINT32 ((tS32) (x))
```

6.197 INT32TOINT16

Type casting - signed 32-bit integer value cast to a signed 16-bit integer.

6.197.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.197.2 Macro Definition

```
#define INT32TOINT16 ((tS16) (x))
```

6.198 INT32TOINT64

Type casting - signed 32-bit integer value cast to a signed 64-bit integer.

6.198.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.198.2 Macro Definition

```
#define INT32TOINT64 ((tS64) (x))
```

6.199 INT64TOINT32

Type casting - signed 64-bit integer value cast to a signed 32-bit integer.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.199.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.199.2 Macro Definition

```
#define INT64TOINT32 ((tS32) (x))
```

6.200 F16TOINT16

Type casting - signed 16-bit fractional value cast to a signed 16-bit integer.

6.200.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.200.2 Macro Definition

```
#define F16TOINT16 ((tS16) (x))
```

6.201 F32TOINT16

Type casting - lower 16 bits of a signed 32-bit fractional value cast to a signed 16-bit integer.

6.201.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.201.2 Macro Definition

```
#define F32TOINT16 ((tS16) (x))
```

6.202 F64TOINT16

Type casting - lower 16 bits of a signed 64-bit fractional value cast to a signed 16-bit integer.

6.202.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.202.2 Macro Definition

```
#define F64TOINT16 ((tS16) (x))
```

6.203 F16TOINT32

Type casting - a signed 16-bit fractional value cast to a signed 32-bit integer, the value placed at the lower 16-bits of the 32-bit result.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.203.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.203.2 Macro Definition

```
#define F16TOINT32 ((tS32) (x))
```

6.204 F32TOINT32

Type casting - signed 32-bit fractional value cast to a signed 32-bit integer.

6.204.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.204.2 Macro Definition

```
#define F32TOINT32 ((tS32) (x))
```

6.205 F64TOINT32

Type casting - lower 32 bits of a signed 64-bit fractional value cast to a signed 32-bit integer.

6.205.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.205.2 Macro Definition

```
#define F64TOINT32 ((tS32) (x))
```

6.206 F16TOINT64

Type casting - signed 16-bit fractional value cast to a signed 64-bit integer, the value placed at the lower 16-bits of the 64-bit result.

6.206.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.206.2 Macro Definition

```
#define F16TOINT64 ((tS64) (x))
```

6.207 F32TOINT64

Type casting - signed 32-bit fractional value cast to a signed 64-bit integer, the value placed at the lower 32-bits of the 64-bit result.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.207.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.207.2 Macro Definition

```
#define F32TOINT64 ((tS64) (x))
```

6.208 F64TOINT64

Type casting - signed 64-bit fractional value cast to a signed 64-bit integer.

6.208.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.208.2 Macro Definition

```
#define F64TOINT64 ((tS64) (x))
```

6.209 INT16TOF16

Type casting - signed 16-bit integer value cast to a signed 16-bit fractional.

6.209.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.209.2 Macro Definition

```
#define INT16TOF16 ((tFrac16) (x))
```

6.210 INT16TOF32

Type casting - signed 16-bit integer value cast to a signed 32-bit fractional, the value placed at the lower 16 bits of the 32-bit result.

6.210.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.210.2 Macro Definition

```
#define INT16TOF32 ((tFrac32) (x))
```

6.211 INT32TOF16

Type casting - lower 16-bits of a signed 32-bit integer value cast to a signed 16-bit fractional.

6.211.1 Source File

```
#include <SWLIBS_Defines.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.211.2 Macro Definition

```
#define INT32TOF16 ((tFrac16) (x))
```

6.212 INT32TOF32

Type casting - signed 32-bit integer value cast to a signed 32-bit fractional.

6.212.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.212.2 Macro Definition

```
#define INT32TOF32 ((tFrac32) (x))
```

6.213 INT64TOF16

Type casting - lower 16-bits of a signed 64-bit integer value cast to a signed 16-bit fractional.

6.213.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.213.2 Macro Definition

```
#define INT64TOF16 ((tFrac16) (x))
```

6.214 INT64TOF32

Type casting - lower 32-bits of a signed 64-bit integer value cast to a signed 32-bit fractional.

6.214.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.214.2 Macro Definition

```
#define INT64TOF32 ((tFrac32) (x))
```

6.215 F16_1_DIVBY_SQRT3

One over sqrt(3) with a 16-bit result, the result is rounded for a better accuracy.

6.215.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.215.2 Macro Definition

```
#define F16_1_DIVBY_SQRT3 ((tFrac16) 0x49E7U)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.216 F32_1_DIVBY_SQRT3

One over sqrt(3) with a 32-bit result, the result is rounded for a better accuracy.

6.216.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.216.2 Macro Definition

```
#define F32_1_DIVBY_SQRT3 ((tFrac32) 0x49E69D16U)
```

6.217 F16_SQRT3_DIVBY_2

Sqrt(3) divided by two with a 16-bit result, the result is rounded for a better accuracy.

6.217.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.217.2 Macro Definition

```
#define F16_SQRT3_DIVBY_2 ((tFrac16) 0x6EDAU)
```

6.218 F16_MINUS_SQRT3_DIVBY_2

Minus sqrt(3) divided by two with a 16-bit result, the result is rounded for a better accuracy.

6.218.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.218.2 Macro Definition

```
#define F16_MINUS_SQRT3_DIVBY_2 ((tFrac16) 0x9126U)
```

6.219 F16_SQRT3_DIVBY_4

Sqrt(3) divided by four with a 16-bit result.

6.219.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.219.2 Macro Definition

```
#define F16_SQRT3_DIVBY_4 ((tFrac16) 0x376DU)
```

6.220 F16_SQRT3_DIVBY_6

Sqrt(3) divided by six with a 16-bit result.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.220.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.220.2 Macro Definition

```
#define F16_SQRT3_DIVBY_6 ((tFrac16) 0x24F3U)
```

6.221 F32_SQRT3_DIVBY_2

Sqrt(3) divided by two with a 32-bit result, the result is rounded for a better accuracy.

6.221.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.221.2 Macro Definition

```
#define F32_SQRT3_DIVBY_2 ((tFrac32) 0x6ED9EBA1U)
```

6.222 F32_MINUS_SQRT3_DIVBY_2

Minus sqrt(3) divided by two with a 32-bit result, the result is rounded for a better accuracy.

6.222.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.222.2 Macro Definition

```
#define F32_MINUS_SQRT3_DIVBY_2 ((tFrac32) 0x9126145FU)
```

6.223 F32_SQRT3_DIVBY_4

Sqrt(3) divided by four with a 32-bit result.

6.223.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.223.2 Macro Definition

```
#define F32_SQRT3_DIVBY_4 ((tFrac32) 0x376CF5D1U)
```

6.224 F32_SQRT3_DIVBY_6

Sqrt(3) divided by six with a 32-bit result.

6.224.1 Source File

```
#include <SWLIBS_Defines.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.224.2 Macro Definition

```
#define F32_SQRT3_DIVBY_6 ((tFrac32) 0x24F34E8BU)
```

6.225 F16_SQRT2_DIVBY_2

Sqrt(2) divided by two with a 16-bit result, the result is rounded for a better accuracy.

6.225.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.225.2 Macro Definition

```
#define F16_SQRT2_DIVBY_2 ((tFrac16) 0x5A82U)
```

6.226 F32_SQRT2_DIVBY_2

Sqrt(2) divided by two with a 32-bit result, the result is rounded for a better accuracy.

6.226.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.226.2 Macro Definition

```
#define F32_SQRT2_DIVBY_2 ((tFrac32) 0x5A82799AU)
```

6.227 F16_1_DIVBY_3

One third in 16-bit resolution, the result is rounded for a better accuracy.

6.227.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.227.2 Macro Definition

```
#define F16_1_DIVBY_3 ((tFrac16) 0x2AABU)
```

6.228 F32_1_DIVBY_3

One third in 32-bit resolution, the result is rounded for a better accuracy.

6.228.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.228.2 Macro Definition

```
#define F32_1_DIVBY_3 ((tFrac32) 0x2AAAAAABU)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.229 F16_MINUS_1_DIVBY_3

One third in 16-bit resolution, the result is rounded for a better accuracy.

6.229.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.229.2 Macro Definition

```
#define F16_MINUS_1_DIVBY_3 ((tFrac16) 0xD556U)
```

6.230 F32_MINUS_1_DIVBY_3

One third in 32-bit resolution, the result is rounded for a better accuracy.

6.230.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.230.2 Macro Definition

```
#define F32_MINUS_1_DIVBY_3 ((tFrac32) 0xD5555556U)
```

6.231 F16_MINUS_1_DIVBY_2

One half in 16-bit resolution, the result is rounded for a better accuracy.

6.231.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.231.2 Macro Definition

```
#define F16_MINUS_1_DIVBY_2 ((tFrac16) 0xC000U)
```

6.232 F32_MINUS_1_DIVBY_2

One half in 32-bit resolution, the result is rounded for a better accuracy.

6.232.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.232.2 Macro Definition

```
#define F32_MINUS_1_DIVBY_2 ((tFrac32) 0xC0000000U)
```

6.233 F16_2_DIVBY_3

Two thirds in 16-bit resolution, the result is rounded for a better accuracy.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.233.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.233.2 Macro Definition

```
#define F16_2_DIVBY_3 ((tFrac16) 0x5555U)
```

6.234 F32_2_DIVBY_3

Two thirds in 32-bit resolution, the result is rounded for a better accuracy.

6.234.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.234.2 Macro Definition

```
#define F32_2_DIVBY_3 ((tFrac32) 0x55555555U)
```

6.235 FRAC16

Macro converting a signed fractional [-1,1) number into a 16-bit fixed point number in format Q1.15.

6.235.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.235.2 Macro Definition

```
#define FRAC16 ((tFrac16) (((x) < SFRACT_MAX) ? (((x) >=
SFRACT_MIN) ? ((x)*32768.0) : INT16_MIN) : INT16_MAX))
```

6.236 FRAC32

Macro converting a signed fractional [-1,1) number into a 32-bit fixed point number in format Q1.31.

6.236.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.236.2 Macro Definition

```
#define FRAC32 ((tFrac32) (((x) < FRACT_MAX) ? (((x) >=
FRACT_MIN) ? ((x)*2147483648.0) : INT32_MIN) : INT32_MAX))
```

6.237 FLOAT_DIVBY_SQRT3

One over sqrt(3) in single precision floating point format.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.237.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.237.2 Macro Definition

```
#define FLOAT_DIVBY_SQRT3 ((tFloat) 0.5773502691896258)
```

6.238 FLOAT_SQRT3

Sqrt(3) in single precision floating point format.

6.238.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.238.2 Macro Definition

```
#define FLOAT_SQRT3 ((tFloat) 1.732050776481628)
```

6.239 FLOAT_SQRT3_DIVBY_2

Sqrt(3) divided by two in single precision floating point format.

6.239.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.239.2 Macro Definition

```
#define FLOAT_SQRT3_DIVBY_2 ((tFloat) 0.866025403784439)
```

6.240 FLOAT_MINUS_SQRT3_DIVBY_2

Sqrt(3) divided by two in single precision floating point format.

6.240.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.240.2 Macro Definition

```
#define FLOAT_MINUS_SQRT3_DIVBY_2 ((tFloat) -0.866025403784439)
```

6.241 FLOAT_SQRT3_DIVBY_4

Sqrt(3) divided by four in single precision floating point format.

6.241.1 Source File

```
#include <SWLIBS_Defines.h>
```


© NXP B.V. 2023. All rights reserved.

6.241.2 Macro Definition

```
#define FLOAT_SQRT3_DIVBY_4 ((tFloat) 0.4330127018922190)
```

6.242 FLOAT_SQRT3_DIVBY_6

Sqrt(3) divided by six in single precision floating point format.

6.242.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.242.2 Macro Definition

```
#define FLOAT_SQRT3_DIVBY_6 ((tFloat) 0.2886751)
```

6.243 FLOAT_SQRT3_DIVBY_4_CORRECTION

Sqrt(3) divided by four correction constant.

6.243.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.243.2 Macro Definition

```
#define FLOAT_SQRT3_DIVBY_4_CORRECTION ((tFloat)0)
```

6.244 FLOAT_2_PI

2* π in single precision floating point format.

6.244.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.244.2 Macro Definition

```
#define FLOAT_2_PI ((tFloat) 6.28318530717958)
```

6.245 FLOAT_2_PI_DIVBY_3

2* π divided by three in single precision floating point format.

6.245.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.245.2 Macro Definition

```
#define FLOAT_2_PI_DIVBY_3 ((tFloat) 2.094395160675049)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.246 FLOAT_PI

π in single precision floating point format.

6.246.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.246.2 Macro Definition

```
#define FLOAT_PI ((tFloat) 3.14159265358979)
```

6.247 FLOAT_PI_DIVBY_2

π/2 in single precision floating point format.

6.247.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.247.2 Macro Definition

```
#define FLOAT_PI_DIVBY_2 ((tFloat) 1.57079632679490)
```

6.248 FLOAT_TAN_PI_DIVBY_6

Tan(π/6) in single precision floating point format.

6.248.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.248.2 Macro Definition

```
#define FLOAT_TAN_PI_DIVBY_6 ((tFloat)0.577350269189626000)
```

6.249 FLOAT_TAN_PI_DIVBY_12

Tan(π/12) in single precision floating point format.

6.249.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.249.2 Macro Definition

```
#define FLOAT_TAN_PI_DIVBY_12 ((tFloat)0.267949192431123000)
```

6.250 FLOAT_PI_DIVBY_6

π/6 in single precision floating point format.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.250.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.250.2 Macro Definition

```
#define FLOAT_PI_DIVBY_6 ((tFloat)0.523598775598299000)
```

6.251 FLOAT_PI_SINGLE_CORRECTION

Double to single precision correction constant for π, equal to (π(Double) - π(Single)).

6.251.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.251.2 Macro Definition

```
#define FLOAT_PI_SINGLE_CORRECTION
((tFloat)4.371139006309477e-08)
```

6.252 FLOAT_PI_CORRECTION

Double to single precision correction constant for π, equal to (2*(π(Double) - π(Single)).

6.252.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.252.2 Macro Definition

```
#define FLOAT_PI_CORRECTION ((tFloat)8.742278012618954e-08)
```

6.253 FLOAT_PI_DIVBY_4

π/4 in single precision floating point format.

6.253.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.253.2 Macro Definition

```
#define FLOAT_PI_DIVBY_4 ((tFloat) 0.7853981633974480)
```

6.254 FLOAT_4_DIVBY_PI

Number four divided by π in single precision floating point format.

6.254.1 Source File

```
#include <SWLIBS_Defines.h>
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.254.2 Macro Definition

```
#define FLOAT_4_DIVBY_PI ((tFloat) 1.2732395447351600)
```

6.255 FLOAT_0_5

Value 0.5 in single precision floating point format.

6.255.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.255.2 Macro Definition

```
#define FLOAT_0_5 ((tFloat) 0.5)
```

6.256 FLOAT_MINUS_0_5

Value -0.5 in single precision floating point format.

6.256.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.256.2 Macro Definition

```
#define FLOAT_MINUS_0_5 ((tFloat) -0.5)
```

6.257 FLOAT_PLUS_1

Value 1 in single precision floating point format.

6.257.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.257.2 Macro Definition

```
#define FLOAT_PLUS_1 ((tFloat) 1)
```

6.258 FLOAT_MINUS_1

Value -1 in single precision floating point format.

6.258.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.258.2 Macro Definition

```
#define FLOAT_MINUS_1 ((tFloat) -1)
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.259 FLOAT_MIN_NORM

Constant representing the smallest positive normalized 32-bit floating-point value.

6.259.1 Source File

```
#include <SWLIBS_Defines.h>
```

6.259.2 Macro Definition

```
#define FLOAT_MIN_NORM ((tFloat) 1.175494350822288e-38)
```

6.260 SWLIBS_2Syst

Definition of SWLIBS_2Syst as alias for SWLIBS_2Syst_F16 array in case the 16-bit fractional implementation is selected.

6.260.1 Source File

```
#include <SWLIBS_Typedefs.h>
```

6.260.2 Macro Definition

```
#define SWLIBS_2Syst SWLIBS_2Syst_F16
```

6.261 SWLIBS_3Syst

Definition of SWLIBS_3Syst as alias for SWLIBS_3Syst_F16 array in case the 16-bit fractional implementation is selected.

6.261.1 Source File

```
#include <SWLIBS_Typedefs.h>
```

6.261.2 Macro Definition

```
#define SWLIBS_3Syst SWLIBS_3Syst_F16
```

6.262 SWLIBS_6Syst

Definition of SWLIBS_6Syst as alias for SWLIBS_3Syst_F16 array in case the 16-bit fractional implementation is selected.

6.262.1 Source File

```
#include <SWLIBS_Typedefs.h>
```

6.262.2 Macro Definition

```
#define SWLIBS_6Syst SWLIBS_6Syst_F16
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.263 SWLIBS_ID

Library identification string.

6.263.1 Source File

```
#include <SWLIBS_Version.h>
```

6.263.2 Macro Definition

```
#define SWLIBS_ID {(unsigned char)0x90U,(unsigned char)0x71U,
(unsigned char)0x77U,(unsigned char)0x68U}
```


All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

7 Legal information

7.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

7.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors. In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory. Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification. Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products. NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer. In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages. Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

7.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

Contents

1 Introduction ... 3
1.1 Architecture Overview ... 3
1.2 General Information ... 4
1.3 Installation ... 4
1.4 Library File Structure ... 7
1.5 Multiple Implementation Support ... 8
1.5.1 API Postfix Option ... 9
1.5.2 Additional Parameter Option ... 9
1.5.3 Global Configuration Option ... 9
1.6 Supported Compilers ... 9
1.7 Integration Assumption ... 10
1.8 Library Integration into a Green Hills Multi Development Environment ... 10
1.9 Library Integration into a IAR Project ... 12
1.10 Library Integration into a S32 Design Studio for S32 Platform as a RTD plugin ... 15
1.11 Library Integration into a S32 Design Studio for S32 Platform as a standalone package ... 17
1.12 MATLAB Integration ... 17
1.13 Library Testing ... 20
1.13.1 AMMCLIB Testing based on the Matlab Simulink Toolbox ... 21
1.13.2 AMMCLIB target-in-loop Testing based on the SFIO Toolbox ... 22
1.14 Functions Accuracy ... 23
2 Functions ... 25
2.1 Function index ... 25
2.2 Function AMCLIB_BemfObsrvDQ ... 51
2.2.1 Description ... 51
2.2.2 Re-entrancy ... 53
2.2.3 Function AMCLIB_BemfObsrvDQ_F32 ... 54
2.2.3.1 Declaration ... 54
2.2.3.2 Arguments ... 54
2.2.3.3 Return ... 54
2.2.3.4 Implementation details ... 54
2.2.3.5 Code Example ... 59
2.2.4 Function AMCLIB_BemfObsrvDQ_F16 ... 60
2.2.4.1 Declaration ... 60
2.2.4.2 Arguments ... 61
2.2.4.3 Return ... 61
2.2.4.4 Implementation details ... 61
2.2.4.5 Code Example ... 66
2.2.5 Function AMCLIB_BemfObsrvDQ_FLT ... 67
2.2.5.1 Declaration ... 67
2.2.5.2 Arguments ... 67
2.2.5.3 Return ... 68
2.2.5.4 Implementation details ... 68
2.2.5.5 Code Example ... 70
2.2.6 Function AMCLIB_BemfObsrvDQInit ... 72
2.2.6.1 Description ... 72
2.2.6.2 Re-entrancy ... 72
2.2.6.3 Function AMCLIB_BemfObsrvDQInit_F32 ... 72
2.2.6.4 Function AMCLIB_BemfObsrvDQInit_F16 ... 74
2.2.6.5 Function AMCLIB_BemfObsrvDQInit_FLT ... 76
2.2.7 Function AMCLIB_BemfObsrvDQSetState ... 78
2.2.7.1 Description ... 78
2.2.7.2 Re-entrancy ... 78
2.2.7.3 Function AMCLIB_BemfObsrvDQSetState_F32 ... 79
2.2.7.4 Function AMCLIB_BemfObsrvDQSetState_F16 ... 81
2.2.7.5 Function AMCLIB_BemfObsrvDQSetState_FLT ... 83
2.3 Function AMCLIB_CurrentLoop ... 85
2.3.1 Description ... 85
2.3.2 Re-entrancy ... 86
2.3.3 Function AMCLIB_CurrentLoop_F32 ... 86
2.3.3.1 Declaration ... 86
2.3.3.2 Arguments ... 86
2.3.3.3 Implementation details ... 87
2.3.3.4 Code Example ... 88
2.3.4 Function AMCLIB_CurrentLoop_F16 ... 90
2.3.4.1 Declaration ... 90
2.3.4.2 Arguments ... 90
2.3.4.3 Implementation details ... 90
2.3.4.4 Code Example ... 92
2.3.5 Function AMCLIB_CurrentLoop_FLT ... 94
2.3.5.1 Declaration ... 94
2.3.5.2 Arguments ... 94
2.3.5.3 Implementation details ... 94
2.3.5.4 Code Example ... 96
2.3.6 Function AMCLIB_CurrentLoopInit ... 97
2.3.6.1 Description ... 97
2.3.6.2 Re-entrancy ... 97
2.3.6.3 Function AMCLIB_CurrentLoopInit_F32 ... 97
2.3.6.4 Function AMCLIB_CurrentLoopInit_F16 ... 99
2.3.6.5 Function AMCLIB_CurrentLoopInit_FLT ... 101
2.3.7 Function AMCLIB_CurrentLoopSetState ... 103
2.3.7.1 Description ... 103
2.3.7.2 Re-entrancy ... 104
2.3.7.3 Function AMCLIB_CurrentLoopSetState_F32 ... 104
2.3.7.4 Function AMCLIB_CurrentLoopSetState_F16 ... 106
2.3.7.5 Function AMCLIB_CurrentLoopSetState_FLT ... 108
2.4 Function AMCLIB_CurrentLoopPIp ... 110
2.4.1 Description ... 110
2.4.2 Re-entrancy ... 111
2.4.3 Function AMCLIB_CurrentLoopPIp_F32 ... 111
2.4.3.1 Declaration ... 111
2.4.3.2 Arguments ... 111
2.4.3.3 Implementation details ... 111
2.4.3.4 Code Example ... 113
2.4.4 Function AMCLIB_CurrentLoopPIp_F16 ... 114
2.4.4.1 Declaration ... 114
2.4.4.2 Arguments ... 114
2.4.4.3 Return ... 114
2.4.4.4 Implementation details ... 114
2.4.4.5 Code Example ... 116
2.4.5 Function AMCLIB_CurrentLoopPIp_FLT ... 117
2.4.5.1 Declaration ... 117

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.4.5.2 Arguments 117
2.4.5.3 Implementation details 117
2.4.5.4 Code Example 118
2.4.6 Function AMCLIB_CurrentLoopPIpInit 119
2.4.6.1 Description 119
2.4.6.2 Re-entrancy 119
2.4.6.3 Function AMCLIB_CurrentLoopPIpInit_F32 119
2.4.6.4 Function AMCLIB_CurrentLoopPIpInit_F16 120
2.4.6.5 Function AMCLIB_CurrentLoopPIpInit_FLT 120
2.4.7 Function AMCLIB_CurrentLoopPIpSetState 121
2.4.7.1 Description 121
2.4.7.2 Re-entrancy 121
2.4.7.3 Function AMCLIB_CurrentLoopPIpSetState_F32 121
2.4.7.4 Function AMCLIB_CurrentLoopPIpSetState_F16 122
2.4.7.5 Function AMCLIB_CurrentLoopPIpSetState_FLT 123
2.5 Function AMCLIB_EstimBJ_FLT 124
2.5.1 Declaration 124
2.5.2 Arguments 124
2.5.3 Return 124
2.5.4 Description 124
2.5.5 Code Example 125
2.5.6 Reentrancy 127
2.5.7 Function AMCLIB_EstimBJInit_FLT 127
2.5.7.1 Declaration 127
2.5.7.2 Arguments 127
2.5.7.3 Return 127
2.5.7.4 Implementation details 127
2.5.7.5 Code Example 127
2.5.7.6 Reentrancy 128
2.6 Function AMCLIB_EstimRL 128
2.6.1 Description 129
2.6.2 Re-entrancy 130
2.6.3 Function AMCLIB_EstimRL_FLT 130
2.6.3.1 Declaration 130
2.6.3.2 Arguments 131
2.6.3.3 Return 131
2.6.3.4 Implementation details 131
2.6.3.5 Code Example 131
2.6.4 Function AMCLIB_EstimRLInit 132
2.6.4.1 Description 133
2.6.4.2 Re-entrancy 133
2.6.4.3 Function AMCLIB_EstimRLInit_FLT 133
2.7 Function AMCLIB_FluxEstimAB 135
2.7.1 Description 135
2.7.2 Re-entrancy 136
2.7.3 Function AMCLIB_FluxEstimAB_F32 136
2.7.3.1 Declaration 136
2.7.3.2 Arguments 136
2.7.3.3 Return 136
2.7.3.4 Implementation details 137
2.7.3.5 Code Example 137
2.7.4 Function AMCLIB_FluxEstimAB_F16 138
2.7.4.1 Declaration 138
2.7.4.2 Arguments 138
2.7.4.3 Return 138
2.7.4.4 Implementation details 138
2.7.4.5 Code Example 139
2.7.5 Function AMCLIB_FluxEstimAB_FLT 140
2.7.5.1 Declaration 140
2.7.5.2 Arguments 140
2.7.5.3 Return 140
2.7.5.4 Code Example 140
2.8 Function AMCLIB_FW 141
2.8.1 Description 141
2.8.2 Re-entrancy 143
2.8.3 Function AMCLIB_FW_F32 143
2.8.3.1 Declaration 143
2.8.3.2 Arguments 144
2.8.3.3 Implementation details 144
2.8.3.4 Code Example 145
2.8.4 Function AMCLIB_FW_F16 147
2.8.4.1 Declaration 147
2.8.4.2 Arguments 147
2.8.4.3 Implementation details 147
2.8.4.4 Code Example 148
2.8.5 Function AMCLIB_FW_FLT 150
2.8.5.1 Declaration 150
2.8.5.2 Arguments 150
2.8.5.3 Implementation details 150
2.8.5.4 Code Example 151
2.8.6 Function AMCLIB_FWInit 153
2.8.6.1 Description 153
2.8.6.2 Re-entrancy 153
2.8.6.3 Function AMCLIB_FWInit_F32 153
2.8.6.4 Function AMCLIB_FWInit_F16 155
2.8.6.5 Function AMCLIB_FWInit_FLT 157
2.8.7 Function AMCLIB_FWSetState 159
2.8.7.1 Description 159
2.8.7.2 Re-entrancy 159
2.8.7.3 Function AMCLIB_FWSetState_F32 159
2.8.7.4 Function AMCLIB_FWSetState_F16 162
2.8.7.5 Function AMCLIB_FWSetState_FLT 164
2.8.8 Function AMCLIB_FWDebug 166
2.8.8.1 Description 166
2.8.8.2 Re-entrancy 166
2.8.8.3 Function AMCLIB_FWDebug_F32 166
2.8.8.4 Function AMCLIB_FWDebug_F16 169
2.8.8.5 Function AMCLIB_FWDebug_FLT 171
2.9 Function AMCLIB_FWSpeedLoop 174
2.9.1 Description 174
2.9.2 Re-entrancy 175
2.9.3 Function AMCLIB_FWSpeedLoop_F32 175
2.9.3.1 Declaration 175
2.9.3.2 Arguments 175
2.9.3.3 Implementation details 176
2.9.3.4 Code Example 178
2.9.4 Function AMCLIB_FWSpeedLoop_F16 180
2.9.4.1 Declaration 180
2.9.4.2 Arguments 180
2.9.4.3 Implementation details 180
2.9.4.4 Code Example 182
2.9.5 Function AMCLIB_FWSpeedLoop_FLT 184
2.9.5.1 Declaration 184
2.9.5.2 Arguments 185
2.9.5.3 Implementation details 185

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.9.5.4 Code Example 186
2.9.6 Function AMCLIB_FWSpeedLoopInit 188
2.9.6.1 Description 188
2.9.6.2 Re-entrancy 188
2.9.6.3 Function AMCLIB_FWSpeedLoopInit_F32 188
2.9.6.4 Function AMCLIB_FWSpeedLoopInit_F16 190
2.9.6.5 Function AMCLIB_FWSpeedLoopInit_FLT 193
2.9.7 Function AMCLIB_FWSpeedLoopSetState ... 195
2.9.7.1 Description 195
2.9.7.2 Re-entrancy 195
2.9.7.3 Function AMCLIB_ FWSpeedLoopSetState_F32 195
2.9.7.4 Function AMCLIB_ FWSpeedLoopSetState_F16 198
2.9.7.5 Function AMCLIB_ FWSpeedLoopSetState_FLT 200
2.9.8 Function AMCLIB_FWSpeedLoopDebug 203
2.9.8.1 Description 203
2.9.8.2 Re-entrancy 203
2.9.8.3 Function AMCLIB_FWSpeedLoopDebug_ F32 204
2.9.8.4 Function AMCLIB_FWSpeedLoopDebug_ F16 206
2.9.8.5 Function AMCLIB_FWSpeedLoopDebug_ FLT 209
2.10 Function AMCLIB_SpeedLoop 212
2.10.1 Description 212
2.10.2 Re-entrancy 213
2.10.3 Function AMCLIB_SpeedLoop_F32 213
2.10.3.1 Declaration 213
2.10.3.2 Arguments 213
2.10.3.3 Implementation details 213
2.10.3.4 Code Example 215
2.10.4 Function AMCLIB_SpeedLoop_F16 216
2.10.4.1 Declaration 216
2.10.4.2 Arguments 216
2.10.4.3 Implementation details 217
2.10.4.4 Code Example 218
2.10.5 Function AMCLIB_SpeedLoop_FLT 220
2.10.5.1 Declaration 220
2.10.5.2 Arguments 220
2.10.5.3 Implementation details 220
2.10.5.4 Code Example 221
2.10.6 Function AMCLIB_SpeedLoopInit 222
2.10.6.1 Description 222
2.10.6.2 Re-entrancy 222
2.10.6.3 Function AMCLIB_SpeedLoopInit_F32 222
2.10.6.4 Function AMCLIB_SpeedLoopInit_F16 224
2.10.6.5 Function AMCLIB_SpeedLoopInit_FLT 226
2.10.7 Function AMCLIB_SpeedLoopSetState 228
2.10.7.1 Description 228
2.10.7.2 Re-entrancy 228
2.10.7.3 Function AMCLIB_SpeedLoopSetState_ F32 228
2.10.7.4 Function AMCLIB_SpeedLoopSetState_ F16 230
2.10.7.5 Function AMCLIB_SpeedLoopSetState_ FLT 232
2.10.8 Function AMCLIB_SpeedLoopDebug 234
2.10.8.1 Description 234
2.10.8.2 Re-entrancy 234
2.10.8.3 Function AMCLIB_SpeedLoopDebug_F32 235
2.10.8.4 Function AMCLIB_SpeedLoopDebug_F16 237
2.10.8.5 Function AMCLIB_SpeedLoopDebug_FLT 239
2.11 Function AMCLIB_TrackObsrv 241
2.11.1 Description 241
2.11.2 Re-entrancy 244
2.11.3 Function AMCLIB_TrackObsrv_F32 244
2.11.3.1 Declaration 244
2.11.3.2 Arguments 244
2.11.3.3 Implementation details 244
2.11.3.4 Code example 246
2.11.4 Function AMCLIB_TrackObsrv_F16 247
2.11.4.1 Declaration 247
2.11.4.2 Arguments 247
2.11.4.3 Implementation details 247
2.11.4.4 Code example 249
2.11.5 Function AMCLIB_TrackObsrv_FLT 250
2.11.5.1 Declaration 250
2.11.5.2 Arguments 250
2.11.5.3 Implementation details 250
2.11.5.4 Code example: 251
2.11.6 Function AMCLIB_TrackObsrvInit 252
2.11.6.1 Description 252
2.11.6.2 Re-entrancy: 253
2.11.6.3 Function AMCLIB_TrackObsrvInit_F32 253
2.11.6.4 Function AMCLIB_TrackObsrvInit_F16 254
2.11.6.5 Function AMCLIB_TrackObsrvInit_FLT 256
2.11.7 Function AMCLIB_TrackObsrvSetState 258
2.11.7.1 Description 258
2.11.7.2 Re-entrancy: 258
2.11.7.3 Function AMCLIB_TrackObsrvSetState_ F32 258
2.11.7.4 Function AMCLIB_TrackObsrvSetState_ F16 260
2.11.7.5 Function AMCLIB_TrackObsrvSetState_ FLT 262
2.12 Function AMCLIB_Windmilling 264
2.12.1 Description 264
2.12.2 Re-entrancy 265
2.12.3 Function AMCLIB_Windmilling_F32 265
2.12.3.1 Declaration 265
2.12.3.2 Arguments 265
2.12.3.3 Return 266
2.12.3.4 Implementation details 266
2.12.3.5 Code Example 266
2.12.4 Function AMCLIB_Windmilling_F16 267
2.12.4.1 Declaration 267
2.12.4.2 Arguments 268
2.12.4.3 Return 268
2.12.4.4 Implementation details 268
2.12.4.5 Code Example 268
2.12.5 Function AMCLIB_Windmilling_FLT 270
2.12.5.1 Declaration 270
2.12.5.2 Arguments 270
2.12.5.3 Return 270
2.12.5.4 Implementation details 270
2.12.5.5 Code Example 270

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.12.6 Function AMCLIB_WindmillingInit 272
2.12.6.1 Description 272
2.12.6.2 Re-entrancy: 272
2.12.6.3 Function AMCLIB_WindmillingInit_F32 272
2.12.6.4 Function AMCLIB_WindmillingInit_F16 274
2.12.6.5 Function AMCLIB_WindmillingInit_FLT 275
2.13 Function GDFLIB_FilterFIR 277
2.13.1 Description 277
2.13.2 Re-entrancy 278
2.13.3 Function GDFLIB_FilterFIR_F32 278
2.13.3.1 Declaration 278
2.13.3.2 Arguments 278
2.13.3.3 Return 278
2.13.3.4 Implementation details 278
2.13.3.5 Code Example 278
2.13.4 Function GDFLIB_FilterFIR_F16 280
2.13.4.1 Declaration 280
2.13.4.2 Arguments 280
2.13.4.3 Return 280
2.13.4.4 Implementation details 280
2.13.4.5 Code Example 280
2.13.5 Function GDFLIB_FilterFIR_FLT 282
2.13.5.1 Declaration 282
2.13.5.2 Arguments 282
2.13.5.3 Return 282
2.13.5.4 Code Example 282
2.13.6 Function GDFLIB_FilterFIRInit 284
2.13.6.1 Description 284
2.13.6.2 Re-entrancy 285
2.13.6.3 Function GDFLIB_FilterFIRInit_F32 285
2.13.6.4 Function GDFLIB_FilterFIRInit_F16 287
2.13.6.5 Function GDFLIB_FilterFIRInit_FLT 289
2.14 Function GDFLIB_FilterIIR1 291
2.14.1 Description 291
2.14.2 Re-entrancy 292
2.14.3 Function GDFLIB_FilterIIR1_F32 292
2.14.3.1 Declaration 292
2.14.3.2 Arguments 293
2.14.3.3 Return 293
2.14.3.4 Implementation details 293
2.14.3.5 Code Example 293
2.14.4 Function GDFLIB_FilterIIR1_F16 294
2.14.4.1 Declaration 294
2.14.4.2 Arguments 294
2.14.4.3 Return 294
2.14.4.4 Implementation details 294
2.14.4.5 Code Example 295
2.14.5 Function GDFLIB_FilterIIR1_FLT 296
2.14.5.1 Declaration 296
2.14.5.2 Arguments 296
2.14.5.3 Return 296
2.14.5.4 Implementation details 296
2.14.5.5 Code Example 296
2.14.6 Function GDFLIB_FilterIIR1Init 297
2.14.6.1 Description 297
2.14.6.2 Re-entrancy 297
2.14.6.3 Function GDFLIB_FilterIIR1Init_F32 297
2.14.6.4 Function GDFLIB_FilterIIR1Init_F16 298
2.14.6.5 Function GDFLIB_FilterIIR1Init_FLT 299
2.15 Function GDFLIB_FilterIIR2 299
2.15.1 Description 300
2.15.2 Re-entrancy 301
2.15.3 Function GDFLIB_FilterIIR2_F32 301
2.15.3.1 Declaration 301
2.15.3.2 Arguments 301
2.15.3.3 Return 301
2.15.3.4 Implementation details 302
2.15.3.5 Code Example 302
2.15.4 Function GDFLIB_FilterIIR2_F16 303
2.15.4.1 Declaration 303
2.15.4.2 Arguments 303
2.15.4.3 Return 303
2.15.4.4 Implementation details 303
2.15.4.5 Code Example 304
2.15.5 Function GDFLIB_FilterIIR2_FLT 305
2.15.5.1 Declaration 305
2.15.5.2 Arguments 305
2.15.5.3 Return 305
2.15.5.4 Implementation details 305
2.15.5.5 Code Example 306
2.15.6 Function GDFLIB_FilterIIR2Init 306
2.15.6.1 Description 306
2.15.6.2 Re-entrancy 306
2.15.6.3 Function GDFLIB_FilterIIR2Init_F32 307
2.15.6.4 Function GDFLIB_FilterIIR2Init_F16 307
2.15.6.5 Function GDFLIB_FilterIIR2Init_FLT 308
2.16 Function GDFLIB_FilterMA 309
2.16.1 Description 309
2.16.2 Re-entrancy 309
2.16.3 Function GDFLIB_FilterMA_F32 309
2.16.3.1 Declaration 309
2.16.3.2 Arguments 310
2.16.3.3 Return 310
2.16.3.4 Implementation details 310
2.16.3.5 Code Example 310
2.16.4 Function GDFLIB_FilterMA_F16 311
2.16.4.1 Declaration 311
2.16.4.2 Arguments 311
2.16.4.3 Return 311
2.16.4.4 Implementation details 311
2.16.4.5 Code Example 311
2.16.5 Function GDFLIB_FilterMA_FLT 312
2.16.5.1 Declaration 312
2.16.5.2 Arguments 312
2.16.5.3 Return 312
2.16.5.4 Implementation details 312
2.16.5.5 Code Example 313
2.16.6 Function GDFLIB_FilterMAInit 314
2.16.6.1 Description 314
2.16.6.2 Re-entrancy 314
2.16.6.3 Function GDFLIB_FilterMAInit_F32 314
2.16.6.4 Function GDFLIB_FilterMAInit_F16 315
2.16.6.5 Function GDFLIB_FilterMAInit_FLT 316
2.16.7 Function GDFLIB_FilterMASetState 316
2.16.7.1 Description 316
2.16.7.2 Re-entrancy 316
2.16.7.3 Function GDFLIB_FilterMASetState_F32 317
2.16.7.4 Function GDFLIB_FilterMASetState_F16 317

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.16.7.5 Function GDFLIB_FilterMASetState_FLT 318
2.17 Function GFLIB_Acos 319
2.17.1 Description 319
2.17.2 Re-entrancy 319
2.17.3 Function GFLIB_Acos_F32 320
2.17.3.1 Declaration 320
2.17.3.2 Arguments 320
2.17.3.3 Return 320
2.17.3.4 Implementation details 320
2.17.3.5 Code Example 321
2.17.4 Function GFLIB_Acos_F16 322
2.17.4.1 Declaration 322
2.17.4.2 Arguments 322
2.17.4.3 Return 322
2.17.4.4 Implementation details 322
2.17.4.5 Code Example 323
2.17.5 Function GFLIB_Acos_FLT 324
2.17.5.1 Declaration 324
2.17.5.2 Arguments 324
2.17.5.3 Return 324
2.17.5.4 Implementation details 324
2.17.5.5 Code Example 325
2.18 Function GFLIB_Asin 325
2.18.1 Description 325
2.18.2 Re-entrancy 326
2.18.3 Function GFLIB_Asin_F32 326
2.18.3.1 Declaration 326
2.18.3.2 Arguments 326
2.18.3.3 Return 326
2.18.3.4 Implementation details 326
2.18.3.5 Code Example 328
2.18.4 Function GFLIB_Asin_F16 328
2.18.4.1 Declaration 328
2.18.4.2 Arguments 328
2.18.4.3 Return 328
2.18.4.4 Implementation details 329
2.18.4.5 Code Example 330
2.18.5 Function GFLIB_Asin_FLT 330
2.18.5.1 Declaration 330
2.18.5.2 Arguments 331
2.18.5.3 Return 331
2.18.5.4 Implementation details 331
2.18.5.5 Code Example 332
2.19 Function GFLIB_Atan 332
2.19.1 Description 332
2.19.2 Re-entrancy 333
2.19.3 Function GFLIB_Atan_F32 333
2.19.3.1 Declaration 333
2.19.3.2 Arguments 333
2.19.3.3 Return 333
2.19.3.4 Implementation details 333
2.19.3.5 Code Example 334
2.19.4 Function GFLIB_Atan_F16 335
2.19.4.1 Declaration 335
2.19.4.2 Arguments 335
2.19.4.3 Return 335
2.19.4.4 Implementation details 335
2.19.4.5 Code Example 336
2.19.5 Function GFLIB_Atan_FLT 336
2.19.5.1 Declaration 336
2.19.5.2 Arguments 336
2.19.5.3 Return 337
2.19.5.4 Implementation details 337
2.19.5.5 Code Example 338
2.20 Function GFLIB_AtanYX 338
2.20.1 Description 338
2.20.2 Re-entrancy 338
2.20.3 Function GFLIB_AtanYX_F32 338
2.20.3.1 Declaration 338
2.20.3.2 Arguments 339
2.20.3.3 Return 339
2.20.3.4 Implementation details 339
2.20.3.5 Code Example 339
2.20.4 Function GFLIB_AtanYX_F16 340
2.20.4.1 Declaration 340
2.20.4.2 Arguments 340
2.20.4.3 Return 340
2.20.4.4 Implementation details 340
2.20.4.5 Code Example 341
2.20.5 Function GFLIB_AtanYX_FLT 341
2.20.5.1 Declaration 341
2.20.5.2 Arguments 341
2.20.5.3 Return 341
2.20.5.4 Implementation details 342
2.20.5.5 Code Example 342
2.21 Function GFLIB_AtanYXShifted 342
2.21.1 Description 343
2.21.2 Re-entrancy 343
2.21.3 Function GFLIB_AtanYXShifted_F32 343
2.21.3.1 Declaration 343
2.21.3.2 Arguments 343
2.21.3.3 Return 343
2.21.3.4 Implementation details 343
2.21.3.5 Code Example 346
2.21.4 Function GFLIB_AtanYXShifted_F16 347
2.21.4.1 Declaration 347
2.21.4.2 Arguments 347
2.21.4.3 Return 348
2.21.4.4 Implementation details 348
2.21.4.5 Code Example 350
2.21.5 Function GFLIB_AtanYXShifted_FLT 351
2.21.5.1 Declaration 351
2.21.5.2 Arguments 351
2.21.5.3 Return 352
2.21.5.4 Implementation details 352
2.21.5.5 Code Example 354
2.22 Function GFLIB_ControllerPIDpAW 355
2.22.1 Description 355
2.22.2 Re-entrancy 356
2.22.3 Function GFLIB_ControllerPIDpAW_F32 357
2.22.3.1 Declaration 357
2.22.3.2 Arguments 357
2.22.3.3 Return 357
2.22.3.4 Implementation details 357
2.22.3.5 Code Example 359
2.22.4 Function GFLIB_ControllerPIDpAW_F16 360
2.22.4.1 Declaration 360
2.22.4.2 Arguments 360

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.22.4.3 Return ... 361
2.22.4.4 Implementation details ... 361
2.22.4.5 Code Example ... 363
2.22.5 Function GFLIB_ControllerPIDpAW_FLT ... 364
2.22.5.1 Declaration ... 364
2.22.5.2 Arguments ... 364
2.22.5.3 Return ... 364
2.22.5.4 Implementation details ... 365
2.22.5.5 Code Example ... 365
2.22.6 Function GFLIB_ControllerPIDpAWInit ... 367
2.22.6.1 Description ... 367
2.22.6.2 Re-entrancy ... 367
2.22.6.3 Function GFLIB_ControllerPIDpAWInit_F32 ... 367
2.22.6.4 Function GFLIB_ControllerPIDpAWInit_F16 ... 369
2.22.6.5 Function GFLIB_ControllerPIDpAWInit_FLT ... 370
2.22.7 Function GFLIB_ ControllerPIDpAWSetState ... 372
2.22.7.1 Description ... 372
2.22.7.2 Re-entrancy ... 372
2.22.7.3 Function GFLIB_ ControllerPIDpAWSetState_F32 ... 372
2.22.7.4 Function GFLIB_ ControllerPIDpAWSetState_F16 ... 374
2.22.7.5 Function GFLIB_ ControllerPIDpAWSetState_FLT ... 375
2.23 Function GFLIB_ControllerPIp ... 377
2.23.1 Description ... 377
2.23.2 Re-entrancy ... 378
2.23.3 Function GFLIB_ControllerPIp_F32 ... 378
2.23.3.1 Declaration ... 378
2.23.3.2 Arguments ... 378
2.23.3.3 Return ... 378
2.23.3.4 Implementation details ... 379
2.23.3.5 Code Example ... 380
2.23.4 Function GFLIB_ControllerPIp_F16 ... 381
2.23.4.1 Declaration ... 381
2.23.4.2 Arguments ... 381
2.23.4.3 Return ... 381
2.23.4.4 Implementation details ... 382
2.23.4.5 Code Example ... 383
2.23.5 Function GFLIB_ControllerPIp_FLT ... 384
2.23.5.1 Declaration ... 384
2.23.5.2 Arguments ... 384
2.23.5.3 Return ... 384
2.23.5.4 Implementation details ... 385
2.23.6 Function GFLIB_ControllerPIpInit ... 385
2.23.6.1 Description ... 385
2.23.6.2 Re-entrancy ... 385
2.23.6.3 Function GFLIB_ControllerPIpInit_F32 ... 385
2.23.6.4 Function GFLIB_ControllerPIpInit_F16 ... 387
2.23.6.5 Function GFLIB_ControllerPIpInit_FLT ... 388
2.23.7 Function GFLIB_ControllerPIpSetState ... 390
2.23.7.1 Description ... 390
2.23.7.2 Re-entrancy ... 390
2.23.7.3 Function GFLIB_ControllerPIpSetState_ F32 ... 390
2.23.7.4 Function GFLIB_ControllerPIpSetState_ F16 ... 391
2.23.7.5 Function GFLIB_ControllerPIpSetState_ FLT ... 393
2.24 Function GFLIB_ControllerPIpAW ... 394
2.24.1 Description ... 395
2.24.2 Re-entrancy ... 396
2.24.3 Function GFLIB_ControllerPIpAW_F32 ... 396
2.24.3.1 Declaration ... 396
2.24.3.2 Arguments ... 396
2.24.3.3 Return ... 396
2.24.3.4 Implementation details ... 396
2.24.3.5 Code Example ... 398
2.24.4 Function GFLIB_ControllerPIpAW_F16 ... 399
2.24.4.1 Declaration ... 399
2.24.4.2 Arguments ... 399
2.24.4.3 Return ... 399
2.24.4.4 Implementation details ... 399
2.24.4.5 Code Example ... 401
2.24.5 Function GFLIB_ControllerPIpAW_FLT ... 402
2.24.5.1 Declaration ... 402
2.24.5.2 Arguments ... 402
2.24.5.3 Return ... 403
2.24.5.4 Implementation details ... 403
2.24.5.5 Code Example ... 403
2.24.6 Function GFLIB_ControllerPIpAWInit ... 405
2.24.6.1 Description ... 405
2.24.6.2 Re-entrancy ... 405
2.24.6.3 Function GFLIB_ControllerPIpAWInit_F32 ... 405
2.24.6.4 Function GFLIB_ControllerPIpAWInit_F16 ... 406
2.24.6.5 Function GFLIB_ControllerPIpAWInit_FLT ... 408
2.24.7 Function GFLIB_ControllerPIpAWSetState ... 409
2.24.7.1 Description ... 410
2.24.7.2 Re-entrancy ... 410
2.24.7.3 Function GFLIB_ControllerPIpAWSetState_ F32 ... 410
2.24.7.4 Function GFLIB_ControllerPIpAWSetState_ F16 ... 411
2.24.7.5 Function GFLIB_ControllerPIpAWSetState_ FLT ... 413
2.25 Function GFLIB_ControllerPIr ... 414
2.25.1 Description ... 415
2.25.2 Re-entrancy ... 415
2.25.3 Function GFLIB_ControllerPIr_F32 ... 415
2.25.3.1 Declaration ... 415
2.25.3.2 Arguments ... 416
2.25.3.3 Return ... 416
2.25.3.4 Implementation details ... 416
2.25.3.5 Code Example ... 417
2.25.4 Function GFLIB_ControllerPIr_F16 ... 418
2.25.4.1 Declaration ... 418
2.25.4.2 Arguments ... 418
2.25.4.3 Return ... 418
2.25.4.4 Implementation details ... 419
2.25.4.5 Code Example ... 420
2.25.5 Function GFLIB_ControllerPIr_FLT ... 421
2.25.5.1 Declaration ... 421
2.25.5.2 Arguments ... 421
2.25.5.3 Return ... 421
2.25.5.4 Implementation details ... 421
2.25.5.5 Code Example ... 421

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.25.6 Function GFLIB_ControllerPIrInit 423
2.25.6.1 Description 423
2.25.6.2 Re-entrancy 423
2.25.6.3 Function GFLIB_ControllerPIrInit_F32 423
2.25.6.4 Function GFLIB_ControllerPIrInit_F16 424
2.25.6.5 Function GFLIB_ControllerPIrInit_FLT 426
2.25.7 Function GFLIB_ControllerPIrSetState 427
2.25.7.1 Description 427
2.25.7.2 Re-entrancy 428
2.25.7.3 Function GFLIB_ControllerPIrSetState_F32 ..428
2.25.7.4 Function GFLIB_ControllerPIrSetState_F16 ..429
2.25.7.5 Function GFLIB_ControllerPIrSetState_FLT ..431
2.26 Function GFLIB_ControllerPIrAW 432
2.26.1 Description 432
2.26.2 Re-entrancy 433
2.26.3 Function GFLIB_ControllerPIrAW_F32 433
2.26.3.1 Declaration 433
2.26.3.2 Arguments 433
2.26.3.3 Return 433
2.26.3.4 Implementation details 433
2.26.3.5 Code Example 435
2.26.4 Function GFLIB_ControllerPIrAW_F16 436
2.26.4.1 Declaration 436
2.26.4.2 Arguments 436
2.26.4.3 Return 436
2.26.4.4 Implementation details 437
2.26.4.5 Code Example 438
2.26.5 Function GFLIB_ControllerPIrAW_FLT 439
2.26.5.1 Declaration 439
2.26.5.2 Arguments 439
2.26.5.3 Return 440
2.26.5.4 Implementation details 440
2.26.5.5 Code Example 440
2.26.6 Function GFLIB_ControllerPIrAWInit 441
2.26.6.1 Description 442
2.26.6.2 Re-entrancy 442
2.26.6.3 Function GFLIB_ControllerPIrAWInit_F32 442
2.26.6.4 Function GFLIB_ControllerPIrAWInit_F16 443
2.26.6.5 Function GFLIB_ControllerPIrAWInit_FLT 445
2.26.7 Function GFLIB_ControllerPIrAWSetState 446
2.26.7.1 Description 446
2.26.7.2 Re-entrancy 446
2.26.7.3 Function GFLIB_ControllerPIrAWSetState_F32 447
2.26.7.4 Function GFLIB_ControllerPIrAWSetState_F16 448
2.26.7.5 Function GFLIB_ControllerPIrAWSetState_FLT 450
2.27 Function GFLIB_Cos 451
2.27.1 Description 451
2.27.2 Re-entrancy 452
2.27.3 Function GFLIB_Cos_F32 452
2.27.3.1 Declaration 452
2.27.3.2 Arguments 452
2.27.3.3 Return 452
2.27.3.4 Implementation details 452
2.27.3.5 Code Example 453
2.27.4 Function GFLIB_Cos_F16 453
2.27.4.1 Declaration 453
2.27.4.2 Arguments 453
2.27.4.3 Return 454
2.27.4.4 Implementation details 454
2.27.4.5 Code Example 454
2.27.5 Function GFLIB_Cos_FLT 454
2.27.5.1 Declaration 454
2.27.5.2 Arguments 455
2.27.5.3 Return 455
2.27.5.4 Implementation details 455
2.27.5.5 Code Example 455
2.28 Function GFLIB_Hyst 456
2.28.1 Description 456
2.28.2 Re-entrancy 457
2.28.3 Function GFLIB_Hyst_F32 457
2.28.3.1 Declaration 457
2.28.3.2 Arguments 457
2.28.3.3 Return 457
2.28.3.4 Code Example 457
2.28.4 Function GFLIB_Hyst_F16 458
2.28.4.1 Declaration 458
2.28.4.2 Arguments 458
2.28.4.3 Return 458
2.28.4.4 Code Example 458
2.28.5 Function GFLIB_Hyst_FLT 459
2.28.5.1 Declaration 459
2.28.5.2 Arguments 459
2.28.5.3 Return 459
2.28.5.4 Code Example 460
2.29 Function GFLIB_IntegratorTR 460
2.29.1 Description 460
2.29.2 Re-entrancy 461
2.29.3 Function GFLIB_IntegratorTR_F32 461
2.29.3.1 Declaration 461
2.29.3.2 Arguments 461
2.29.3.3 Return 461
2.29.3.4 Implementation details 461
2.29.3.5 Code Example 463
2.29.4 Function GFLIB_IntegratorTR_F16 464
2.29.4.1 Declaration 464
2.29.4.2 Arguments 464
2.29.4.3 Return 464
2.29.4.4 Implementation details 464
2.29.4.5 Code Example 465
2.29.5 Function GFLIB_IntegratorTR_FLT 466
2.29.5.1 Declaration 466
2.29.5.2 Arguments 466
2.29.5.3 Return 466
2.29.5.4 Implementation details 466
2.29.5.5 Code Example 468
2.29.6 Function GFLIB_IntegratorTRSetState 468
2.29.6.1 Description 469
2.29.6.2 Re-entrancy 469
2.29.6.3 Function GFLIB_IntegratorTRSetState_F32 ..469
2.29.6.4 Function GFLIB_IntegratorTRSetState_F16 ..470
2.29.6.5 Function GFLIB_IntegratorTRSetState_FLT ..471
2.30 Function GFLIB_Limit 472
2.30.1 Description 472
2.30.2 Re-entrancy 472
2.30.3 Function GFLIB_Limit_F32 472

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.30.3.1 Declaration 472
2.30.3.2 Arguments 472
2.30.3.3 Return 473
2.30.3.4 Code Example 473
2.30.4 Function GFLIB_Limit_F16 473
2.30.4.1 Declaration 473
2.30.4.2 Arguments 473
2.30.4.3 Return 474
2.30.4.4 Code Example 474
2.30.5 Function GFLIB_Limit_FLT 474
2.30.5.1 Declaration 474
2.30.5.2 Arguments 474
2.30.5.3 Return 475
2.30.5.4 Code Example 475
2.31 Function GFLIB_Log10_FLT 475
2.31.1 Declaration 475
2.31.2 Arguments 476
2.31.3 Return 476
2.31.4 Description 476
2.31.5 Code Example 476
2.32 Function GFLIB_LowerLimit 476
2.32.1 Description 477
2.32.2 Re-entrancy 477
2.32.3 Function GFLIB_LowerLimit_F32 477
2.32.3.1 Declaration 477
2.32.3.2 Arguments 477
2.32.3.3 Return 477
2.32.3.4 Code Example 477
2.32.4 Function GFLIB_LowerLimit_F16 478
2.32.4.1 Declaration 478
2.32.4.2 Arguments 478
2.32.4.3 Return 478
2.32.4.4 Code Example 478
2.32.5 Function GFLIB_LowerLimit_FLT 479
2.32.5.1 Declaration 479
2.32.5.2 Arguments 479
2.32.5.3 Return 479
2.32.5.4 Code Example 479
2.33 Function GFLIB_Lut1D 480
2.33.1 Description 480
2.33.2 Re-entrancy 480
2.33.3 Function GFLIB_Lut1D_F32 480
2.33.3.1 Declaration 480
2.33.3.2 Arguments 481
2.33.3.3 Return 481
2.33.3.4 Implementation details 481
2.33.3.5 Code Example 482
2.33.4 Function GFLIB_Lut1D_F16 484
2.33.4.1 Declaration 484
2.33.4.2 Arguments 484
2.33.4.3 Return 484
2.33.4.4 Implementation details 484
2.33.4.5 Code Example 486
2.33.5 Function GFLIB_Lut1D_FLT 487
2.33.5.1 Declaration 487
2.33.5.2 Arguments 487
2.33.5.3 Return 487
2.33.5.4 Implementation details 487
2.33.5.5 Code Example 489
2.34 Function GFLIB_Lut2D 490
2.34.1 Description 490
2.34.2 Re-entrancy 491
2.34.3 Function GFLIB_Lut2D_F32 491
2.34.3.1 Declaration 491
2.34.3.2 Arguments 491
2.34.3.3 Return 492
2.34.3.4 Implementation details 492
2.34.3.5 Code Example 493
2.34.4 Function GFLIB_Lut2D_F16 494
2.34.4.1 Declaration 494
2.34.4.2 Arguments 494
2.34.4.3 Return 494
2.34.4.4 Implementation details 494
2.34.4.5 Code Example 495
2.34.5 Function GFLIB_Lut2D_FLT 496
2.34.5.1 Declaration 496
2.34.5.2 Arguments 496
2.34.5.3 Return 497
2.34.5.4 Implementation details 497
2.34.5.5 Code Example 498
2.35 Function GFLIB_Ramp 499
2.35.1 Description 499
2.35.2 Re-entrancy 500
2.35.3 Function GFLIB_Ramp_F32 500
2.35.3.1 Declaration 500
2.35.3.2 Arguments 500
2.35.3.3 Return 500
2.35.3.4 Code Example 500
2.35.4 Function GFLIB_Ramp_F16 501
2.35.4.1 Declaration 501
2.35.4.2 Arguments 501
2.35.4.3 Return 501
2.35.4.4 Code Example 501
2.35.5 Function GFLIB_Ramp_FLT 502
2.35.5.1 Declaration 502
2.35.5.2 Arguments 502
2.35.5.3 Return 502
2.35.5.4 Code Example 502
2.36 Function GFLIB_Sign 503
2.36.1 Description 503
2.36.2 Re-entrancy 503
2.36.3 Function GFLIB_Sign_F32 504
2.36.3.1 Declaration 504
2.36.3.2 Arguments 504
2.36.3.3 Return 504
2.36.3.4 Implementation details 504
2.36.3.5 Code Example 504
2.36.4 Function GFLIB_Sign_F16 504
2.36.4.1 Declaration 504
2.36.4.2 Arguments 505
2.36.4.3 Return 505
2.36.4.4 Implementation details 505
2.36.4.5 Code Example 505
2.36.5 Function GFLIB_Sign_FLT 505
2.36.5.1 Declaration 505
2.36.5.2 Arguments 505
2.36.5.3 Return 506
2.36.5.4 Code Example 506

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.37 Function GFLIB_Sin 506
2.37.1 Description 506
2.37.2 Re-entrancy 507
2.37.3 Function GFLIB_Sin_F32 507
2.37.3.1 Declaration 507
2.37.3.2 Arguments 507
2.37.3.3 Return 507
2.37.3.4 Implementation details 507
2.37.3.5 Code Example 508
2.37.4 Function GFLIB_Sin_F16 508
2.37.4.1 Declaration 508
2.37.4.2 Arguments 508
2.37.4.3 Return 508
2.37.4.4 Implementation details 508
2.37.4.5 Code Example 509
2.37.5 Function GFLIB_Sin_FLT 509
2.37.5.1 Declaration 509
2.37.5.2 Arguments 509
2.37.5.3 Return 509
2.37.5.4 Implementation details 510
2.37.5.5 Code Example 510
2.38 Function GFLIB_SinCos 510
2.38.1 Description 510
2.38.2 Re-entrancy 510
2.38.3 Function GFLIB_SinCos_F32 511
2.38.3.1 Declaration 511
2.38.3.2 Arguments 511
2.38.3.3 Return 511
2.38.3.4 Implementation details 511
2.38.3.5 Code Example 511
2.38.4 Function GFLIB_SinCos_F16 512
2.38.4.1 Declaration 512
2.38.4.2 Arguments 512
2.38.4.3 Return 512
2.38.4.4 Implementation details 512
2.38.4.5 Code Example 513
2.38.5 Function GFLIB_SinCos_FLT 513
2.38.5.1 Declaration 513
2.38.5.2 Arguments 513
2.38.5.3 Return 514
2.38.5.4 Implementation details 514
2.38.5.5 Code Example 514
2.39 Function GFLIB_Sqrt 515
2.39.1 Description 515
2.39.2 Re-entrancy 515
2.39.3 Function GFLIB_Sqrt_F32 515
2.39.3.1 Declaration 515
2.39.3.2 Arguments 515
2.39.3.3 Return 515
2.39.3.4 Implementation details 515
2.39.3.5 Code Example 515
2.39.4 Function GFLIB_Sqrt_F16 516
2.39.4.1 Declaration 516
2.39.4.2 Arguments 516
2.39.4.3 Return 516
2.39.4.4 Implementation details 516
2.39.4.5 Code Example 516
2.39.5 Function GFLIB_Sqrt_FLT 517
2.39.5.1 Declaration 517
2.39.5.2 Arguments 517
2.39.5.3 Return 517
2.39.5.4 Code Example 517
2.40 Function GFLIB_Tan 518
2.40.1 Description 518
2.40.2 Re-entrancy 518
2.40.3 Function GFLIB_Tan_F32 518
2.40.3.1 Declaration 518
2.40.3.2 Arguments 518
2.40.3.3 Return 518
2.40.3.4 Implementation details 518
2.40.3.5 Code Example 519
2.40.4 Function GFLIB_Tan_F16 519
2.40.4.1 Declaration 519
2.40.4.2 Arguments 520
2.40.4.3 Return 520
2.40.4.4 Implementation details 520
2.40.4.5 Code Example 520
2.40.5 Function GFLIB_Tan_FLT 521
2.40.5.1 Declaration 521
2.40.5.2 Arguments 521
2.40.5.3 Return 521
2.40.5.4 Implementation details 521
2.40.5.5 Code Example 522
2.41 Function GFLIB_UpperLimit 522
2.41.1 Description 523
2.41.2 Re-entrancy 523
2.41.3 Function GFLIB_UpperLimit_F32 523
2.41.3.1 Declaration 523
2.41.3.2 Arguments 523
2.41.3.3 Return 523
2.41.3.4 Code Example 523
2.41.4 Function GFLIB_UpperLimit_F16 524
2.41.4.1 Declaration 524
2.41.4.2 Arguments 524
2.41.4.3 Return 524
2.41.4.4 Code Example 524
2.41.5 Function GFLIB_UpperLimit_FLT 525
2.41.5.1 Declaration 525
2.41.5.2 Arguments 525
2.41.5.3 Return 525
2.41.5.4 Code Example 525
2.42 Function GFLIB_VectorLimit 526
2.42.1 Description 526
2.42.2 Re-entrancy 527
2.42.3 Function GFLIB_VectorLimit_F32 527
2.42.3.1 Declaration 527
2.42.3.2 Arguments 527
2.42.3.3 Return 527
2.42.3.4 Implementation details 527
2.42.3.5 Code Example 527
2.42.4 Function GFLIB_VectorLimit_F16 528
2.42.4.1 Declaration 528
2.42.4.2 Arguments 528
2.42.4.3 Return 529
2.42.4.4 Code Example 529
2.42.5 Function GFLIB_VectorLimit_FLT 530
2.42.5.1 Declaration 530
2.42.5.2 Arguments 530

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.42.5.3 Return ... 530
2.42.5.4 Code Example ... 530
2.43 Function GFLIB_VLog10_FLT ... 531
2.43.1 Declaration ... 531
2.43.2 Arguments ... 531
2.43.3 Description ... 531
2.43.4 Code Example ... 532
2.43.5 Reentrancy ... 532
2.44 Function GFLIB_VMin ... 532
2.44.1 Description ... 532
2.44.2 Re-entrancy ... 532
2.44.3 Function GFLIB_VMin_F32 ... 532
2.44.3.1 Declaration ... 532
2.44.3.2 Arguments ... 532
2.44.3.3 Return ... 533
2.44.3.4 Code Example ... 533
2.44.4 Function GFLIB_VMin_F16 ... 533
2.44.4.1 Declaration ... 533
2.44.4.2 Arguments ... 533
2.44.4.3 Return ... 533
2.44.4.4 Code Example ... 533
2.44.5 Function GFLIB_VMin_FLT ... 534
2.44.5.1 Declaration ... 534
2.44.5.2 Arguments ... 534
2.44.5.3 Return ... 534
2.44.5.4 Code Example ... 534
2.44.6 Function GFLIB_VMin4_F16 ... 535
2.44.6.1 Declaration ... 535
2.44.6.2 Arguments ... 535
2.44.6.3 Return ... 535
2.44.6.4 Implementation details ... 535
2.44.6.5 Code Example ... 535
2.44.7 Function GFLIB_VMin5_F16 ... 535
2.44.7.1 Declaration ... 535
2.44.7.2 Arguments ... 536
2.44.7.3 Return ... 536
2.44.7.4 Implementation details ... 536
2.44.7.5 Code Example ... 536
2.44.8 Function GFLIB_VMin6_F16 ... 536
2.44.8.1 Declaration ... 536
2.44.8.2 Arguments ... 536
2.44.8.3 Return ... 536
2.44.8.4 Implementation details ... 536
2.44.8.5 Code Example ... 537
2.44.9 Function GFLIB_VMin7_F16 ... 537
2.44.9.1 Declaration ... 537
2.44.9.2 Arguments ... 537
2.44.9.3 Return ... 537
2.44.9.4 Implementation details ... 537
2.44.9.5 Code Example ... 537
2.44.10 Function GFLIB_VMin8_F16 ... 537
2.44.10.1 Declaration ... 537
2.44.10.2 Arguments ... 538
2.44.10.3 Return ... 538
2.44.10.4 Implementation details ... 538
2.44.10.5 Code Example ... 538
2.44.11 Function GFLIB_VMin9_F16 ... 538
2.44.11.1 Declaration ... 538
2.44.11.2 Arguments ... 538
2.44.11.3 Return ... 538
2.44.11.4 Implementation details ... 538
2.44.11.5 Code Example ... 539
2.44.12 Function GFLIB_VMin10_F16 ... 539
2.44.12.1 Declaration ... 539
2.44.12.2 Arguments ... 539
2.44.12.3 Return ... 539
2.44.12.4 Implementation details ... 539
2.44.12.5 Code Example ... 539
2.44.13 Function GFLIB_VMin11_F16 ... 539
2.44.13.1 Declaration ... 539
2.44.13.2 Arguments ... 540
2.44.13.3 Return ... 540
2.44.13.4 Implementation details ... 540
2.44.13.5 Code Example ... 540
2.44.14 Function GFLIB_VMin12_F16 ... 540
2.44.14.1 Declaration ... 540
2.44.14.2 Arguments ... 540
2.44.14.3 Return ... 540
2.44.14.4 Implementation details ... 540
2.44.14.5 Code Example ... 541
2.44.15 Function GFLIB_VMin13_F16 ... 541
2.44.15.1 Declaration ... 541
2.44.15.2 Arguments ... 541
2.44.15.3 Return ... 541
2.44.15.4 Implementation details ... 541
2.44.15.5 Code Example ... 541
2.44.16 Function GFLIB_VMin14_F16 ... 542
2.44.16.1 Declaration ... 542
2.44.16.2 Arguments ... 542
2.44.16.3 Return ... 542
2.44.16.4 Implementation details ... 542
2.44.16.5 Code Example ... 542
2.44.17 Function GFLIB_VMin15_F16 ... 542
2.44.17.1 Declaration ... 542
2.44.17.2 Arguments ... 542
2.44.17.3 Return ... 542
2.44.17.4 Implementation details ... 543
2.44.17.5 Code Example ... 543
2.44.18 Function GFLIB_VMin16_F16 ... 543
2.44.18.1 Declaration ... 543
2.44.18.2 Arguments ... 543
2.44.18.3 Return ... 543
2.44.18.4 Implementation details ... 543
2.44.18.5 Code Example ... 543
2.45 Function GMCLIB_BetaProjection ... 544
2.45.1 Description ... 544
2.45.2 Re-entrancy ... 544
2.45.3 Function GMCLIB_BetaProjection_F32 ... 544
2.45.3.1 Declaration ... 544
2.45.3.2 Arguments ... 544
2.45.3.3 Code Example ... 545
2.45.4 Function GMCLIB_BetaProjection_F16 ... 545
2.45.4.1 Declaration ... 545
2.45.4.2 Arguments ... 545
2.45.4.3 Code Example ... 546
2.45.5 Function GMCLIB_BetaProjection_FLT ... 546
2.45.5.1 Declaration ... 546
2.45.5.2 Arguments ... 546

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.45.5.3 Code Example 547
2.46 Function GMCLIB_BetaProjection3Ph 547
2.46.1 Description 547
2.46.2 Re-entrancy 548
2.46.3 Function GMCLIB_BetaProjection3Ph_F32 548
2.46.3.1 Declaration 548
2.46.3.2 Arguments 548
2.46.3.3 Code Example 548
2.46.4 Function GMCLIB_BetaProjection3Ph_F16 549
2.46.4.1 Declaration 549
2.46.4.2 Arguments 549
2.46.4.3 Code Example 549
2.46.5 Function GMCLIB_BetaProjection3Ph_FLT 550
2.46.5.1 Declaration 550
2.46.5.2 Arguments 550
2.46.5.3 Code Example 551
2.47 Function GMCLIB_Clark 552
2.47.1 Description 552
2.47.2 Re-entrancy 552
2.47.3 Function GMCLIB_Clark_F32 552
2.47.3.1 Declaration 552
2.47.3.2 Arguments 552
2.47.3.3 Code Example 552
2.47.4 Function GMCLIB_Clark_F16 553
2.47.4.1 Declaration 553
2.47.4.2 Arguments 553
2.47.4.3 Code Example 553
2.47.5 Function GMCLIB_Clark_FLT 554
2.47.5.1 Declaration 554
2.47.5.2 Arguments 554
2.47.5.3 Code Example 555
2.48 Function GMCLIB_Clark6Ph 555
2.48.1 Description 555
2.48.2 Re-entrancy 556
2.48.3 Function GMCLIB_Clark6Ph_F32 556
2.48.3.1 Declaration 556
2.48.3.2 Arguments 556
2.48.3.3 Code Example 556
2.48.4 Function GMCLIB_Clark6Ph_F16 557
2.48.4.1 Declaration 557
2.48.4.2 Arguments 557
2.48.4.3 Code Example 558
2.48.5 Function GMCLIB_Clark6Ph_FLT 559
2.48.5.1 Declaration 559
2.48.5.2 Arguments 559
2.48.5.3 Code Example 559
2.48.6 Function GMCLIB_Clark6PhInit 560
2.48.6.1 Description 560
2.48.6.2 Re-entrancy 561
2.48.6.3 Function GMCLIB_Clark6PhInit_F32 561
2.48.6.4 Function GMCLIB_Clark6PhInit_F16 562
2.48.6.5 Function GMCLIB_Clark6PhInit_FLT 564
2.48.6.6 Function GMCLIB_ClarkInv6PhInit_F16 565
2.48.6.7 Function GMCLIB_ClarkInv6PhInit_FLT 567
2.48.6.8 Function GMCLIB_SvmStd6PhInit_F16 568
2.49 Function GMCLIB_ClarkInv 569
2.49.1 Description 569
2.49.2 Re-entrancy 569
2.49.3 Function GMCLIB_ClarkInv_F32 569
2.49.3.1 Declaration 569
2.49.3.2 Arguments 570
2.49.3.3 Code Example 570
2.49.4 Function GMCLIB_ClarkInv_F16 571
2.49.4.1 Declaration 571
2.49.4.2 Arguments 571
2.49.4.3 Code Example 571
2.49.5 Function GMCLIB_ClarkInv_FLT 572
2.49.5.1 Declaration 572
2.49.5.2 Arguments 572
2.49.5.3 Code Example 572
2.50 Function GMCLIB_ClarkInv6Ph 573
2.50.1 Description 573
2.50.2 Re-entrancy 573
2.50.3 Function GMCLIB_ClarkInv6Ph_F32 573
2.50.3.1 Declaration 573
2.50.3.2 Arguments 573
2.50.3.3 Code Example 574
2.50.4 Function GMCLIB_ClarkInv6Ph_F16 575
2.50.4.1 Declaration 575
2.50.4.2 Arguments 575
2.50.4.3 Code Example 575
2.50.5 Function GMCLIB_ClarkInv6Ph_FLT 576
2.50.5.1 Declaration 576
2.50.5.2 Arguments 576
2.50.5.3 Implementation details 576
2.50.5.4 Code Example 576
2.50.6 Function GMCLIB_ClarkInv6PhInit 577
2.50.6.1 Description 577
2.50.6.2 Re-entrancy 578
2.50.6.3 Function GMCLIB_ClarkInv6PhInit_F32 578
2.51 Function GMCLIB_DecouplingPMSM 579
2.51.1 Description 579
2.51.2 Re-entrancy 580
2.51.3 Function GMCLIB_DecouplingPMSM_F32 581
2.51.3.1 Declaration 581
2.51.3.2 Arguments 581
2.51.3.3 Implementation details 581
2.51.3.4 Code Example 583
2.51.4 Function GMCLIB_DecouplingPMSM_F16 584
2.51.4.1 Declaration 584
2.51.4.2 Arguments 584
2.51.4.3 Implementation details 584
2.51.4.4 Code Example 586
2.51.5 Function GMCLIB_DecouplingPMSM_FLT 587
2.51.5.1 Declaration 587
2.51.5.2 Arguments 587
2.51.5.3 Code Example 587
2.52 Function GMCLIB_DoubleSwitching_F16 588
2.52.1 Declaration 588
2.52.2 Arguments 588
2.52.3 Description 589
2.52.4 Code Example 593
2.53 Function GMCLIB_ElimDcBusRip 593
2.53.1 Description 593
2.53.2 Re-entrancy 594
2.53.3 Function GMCLIB_ElimDcBusRip_F32 594
2.53.3.1 Declaration 594
2.53.3.2 Arguments 594

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.53.3.3 Return ... 594
2.53.3.4 Implementation details ... 594
2.53.3.5 Code Example ... 596
2.53.4 Function GMCLIB_ElimDcBusRip_F16 ... 597
2.53.4.1 Declaration ... 597
2.53.4.2 Arguments ... 597
2.53.4.3 Return ... 597
2.53.4.4 Implementation details ... 597
2.53.4.5 Code Example ... 598
2.53.5 Function GMCLIB_ElimDcBusRip_FLT ... 599
2.53.5.1 Declaration ... 599
2.53.5.2 Arguments ... 599
2.53.5.3 Return ... 600
2.53.5.4 Implementation details ... 600
2.53.5.5 Code Example ... 601
2.54 Function GMCLIB_Park ... 602
2.54.1 Description ... 602
2.54.2 Re-entrancy ... 602
2.54.3 Function GMCLIB_Park_F32 ... 602
2.54.3.1 Declaration ... 602
2.54.3.2 Arguments ... 602
2.54.3.3 Code Example ... 603
2.54.4 Function GMCLIB_Park_F16 ... 604
2.54.4.1 Declaration ... 604
2.54.4.2 Arguments ... 604
2.54.4.3 Code Example ... 604
2.54.5 Function GMCLIB_Park_FLT ... 605
2.54.5.1 Declaration ... 605
2.54.5.2 Arguments ... 605
2.54.5.3 Code Example ... 605
2.55 Function GMCLIB_ParkInv ... 606
2.55.1 Description ... 606
2.55.2 Re-entrancy ... 606
2.55.3 Function GMCLIB_ParkInv_F32 ... 606
2.55.3.1 Declaration ... 606
2.55.3.2 Arguments ... 606
2.55.3.3 Code Example ... 607
2.55.4 Function GMCLIB_ParkInv_F16 ... 607
2.55.4.1 Declaration ... 607
2.55.4.2 Arguments ... 608
2.55.4.3 Code Example ... 608
2.55.5 Function GMCLIB_ParkInv_FLT ... 609
2.55.5.1 Declaration ... 609
2.55.5.2 Arguments ... 609
2.55.5.3 Code Example ... 609
2.56 Function GMCLIB_PwmIct ... 610
2.56.1 Description ... 610
2.56.2 Re-entrancy ... 611
2.56.3 Function GMCLIB_PwmIct_F32 ... 611
2.56.3.1 Declaration ... 611
2.56.3.2 Arguments ... 611
2.56.3.3 Return ... 611
2.56.3.4 Code Example ... 611
2.56.4 Function GMCLIB_PwmIct_F16 ... 612
2.56.4.1 Declaration ... 612
2.56.4.2 Arguments ... 612
2.56.4.3 Return ... 612
2.56.4.4 Code Example ... 612
2.56.5 Function GMCLIB_PwmIct_FLT ... 613
2.56.5.1 Declaration ... 613
2.56.5.2 Arguments ... 613
2.56.5.3 Return ... 613
2.56.5.4 Implementation details ... 613
2.56.5.5 Code Example ... 614
2.57 Function GMCLIB_SpreadPWM ... 614
2.57.1 Declaration ... 614
2.57.2 Arguments ... 615
2.57.3 Return ... 615
2.57.4 Description ... 615
2.57.5 Code Example ... 616
2.57.6 Re-entrancy ... 617
2.57.7 Function GMCLIB_SpreadPWMInit ... 617
2.57.7.1 Declaration ... 617
2.57.7.2 Arguments ... 617
2.57.7.3 Description ... 617
2.57.7.4 Code Example ... 617
2.57.7.5 Re-entrancy ... 618
2.58 Function GMCLIB_SvmSci ... 618
2.58.1 Description ... 618
2.58.2 Re-entrancy ... 620
2.58.3 Function GMCLIB_SvmSci_F32 ... 620
2.58.3.1 Declaration ... 620
2.58.3.2 Arguments ... 621
2.58.3.3 Return ... 621
2.58.3.4 Code Example ... 621
2.58.4 Function GMCLIB_SvmSci_F16 ... 622
2.58.4.1 Declaration ... 622
2.58.4.2 Arguments ... 622
2.58.4.3 Return ... 622
2.58.4.4 Code Example ... 622
2.58.5 Function GMCLIB_SvmSci_FLT ... 623
2.58.5.1 Declaration ... 623
2.58.5.2 Arguments ... 623
2.58.5.3 Return ... 623
2.58.5.4 Implementation details ... 623
2.58.5.5 Code Example ... 623
2.59 Function GMCLIB_SvmStd ... 624
2.59.1 Description ... 624
2.59.2 Re-entrancy ... 634
2.59.3 Function GMCLIB_SvmStd_F32 ... 634
2.59.3.1 Declaration ... 634
2.59.3.2 Arguments ... 634
2.59.3.3 Return ... 635
2.59.3.4 Code Example ... 635
2.59.4 Function GMCLIB_SvmStd_F16 ... 635
2.59.4.1 Declaration ... 635
2.59.4.2 Arguments ... 636
2.59.4.3 Return ... 636
2.59.4.4 Code Example ... 636
2.59.5 Function GMCLIB_SvmStd_FLT ... 637
2.59.5.1 Declaration ... 637
2.59.5.2 Arguments ... 637
2.59.5.3 Return ... 637
2.59.5.4 Implementation details ... 637
2.59.5.5 Code Example ... 637
2.60 Function GMCLIB_SvmStd6Ph ... 638
2.60.1 Description ... 638
2.60.2 Re-entrancy ... 638

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.60.3 Function GMCLIB_SvmStd6Ph_F32 638
2.60.3.1 Declaration 638
2.60.3.2 Arguments 638
2.60.3.3 Code Example 639
2.60.4 Function GMCLIB_SvmStd6Ph_F16 640
2.60.4.1 Declaration 640
2.60.4.2 Arguments 640
2.60.4.3 Code Example 640
2.60.5 Function GMCLIB_SvmStd6Ph_FLT 641
2.60.5.1 Declaration 641
2.60.5.2 Arguments 641
2.60.5.3 Implementation details 641
2.60.5.4 Code Example 642
2.60.6 Function GMCLIB_SvmStd6PhInit 642
2.60.6.1 Description 643
2.60.6.2 Re-entrancy 643
2.60.6.3 Function GMCLIB_SvmStd6PhInit_F32 643
2.60.6.4 Function GMCLIB_SvmStd6PhInit_FLT 644
2.61 Function GMCLIB_SvmU0n 645
2.61.1 Description 646
2.61.2 Re-entrancy 647
2.61.3 Function GMCLIB_SvmU0n_F32 647
2.61.3.1 Declaration 647
2.61.3.2 Arguments 647
2.61.3.3 Return 648
2.61.3.4 Code Example 648
2.61.4 Function GMCLIB_SvmU0n_F16 648
2.61.4.1 Declaration 648
2.61.4.2 Arguments 649
2.61.4.3 Return 649
2.61.4.4 Code Example 649
2.61.5 Function GMCLIB_SvmU0n_FLT 650
2.61.5.1 Declaration 650
2.61.5.2 Arguments 650
2.61.5.3 Return 650
2.61.5.4 Implementation details 650
2.61.5.5 Code Example 650
2.62 Function GMCLIB_SvmU7n 651
2.62.1 Description 651
2.62.2 Re-entrancy 653
2.62.3 Function GMCLIB_SvmU7n_F32 653
2.62.3.1 Declaration 653
2.62.3.2 Arguments 653
2.62.3.3 Return 654
2.62.3.4 Code Example 654
2.62.4 Function GMCLIB_SvmU7n_F16 655
2.62.4.1 Declaration 655
2.62.4.2 Arguments 655
2.62.4.3 Return 655
2.62.4.4 Code Example 655
2.62.5 Function GMCLIB_SvmU7n_FLT 656
2.62.5.1 Declaration 656
2.62.5.2 Arguments 656
2.62.5.3 Return 656
2.62.5.4 Implementation details 656
2.62.5.5 Code Example 656
2.63 Function GMCLIB_Vmag2coord 657
2.63.1 Description 657
2.63.2 Re-entrancy 657
2.63.3 Function GMCLIB_Vmag2coord_F32 657
2.63.3.1 Declaration 657
2.63.3.2 Arguments 658
2.63.3.3 Return 658
2.63.3.4 Implementation details 658
2.63.3.5 Code Example: 658
2.63.4 Function GMCLIB_Vmag2coord_F16 658
2.63.4.1 Declaration 658
2.63.4.2 Arguments 658
2.63.4.3 Return 659
2.63.4.4 Implementation details 659
2.63.4.5 Code Example: 659
2.63.5 Function GMCLIB_Vmag2coord_FLT 659
2.63.5.1 Declaration 659
2.63.5.2 Arguments 659
2.63.5.3 Return 660
2.63.5.4 Implementation details 660
2.63.5.5 Code Example 660
2.64 Function GMCLIB_VRot 660
2.64.1 Description 660
2.64.2 Re-entrancy 661
2.64.3 Function GMCLIB_VRot_F32 661
2.64.3.1 Declaration 661
2.64.3.2 Arguments 661
2.64.3.3 Implementation details 661
2.64.3.4 Code Example: 661
2.64.4 Function GMCLIB_VRot_F16 662
2.64.4.1 Declaration 662
2.64.4.2 Arguments 662
2.64.4.3 Implementation details 662
2.64.4.4 Code Example: 662
2.64.5 Function GMCLIB_VRot_FLT 663
2.64.5.1 Declaration 663
2.64.5.2 Arguments 663
2.64.5.3 Implementation details 663
2.64.5.4 Code Example 663
2.65 Function GMCLIB_VUnit 664
2.65.1 Description 664
2.65.2 Re-entrancy 664
2.65.3 Function GMCLIB_VUnit_F32 664
2.65.3.1 Declaration 664
2.65.3.2 Arguments 665
2.65.3.3 Implementation details 665
2.65.3.4 Code Example: 665
2.65.4 Function GMCLIB_VUnit_F16 665
2.65.4.1 Declaration 665
2.65.4.2 Arguments 665
2.65.4.3 Implementation details 666
2.65.4.4 Code Example: 666
2.65.5 Function GMCLIB_VUnit_FLT 666
2.65.5.1 Declaration 666
2.65.5.2 Arguments 666
2.65.5.3 Implementation details 667
2.65.5.4 Code Example 667
2.66 Function MLIB_Abs 667
2.66.1 Description 667
2.66.2 Re-entrancy 667
2.66.3 Function MLIB_Abs_F32 667
2.66.3.1 Declaration 667

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.66.3.2 Arguments 668
2.66.3.3 Return 668
2.66.3.4 Implementation details 668
2.66.3.5 Code Example 668
2.66.4 Function MLIB_Abs_F16 668
2.66.4.1 Declaration 668
2.66.4.2 Arguments 669
2.66.4.3 Return 669
2.66.4.4 Implementation details 669
2.66.4.5 Code Example 669
2.66.5 Function MLIB_Abs_FLT 669
2.66.5.1 Declaration 669
2.66.5.2 Arguments 670
2.66.5.3 Return 670
2.66.5.4 Implementation details 670
2.66.5.5 Code Example 670
2.67 Function MLIB_AbsSat 671
2.67.1 Description 671
2.67.2 Re-entrancy 671
2.67.3 Function MLIB_AbsSat_F32 671
2.67.3.1 Declaration 671
2.67.3.2 Arguments 671
2.67.3.3 Return 671
2.67.3.4 Implementation details 671
2.67.3.5 Code Example 671
2.67.4 Function MLIB_AbsSat_F16 672
2.67.4.1 Declaration 672
2.67.4.2 Arguments 672
2.67.4.3 Return 672
2.67.4.4 Implementation details 672
2.67.4.5 Code Example 672
2.68 Function MLIB_Add 673
2.68.1 Description 673
2.68.2 Re-entrancy 673
2.68.3 Function MLIB_Add_F32 673
2.68.3.1 Declaration 673
2.68.3.2 Arguments 673
2.68.3.3 Return 673
2.68.3.4 Implementation details 673
2.68.3.5 Code Example: 674
2.68.4 Function MLIB_Add_F16 674
2.68.4.1 Declaration 674
2.68.4.2 Arguments 674
2.68.4.3 Return 674
2.68.4.4 Implementation details 674
2.68.4.5 Code Example 675
2.68.5 Function MLIB_Add_FLT 675
2.68.5.1 Declaration 675
2.68.5.2 Arguments 675
2.68.5.3 Return 675
2.68.5.4 Code Example 676
2.69 Function MLIB_AddSat 676
2.69.1 Description 676
2.69.2 Re-entrancy 676
2.69.3 Function MLIB_AddSat_F32 676
2.69.3.1 Declaration 676
2.69.3.2 Arguments 677
2.69.3.3 Return 677
2.69.3.4 Implementation details 677
2.69.3.5 Code Example 677
2.69.4 Function MLIB_AddSat_F16 678
2.69.4.1 Declaration 678
2.69.4.2 Arguments 678
2.69.4.3 Return 678
2.69.4.4 Implementation details 678
2.69.4.5 Code Example 678
2.70 Function MLIB_Convert 679
2.70.1 Description 679
2.70.2 Re-entrancy 679
2.70.3 Function MLIB_Convert_F32F16 679
2.70.3.1 Declaration 679
2.70.3.2 Arguments 679
2.70.3.3 Return 679
2.70.3.4 Implementation details 679
2.70.3.5 Code Example 680
2.70.4 Function MLIB_Convert_F32FLT 680
2.70.4.1 Declaration 680
2.70.4.2 Arguments 680
2.70.4.3 Return 680
2.70.4.4 Implementation details 680
2.70.4.5 Code Example 681
2.70.5 Function MLIB_Convert_F16F32 681
2.70.5.1 Declaration 681
2.70.5.2 Arguments 682
2.70.5.3 Return 682
2.70.5.4 Implementation details 682
2.70.5.5 Code Example 682
2.70.6 Function MLIB_Convert_F16FLT 683
2.70.6.1 Declaration 683
2.70.6.2 Arguments 683
2.70.6.3 Return 683
2.70.6.4 Implementation details 683
2.70.6.5 Code Example 683
2.70.7 Function MLIB_Convert_FLTF16 684
2.70.7.1 Declaration 684
2.70.7.2 Arguments 684
2.70.7.3 Return 684
2.70.7.4 Implementation details 684
2.70.7.5 Code Example 685
2.70.8 Function MLIB_Convert_FLTF32 685
2.70.8.1 Declaration 685
2.70.8.2 Arguments 686
2.70.8.3 Return 686
2.70.8.4 Implementation details 686
2.70.8.5 Code Example 686
2.71 Function MLIB_ConvertPU 687
2.71.1 Description 687
2.71.2 Re-entrancy 687
2.71.3 Function MLIB_ConvertPU_F32F16 687
2.71.3.1 Declaration 687
2.71.3.2 Arguments 687
2.71.3.3 Return 687
2.71.3.4 Implementation details 687
2.71.3.5 Code Example 688
2.71.4 Function MLIB_ConvertPU_F32FLT 688
2.71.4.1 Declaration 688
2.71.4.2 Arguments 688
2.71.4.3 Return 688

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.71.4.4 Implementation details 688
2.71.4.5 Code Example 689
2.71.5 Function MLIB_ConvertPU_F16F32 689
2.71.5.1 Declaration 689
2.71.5.2 Arguments 689
2.71.5.3 Return 689
2.71.5.4 Implementation details 689
2.71.5.5 Code Example 689
2.71.6 Function MLIB_ConvertPU_F16FLT 690
2.71.6.1 Declaration 690
2.71.6.2 Arguments 690
2.71.6.3 Return 690
2.71.6.4 Implementation details 690
2.71.6.5 Code Example 690
2.71.7 Function MLIB_ConvertPU_FLTF16 691
2.71.7.1 Declaration 691
2.71.7.2 Arguments 691
2.71.7.3 Return 691
2.71.7.4 Implementation details 691
2.71.7.5 Code Example 691
2.71.8 Function MLIB_ConvertPU_FLTF32 692
2.71.8.1 Declaration 692
2.71.8.2 Arguments 692
2.71.8.3 Return 692
2.71.8.4 Implementation details 692
2.71.8.5 Code Example 692
2.72 Function MLIB_Div 693
2.72.1 Description 693
2.72.2 Re-entrancy 693
2.72.3 Function MLIB_Div_F32 693
2.72.3.1 Declaration 693
2.72.3.2 Arguments 693
2.72.3.3 Return 693
2.72.3.4 Implementation details 693
2.72.3.5 Code Example 694
2.72.4 Function MLIB_Div_F16 694
2.72.4.1 Declaration 694
2.72.4.2 Arguments 694
2.72.4.3 Return 694
2.72.4.4 Implementation details 694
2.72.4.5 Code Example 695
2.72.5 Function MLIB_Div_FLT 695
2.72.5.1 Declaration 695
2.72.5.2 Arguments 695
2.72.5.3 Return 695
2.72.5.4 Implementation details 695
2.72.5.5 Code Example 696
2.73 Function MLIB_DivSat 696
2.73.1 Description 696
2.73.2 Re-entrancy 696
2.73.3 Function MLIB_DivSat_F32 696
2.73.3.1 Declaration 696
2.73.3.2 Arguments 697
2.73.3.3 Return 697
2.73.3.4 Implementation details 697
2.73.3.5 Code Example 697
2.73.4 Function MLIB_DivSatAccurate_F32 698
2.73.4.1 Declaration 698
2.73.4.2 Arguments 698
2.73.4.3 Return 698
2.73.4.4 Implementation details 698
2.73.4.5 Code Example 698
2.73.5 Function MLIB_DivSat_F16 699
2.73.5.1 Declaration 699
2.73.5.2 Arguments 699
2.73.5.3 Return 699
2.73.5.4 Implementation details 699
2.73.5.5 Code Example 699
2.74 Function MLIB_Mac 700
2.74.1 Description 700
2.74.2 Re-entrancy 700
2.74.3 Function MLIB_Mac_F32 700
2.74.3.1 Declaration 700
2.74.3.2 Arguments 700
2.74.3.3 Return 700
2.74.3.4 Implementation details 700
2.74.3.5 Code Example 701
2.74.4 Function MLIB_Mac_F32F16F16 701
2.74.4.1 Declaration 701
2.74.4.2 Arguments 701
2.74.4.3 Return 701
2.74.4.4 Implementation details 702
2.74.4.5 Code Example 702
2.74.5 Function MLIB_Mac_F16 702
2.74.5.1 Declaration 702
2.74.5.2 Arguments 703
2.74.5.3 Return 703
2.74.5.4 Implementation details 703
2.74.5.5 Code Example 703
2.74.6 Function MLIB_Mac_FLT 704
2.74.6.1 Declaration 704
2.74.6.2 Arguments 704
2.74.6.3 Return 704
2.74.6.4 Implementation details 704
2.74.6.5 Code Example 704
2.75 Function MLIB_MacSat 705
2.75.1 Description 705
2.75.2 Re-entrancy 705
2.75.3 Function MLIB_MacSat_F32 705
2.75.3.1 Declaration 705
2.75.3.2 Arguments 705
2.75.3.3 Return 705
2.75.3.4 Implementation details 705
2.75.3.5 Code Example 706
2.75.4 Function MLIB_MacSat_F32F16F16 706
2.75.4.1 Declaration 706
2.75.4.2 Arguments 706
2.75.4.3 Return 707
2.75.4.4 Implementation details 707
2.75.4.5 Code Example 707
2.75.5 Function MLIB_MacSat_F16 707
2.75.5.1 Declaration 707
2.75.5.2 Arguments 708
2.75.5.3 Return 708
2.75.5.4 Implementation details 708
2.75.5.5 Code Example 708
2.76 Function MLIB_Mnac 709
2.76.1 Description 709

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.76.2 Re-entrancy ... 709
2.76.3 Function MLIB_Mnac_F32 ... 709
2.76.3.1 Declaration ... 709
2.76.3.2 Arguments ... 709
2.76.3.3 Return ... 709
2.76.3.4 Implementation details ... 709
2.76.3.5 Code Example ... 709
2.76.4 Function MLIB_Mnac_F32F16F16 ... 710
2.76.4.1 Declaration ... 710
2.76.4.2 Arguments ... 710
2.76.4.3 Return ... 710
2.76.4.4 Implementation details ... 710
2.76.4.5 Code Example ... 711
2.76.5 Function MLIB_Mnac_F16 ... 711
2.76.5.1 Declaration ... 711
2.76.5.2 Arguments ... 711
2.76.5.3 Return ... 711
2.76.5.4 Implementation details ... 712
2.76.5.5 Code Example ... 712
2.76.6 Function MLIB_Mnac_FLT ... 712
2.76.6.1 Declaration ... 712
2.76.6.2 Arguments ... 713
2.76.6.3 Return ... 713
2.76.6.4 Implementation details ... 713
2.76.6.5 Code Example ... 713
2.77 Function MLIB_Msu ... 714
2.77.1 Description ... 714
2.77.2 Re-entrancy ... 714
2.77.3 Function MLIB_Msu_F32 ... 714
2.77.3.1 Declaration ... 714
2.77.3.2 Arguments ... 714
2.77.3.3 Return ... 714
2.77.3.4 Implementation details ... 714
2.77.3.5 Code Example ... 715
2.77.4 Function MLIB_Msu_F32F16F16 ... 715
2.77.4.1 Declaration ... 715
2.77.4.2 Arguments ... 715
2.77.4.3 Return ... 715
2.77.4.4 Implementation details ... 716
2.77.4.5 Code Example ... 716
2.77.5 Function MLIB_Msu_F16 ... 716
2.77.5.1 Declaration ... 716
2.77.5.2 Arguments ... 717
2.77.5.3 Return ... 717
2.77.5.4 Implementation details ... 717
2.77.5.5 Code Example ... 717
2.77.6 Function MLIB_Msu_FLT ... 718
2.77.6.1 Declaration ... 718
2.77.6.2 Arguments ... 718
2.77.6.3 Return ... 718
2.77.6.4 Implementation details ... 718
2.77.6.5 Code Example ... 718
2.78 Function MLIB_Mul ... 719
2.78.1 Description ... 719
2.78.2 Re-entrancy ... 719
2.78.3 Function MLIB_Mul_F32 ... 719
2.78.3.1 Declaration ... 719
2.78.3.2 Arguments ... 719
2.78.3.3 Return ... 719
2.78.3.4 Implementation details ... 720
2.78.3.5 Code Example ... 720
2.78.4 Function MLIB_Mul_F32F16F16 ... 720
2.78.4.1 Declaration ... 720
2.78.4.2 Arguments ... 721
2.78.4.3 Return ... 721
2.78.4.4 Implementation details ... 721
2.78.4.5 Code Example ... 721
2.78.5 Function MLIB_Mul_F16 ... 722
2.78.5.1 Declaration ... 722
2.78.5.2 Arguments ... 722
2.78.5.3 Return ... 722
2.78.5.4 Implementation details ... 722
2.78.5.5 Code Example ... 722
2.78.6 Function MLIB_Mul_FLT ... 723
2.78.6.1 Declaration ... 723
2.78.6.2 Arguments ... 723
2.78.6.3 Return ... 723
2.78.6.4 Implementation details ... 723
2.78.6.5 Code Example ... 723
2.79 Function MLIB_MulSat ... 724
2.79.1 Description ... 724
2.79.2 Re-entrancy ... 724
2.79.3 Function MLIB_MulSat_F32 ... 724
2.79.3.1 Declaration ... 724
2.79.3.2 Arguments ... 724
2.79.3.3 Return ... 724
2.79.3.4 Implementation details ... 724
2.79.3.5 Code Example ... 725
2.79.4 Function MLIB_MulSat_F32F16F16 ... 725
2.79.4.1 Declaration ... 725
2.79.4.2 Arguments ... 725
2.79.4.3 Return ... 726
2.79.4.4 Implementation details ... 726
2.79.4.5 Code Example ... 726
2.79.5 Function MLIB_MulSat_F16 ... 726
2.79.5.1 Declaration ... 726
2.79.5.2 Arguments ... 726
2.79.5.3 Return ... 727
2.79.5.4 Implementation details ... 727
2.79.5.5 Code Example ... 727
2.80 Function MLIB_Neg ... 727
2.80.1 Description ... 728
2.80.2 Re-entrancy ... 728
2.80.3 Function MLIB_Neg_F32 ... 728
2.80.3.1 Declaration ... 728
2.80.3.2 Arguments ... 728
2.80.3.3 Return ... 728
2.80.3.4 Implementation details ... 728
2.80.3.5 Code Example ... 728
2.80.4 Function MLIB_Neg_F16 ... 729
2.80.4.1 Declaration ... 729
2.80.4.2 Arguments ... 729
2.80.4.3 Return ... 729
2.80.4.4 Implementation details ... 729
2.80.4.5 Code Example ... 729
2.80.5 Function MLIB_Neg_FLT ... 730
2.80.5.1 Declaration ... 730
2.80.5.2 Arguments ... 730

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.80.5.3 Return ... 730
2.80.5.4 Implementation details ... 730
2.80.5.5 Code Example ... 730
2.81 Function MLIB_NegSat ... 731
2.81.1 Description ... 731
2.81.2 Re-entrancy ... 731
2.81.3 Function MLIB_NegSat_F32 ... 731
2.81.3.1 Declaration ... 731
2.81.3.2 Arguments ... 731
2.81.3.3 Return ... 731
2.81.3.4 Implementation details ... 731
2.81.3.5 Code Example ... 732
2.81.4 Function MLIB_NegSat_F16 ... 732
2.81.4.1 Declaration ... 732
2.81.4.2 Arguments ... 732
2.81.4.3 Return ... 732
2.81.4.4 Implementation details ... 732
2.81.4.5 Code Example ... 733
2.82 Function MLIB_Norm ... 733
2.82.1 Description ... 733
2.82.2 Re-entrancy ... 733
2.82.3 Function MLIB_Norm_F32 ... 733
2.82.3.1 Declaration ... 733
2.82.3.2 Arguments ... 734
2.82.3.3 Return ... 734
2.82.3.4 Code Example ... 734
2.82.4 Function MLIB_Norm_F16 ... 734
2.82.4.1 Declaration ... 734
2.82.4.2 Arguments ... 734
2.82.4.3 Return ... 734
2.82.4.4 Code Example ... 735
2.83 Function MLIB_RndSat_F16F32 ... 735
2.83.1 Declaration ... 735
2.83.2 Arguments ... 735
2.83.3 Return ... 735
2.83.4 Description ... 735
2.83.5 Re-entrancy ... 735
2.83.6 Code Example ... 736
2.84 Function MLIB_Round ... 736
2.84.1 Description ... 736
2.84.2 Re-entrancy ... 736
2.84.3 Function MLIB_Round_F32 ... 736
2.84.3.1 Declaration ... 736
2.84.3.2 Arguments ... 736
2.84.3.3 Return ... 736
2.84.3.4 Code Example ... 737
2.84.4 Function MLIB_Round_F16 ... 737
2.84.4.1 Declaration ... 737
2.84.4.2 Arguments ... 738
2.84.4.3 Return ... 738
2.84.4.4 Code Example ... 738
2.85 Function MLIB_ShBi ... 739
2.85.1 Description ... 739
2.85.2 Re-entrancy ... 739
2.85.3 Function MLIB_ShBi_F32 ... 739
2.85.3.1 Declaration ... 739
2.85.3.2 Arguments ... 739
2.85.3.3 Return ... 739
2.85.3.4 Code Example ... 739
2.85.4 Function MLIB_ShBi_F16 ... 740
2.85.4.1 Declaration ... 740
2.85.4.2 Arguments ... 740
2.85.4.3 Return ... 740
2.85.4.4 Code Example ... 740
2.86 Function MLIB_ShBiSat ... 741
2.86.1 Description ... 741
2.86.2 Re-entrancy ... 741
2.86.3 Function MLIB_ShBiSat_F32 ... 741
2.86.3.1 Declaration ... 741
2.86.3.2 Arguments ... 741
2.86.3.3 Return ... 741
2.86.3.4 Code Example ... 742
2.86.4 Function MLIB_ShBiSat_F16 ... 742
2.86.4.1 Declaration ... 742
2.86.4.2 Arguments ... 742
2.86.4.3 Return ... 742
2.86.4.4 Code Example ... 742
2.87 Function MLIB_ShL ... 743
2.87.1 Description ... 743
2.87.2 Re-entrancy ... 743
2.87.3 Function MLIB_ShL_F32 ... 743
2.87.3.1 Declaration ... 743
2.87.3.2 Arguments ... 743
2.87.3.3 Return ... 744
2.87.3.4 Code Example ... 744
2.87.4 Function MLIB_ShL_F16 ... 744
2.87.4.1 Declaration ... 744
2.87.4.2 Arguments ... 744
2.87.4.3 Return ... 744
2.87.4.4 Code Example ... 745
2.88 Function MLIB_ShLSat ... 745
2.88.1 Description ... 745
2.88.2 Re-entrancy ... 745
2.88.3 Function MLIB_ShLSat_F32 ... 745
2.88.3.1 Declaration ... 745
2.88.3.2 Arguments ... 746
2.88.3.3 Return ... 746
2.88.3.4 Code Example ... 746
2.88.4 Function MLIB_ShLSat_F16 ... 746
2.88.4.1 Declaration ... 746
2.88.4.2 Arguments ... 747
2.88.4.3 Return ... 747
2.88.4.4 Code Example ... 747
2.89 Function MLIB_ShR ... 747
2.89.1 Description ... 747
2.89.2 Re-entrancy ... 748
2.89.3 Function MLIB_ShR_F32 ... 748
2.89.3.1 Declaration ... 748
2.89.3.2 Arguments ... 748
2.89.3.3 Return ... 748
2.89.3.4 Code Example ... 748
2.89.4 Function MLIB_ShR_F16 ... 749
2.89.4.1 Declaration ... 749
2.89.4.2 Arguments ... 749
2.89.4.3 Return ... 749
2.89.4.4 Code Example ... 749
2.90 Function MLIB_Sub ... 749
2.90.1 Description ... 750

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.90.2 Re-entrancy 750
2.90.3 Function MLIB_Sub_F32 750
2.90.3.1 Declaration 750
2.90.3.2 Arguments 750
2.90.3.3 Return 750
2.90.3.4 Implementation details 750
2.90.3.5 Code Example 750
2.90.4 Function MLIB_Sub_F16 751
2.90.4.1 Declaration 751
2.90.4.2 Arguments 751
2.90.4.3 Return 751
2.90.4.4 Implementation details 751
2.90.4.5 Code Example 751
2.90.5 Function MLIB_Sub_FLT 752
2.90.5.1 Declaration 752
2.90.5.2 Arguments 752
2.90.5.3 Return 752
2.90.5.4 Implementation details 752
2.90.5.5 Code Example 753
2.91 Function MLIB_SubSat 753
2.91.1 Description 753
2.91.2 Re-entrancy 753
2.91.3 Function MLIB_SubSat_F32 753
2.91.3.1 Declaration 753
2.91.3.2 Arguments 754
2.91.3.3 Return 754
2.91.3.4 Implementation details 754
2.91.3.5 Code Example 754
2.91.4 Function MLIB_SubSat_F16 755
2.91.4.1 Declaration 755
2.91.4.2 Arguments 755
2.91.4.3 Return 755
2.91.4.4 Implementation details 755
2.91.4.5 Code Example 755
2.92 Function MLIB_VAdd 756
2.92.1 Description 756
2.92.2 Re-entrancy 756
2.92.3 Function MLIB_VAdd_F32 756
2.92.3.1 Declaration 756
2.92.3.2 Arguments 756
2.92.3.3 Implementation details 757
2.92.3.4 Code Example: 757
2.92.4 Function MLIB_VAdd_F16 757
2.92.4.1 Declaration 757
2.92.4.2 Arguments 757
2.92.4.3 Implementation details 758
2.92.4.4 Code Example: 758
2.92.5 Function MLIB_VAdd_FLT 758
2.92.5.1 Declaration 758
2.92.5.2 Arguments 758
2.92.5.3 Implementation details 759
2.92.5.4 Code Example 759
2.93 Function MLIB_VMac 759
2.93.1 Description 759
2.93.2 Re-entrancy 759
2.93.3 Function MLIB_VMac_F32 760
2.93.3.1 Declaration 760
2.93.3.2 Arguments 760
2.93.3.3 Return 760
2.93.3.4 Implementation details 760
2.93.3.5 Code Example 760
2.93.4 Function MLIB_VMac_F32F16F16 761
2.93.4.1 Declaration 761
2.93.4.2 Arguments 761
2.93.4.3 Return 761
2.93.4.4 Implementation details 761
2.93.4.5 Code Example 762
2.93.5 Function MLIB_VMac_F16 762
2.93.5.1 Declaration 762
2.93.5.2 Arguments 762
2.93.5.3 Return 762
2.93.5.4 Implementation details 762
2.93.5.5 Code Example 763
2.93.6 Function MLIB_VMac_FLT 763
2.93.6.1 Declaration 763
2.93.6.2 Arguments 764
2.93.6.3 Return 764
2.93.6.4 Implementation details 764
2.93.6.5 Code Example 764
2.94 Function MLIB_VScale 765
2.94.1 Description 765
2.94.2 Re-entrancy 765
2.94.3 Function MLIB_VScale_F32 765
2.94.3.1 Declaration 765
2.94.3.2 Arguments 765
2.94.3.3 Implementation details 766
2.94.3.4 Code Example: 766
2.94.4 Function MLIB_VScale_F16 766
2.94.4.1 Declaration 766
2.94.4.2 Arguments 766
2.94.4.3 Implementation details 767
2.94.4.4 Code Example: 767
2.94.5 Function MLIB_VScale_FLT 767
2.94.5.1 Declaration 767
2.94.5.2 Arguments 767
2.94.5.3 Implementation details 768
2.94.5.4 Code Example 768
2.95 Function MLIB_VSub 768
2.95.1 Description 768
2.95.2 Re-entrancy 769
2.95.3 Function MLIB_VSub_F32 769
2.95.3.1 Declaration 769
2.95.3.2 Arguments 769
2.95.3.3 Implementation details 769
2.95.3.4 Code Example: 769
2.95.4 Function MLIB_VSub_F16 770
2.95.4.1 Declaration 770
2.95.4.2 Arguments 770
2.95.4.3 Implementation details 770
2.95.4.4 Code Example: 770
2.95.5 Function MLIB_VSub_FLT 771
2.95.5.1 Declaration 771
2.95.5.2 Arguments 771
2.95.5.3 Implementation details 771
2.95.5.4 Code Example 771
2.96 Function SWLIBS_GetVersion 772
2.96.1 Declaration 772
2.96.2 Return 772

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

2.96.3 Description ... 772
2.96.4 Reentrancy ... 772
3 Typedefs ... 772
3.1 Typedef index ... 772
4 Enums ... 773
4.1 AMCLIB_ESTIMBJINIT_RET_T ... 773
4.1.1 Source File ... 773
4.1.2 Enumerated Type Members ... 773
4.2 AMCLIB_ESTIMBJ_RET_T ... 773
4.2.1 Source File ... 773
4.2.2 Enumerated Type Members ... 773
4.3 AMCLIB_ESTIMBJ_STATE_T ... 774
4.3.1 Source File ... 774
4.3.2 Enumerated Type Members ... 774
4.4 AMCLIB_ESTIMRLINIT_RET_T ... 774
4.4.1 Source File ... 774
4.4.2 Enumerated Type Members ... 774
4.5 AMCLIB_ESTIMRL_RET_T ... 774
4.5.1 Source File ... 774
4.5.2 Enumerated Type Members ... 775
4.6 AMCLIB_ESTIMRL_STATE_T ... 775
4.6.1 Source File ... 775
4.6.2 Enumerated Type Members ... 775
4.7 AMCLIB_WINDMILLING_RET_T ... 775
4.7.1 Source File ... 775
4.7.2 Enumerated Type Members ... 776
4.8 tBool ... 776
4.8.1 Source File ... 776
4.8.2 Enumerated Type Members ... 776
5 Compound data types ... 776
5.1 AMCLIB_BEMF_OBSRV_DQ_T_F16 ... 776
5.1.1 Source File ... 776
5.1.2 Compound Type Members ... 776
5.2 AMCLIB_BEMF_OBSRV_DQ_T_F32 ... 777
5.2.1 Source File ... 777
5.2.2 Compound Type Members ... 777
5.3 AMCLIB_BEMF_OBSRV_DQ_T_FLT ... 778
5.3.1 Source File ... 778
5.3.2 Compound Type Members ... 778
5.4 AMCLIB_CURRENT_LOOP_PIP_T_F16 ... 779
5.4.1 Source File ... 779
5.4.2 Compound Type Members ... 779
5.5 AMCLIB_CURRENT_LOOP_PIP_T_F32 ... 779
5.5.1 Source File ... 779
5.5.2 Compound Type Members ... 779
5.6 AMCLIB_CURRENT_LOOP_PIP_T_FLT ... 780
5.6.1 Source File ... 780
5.6.2 Compound Type Members ... 780
5.7 AMCLIB_CURRENT_LOOP_T_F16 ... 780
5.7.1 Source File ... 780
5.7.2 Compound Type Members ... 780
5.8 AMCLIB_CURRENT_LOOP_T_F32 ... 781
5.8.1 Source File ... 781
5.8.2 Compound Type Members ... 781
5.9 AMCLIB_CURRENT_LOOP_T_FLT ... 781
5.9.1 Source File ... 781
5.9.2 Compound Type Members ... 781
5.10 AMCLIB_ESTIMBJ_INIT_T_FLT ... 781
5.10.1 Source File ... 782
5.10.2 Compound Type Members ... 782
5.11 AMCLIB_ESTIMBJ_INNERST_SIM_T_FLT ... 782
5.11.1 Source File ... 782
5.11.2 Compound Type Members ... 783
5.12 AMCLIB_ESTIMBJ_INNERST_T_FLT ... 787
5.12.1 Source File ... 787
5.12.2 Compound Type Members ... 787
5.13 AMCLIB_ESTIMBJ_T_FLT ... 790
5.13.1 Source File ... 790
5.13.2 Compound Type Members ... 790
5.14 AMCLIB_ESTIMRL_INIT_T_FLT ... 791
5.14.1 Source File ... 791
5.14.2 Compound Type Members ... 791
5.15 AMCLIB_ESTIMRL_INNERST_T_FLT ... 791
5.15.1 Source File ... 791
5.15.2 Compound Type Members ... 792
5.16 AMCLIB_ESTIMRL_RUN_T_FLT ... 795
5.16.1 Source File ... 795
5.16.2 Compound Type Members ... 795
5.17 AMCLIB_ESTIMRL_T_FLT ... 795
5.17.1 Source File ... 795
5.17.2 Compound Type Members ... 796
5.18 AMCLIB_FLUXESTIMAB_T_F16 ... 796
5.18.1 Source File ... 796
5.18.2 Compound Type Members ... 796
5.19 AMCLIB_FLUXESTIMAB_T_F32 ... 796
5.19.1 Source File ... 796
5.19.2 Compound Type Members ... 796
5.20 AMCLIB_FLUXESTIMAB_T_FLT ... 797
5.20.1 Source File ... 797
5.20.2 Compound Type Members ... 797
5.21 AMCLIB_FW_DEBUG_T_F16 ... 797
5.21.1 Source File ... 797
5.21.2 Compound Type Members ... 797
5.22 AMCLIB_FW_DEBUG_T_F32 ... 798
5.22.1 Source File ... 798
5.22.2 Compound Type Members ... 798
5.23 AMCLIB_FW_DEBUG_T_FLT ... 799
5.23.1 Source File ... 799
5.23.2 Compound Type Members ... 799
5.24 AMCLIB_FW_SPEED_LOOP_DEBUG_T_F16 ... 800
5.24.1 Source File ... 800
5.24.2 Compound Type Members ... 800
5.25 AMCLIB_FW_SPEED_LOOP_DEBUG_T_F32 ... 801
5.25.1 Source File ... 801
5.25.2 Compound Type Members ... 801
5.26 AMCLIB_FW_SPEED_LOOP_DEBUG_T_FLT ... 802
5.26.1 Source File ... 802
5.26.2 Compound Type Members ... 802
5.27 AMCLIB_FW_SPEED_LOOP_T_F16 ... 803
5.27.1 Source File ... 804
5.27.2 Compound Type Members ... 804
5.28 AMCLIB_FW_SPEED_LOOP_T_F32 ... 804
5.28.1 Source File ... 804
5.28.2 Compound Type Members ... 804
5.29 AMCLIB_FW_SPEED_LOOP_T_FLT ... 805

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.29.1 Source File 805
5.29.2 Compound Type Members 805
5.30 AMCLIB_FW_T_F16 806
5.30.1 Source File 806
5.30.2 Compound Type Members 806
5.31 AMCLIB_FW_T_F32 806
5.31.1 Source File 806
5.31.2 Compound Type Members 807
5.32 AMCLIB_FW_T_FLT 807
5.32.1 Source File 807
5.32.2 Compound Type Members 807
5.33 AMCLIB_SPEED_LOOP_DEBUG_T_F16 808
5.33.1 Source File 808
5.33.2 Compound Type Members 808
5.34 AMCLIB_SPEED_LOOP_DEBUG_T_F32 808
5.34.1 Source File 808
5.34.2 Compound Type Members 808
5.35 AMCLIB_SPEED_LOOP_DEBUG_T_FLT 809
5.35.1 Source File 809
5.35.2 Compound Type Members 809
5.36 AMCLIB_SPEED_LOOP_T_F16 809
5.36.1 Source File 809
5.36.2 Compound Type Members 809
5.37 AMCLIB_SPEED_LOOP_T_F32 810
5.37.1 Source File 810
5.37.2 Compound Type Members 810
5.38 AMCLIB_SPEED_LOOP_T_FLT 810
5.38.1 Source File 810
5.38.2 Compound Type Members 810
5.39 AMCLIB_TRACK_OBSRV_T_F16 810
5.39.1 Source File 811
5.39.2 Compound Type Members 811
5.40 AMCLIB_TRACK_OBSRV_T_F32 811
5.40.1 Source File 811
5.40.2 Compound Type Members 811
5.41 AMCLIB_TRACK_OBSRV_T_FLT 811
5.41.1 Source File 811
5.41.2 Compound Type Members 811
5.42 AMCLIB_WINDMILLING_T_F16 812
5.42.1 Source File 812
5.42.2 Compound Type Members 812
5.43 AMCLIB_WINDMILLING_T_F32 812
5.43.1 Source File 813
5.43.2 Compound Type Members 813
5.44 AMCLIB_WINDMILLING_T_FLT 813
5.44.1 Source File 813
5.44.2 Compound Type Members 814
5.45 GDFLIB_FILTER_IIR1_COEFF_T_F16 814
5.45.1 Source File 814
5.45.2 Compound Type Members 814
5.46 GDFLIB_FILTER_IIR1_COEFF_T_F32 815
5.46.1 Source File 815
5.46.2 Compound Type Members 815
5.47 GDFLIB_FILTER_IIR1_COEFF_T_FLT 815
5.47.1 Source File 815
5.47.2 Compound Type Members 815
5.48 GDFLIB_FILTER_IIR1_T_F16 816
5.48.1 Source File 816
5.48.2 Compound Type Members 816
5.49 GDFLIB_FILTER_IIR1_T_F32 816
5.49.1 Source File 816
5.49.2 Compound Type Members 816
5.50 GDFLIB_FILTER_IIR1_T_FLT 816
5.50.1 Source File 817
5.50.2 Compound Type Members 817
5.51 GDFLIB_FILTER_IIR2_COEFF_T_F16 817
5.51.1 Source File 817
5.51.2 Compound Type Members 817
5.52 GDFLIB_FILTER_IIR2_COEFF_T_F32 817
5.52.1 Source File 818
5.52.2 Compound Type Members 818
5.53 GDFLIB_FILTER_IIR2_COEFF_T_FLT 818
5.53.1 Source File 818
5.53.2 Compound Type Members 818
5.54 GDFLIB_FILTER_IIR2_T_F16 819
5.54.1 Source File 819
5.54.2 Compound Type Members 819
5.55 GDFLIB_FILTER_IIR2_T_F32 819
5.55.1 Source File 819
5.55.2 Compound Type Members 819
5.56 GDFLIB_FILTER_IIR2_T_FLT 819
5.56.1 Source File 819
5.56.2 Compound Type Members 820
5.57 GDFLIB_FILTER_MA_T_F16 820
5.57.1 Source File 820
5.57.2 Compound Type Members 820
5.58 GDFLIB_FILTER_MA_T_F32 820
5.58.1 Source File 820
5.58.2 Compound Type Members 820
5.59 GDFLIB_FILTER_MA_T_FLT 820
5.59.1 Source File 821
5.59.2 Compound Type Members 821
5.60 GDFLIB_FILTERFIR_PARAM_T_F16 821
5.60.1 Source File 821
5.60.2 Compound Type Members 821
5.61 GDFLIB_FILTERFIR_PARAM_T_F32 821
5.61.1 Source File 821
5.61.2 Compound Type Members 821
5.62 GDFLIB_FILTERFIR_PARAM_T_FLT 822
5.62.1 Source File 822
5.62.2 Compound Type Members 822
5.63 GDFLIB_FILTERFIR_STATE_T_F16 822
5.63.1 Source File 822
5.63.2 Compound Type Members 822
5.64 GDFLIB_FILTERFIR_STATE_T_F32 822
5.64.1 Source File 822
5.64.2 Compound Type Members 822
5.65 GDFLIB_FILTERFIR_STATE_T_FLT 823
5.65.1 Source File 823
5.65.2 Compound Type Members 823
5.66 GFLIB_ACOS_T_F16 823
5.66.1 Source File 823
5.66.2 Description 823
5.66.3 Compound Type Members 823
5.67 GFLIB_ACOS_T_F32 823
5.67.1 Source File 824
5.67.2 Description 824
5.67.3 Compound Type Members 824

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.68 GFLIB_ACOS_T_FLT 824
5.68.1 Source File 824
5.68.2 Description 824
5.68.3 Compound Type Members 824
5.69 GFLIB_ACOS_TAYLOR_COEF_T_F16 824
5.69.1 Source File 824
5.69.2 Description 825
5.69.3 Compound Type Members 825
5.70 GFLIB_ACOS_TAYLOR_COEF_T_F32 825
5.70.1 Source File 825
5.70.2 Description 825
5.70.3 Compound Type Members 825
5.71 GFLIB_ASIN_T_F16 825
5.71.1 Source File 825
5.71.2 Description 825
5.71.3 Compound Type Members 826
5.72 GFLIB_ASIN_T_F32 826
5.72.1 Source File 826
5.72.2 Description 826
5.72.3 Compound Type Members 826
5.73 GFLIB_ASIN_T_FLT 826
5.73.1 Source File 826
5.73.2 Description 826
5.73.3 Compound Type Members 826
5.74 GFLIB_ASIN_TAYLOR_COEF_T_F16 827
5.74.1 Source File 827
5.74.2 Description 827
5.74.3 Compound Type Members 827
5.75 GFLIB_ASIN_TAYLOR_COEF_T_F32 827
5.75.1 Source File 827
5.75.2 Description 827
5.75.3 Compound Type Members 827
5.76 GFLIB_ATAN_T_F16 827
5.76.1 Source File 827
5.76.2 Description 828
5.76.3 Compound Type Members 828
5.77 GFLIB_ATAN_T_F32 828
5.77.1 Source File 828
5.77.2 Description 828
5.77.3 Compound Type Members 828
5.78 GFLIB_ATAN_T_FLT 828
5.78.1 Source File 828
5.78.2 Description 828
5.78.3 Compound Type Members 829
5.79 GFLIB_ATAN_TAYLOR_COEF_T_F16 829
5.79.1 Source File 829
5.79.2 Description 829
5.79.3 Compound Type Members 829
5.80 GFLIB_ATAN_TAYLOR_COEF_T_F32 829
5.80.1 Source File 829
5.80.2 Description 829
5.80.3 Compound Type Members 830
5.81 GFLIB_ATANYXSHIFTED_T_F16 830
5.81.1 Source File 830
5.81.2 Compound Type Members 830
5.82 GFLIB_ATANYXSHIFTED_T_F32 830
5.82.1 Source File 830
5.82.2 Compound Type Members 830
5.83 GFLIB_ATANYXSHIFTED_T_FLT 831
5.83.1 Source File 831
5.83.2 Compound Type Members 831
5.84 GFLIB_CONTROLLER_PI_P_T_F16 831
5.84.1 Source File 831
5.84.2 Compound Type Members 831
5.85 GFLIB_CONTROLLER_PI_P_T_F32 831
5.85.1 Source File 831
5.85.2 Compound Type Members 832
5.86 GFLIB_CONTROLLER_PI_P_T_FLT 832
5.86.1 Source File 832
5.86.2 Compound Type Members 832
5.87 GFLIB_CONTROLLER_PI_R_T_F16 832
5.87.1 Source File 832
5.87.2 Compound Type Members 833
5.88 GFLIB_CONTROLLER_PI_R_T_F32 833
5.88.1 Source File 833
5.88.2 Compound Type Members 833
5.89 GFLIB_CONTROLLER_PI_R_T_FLT 833
5.89.1 Source File 833
5.89.2 Compound Type Members 834
5.90 GFLIB_CONTROLLER_PIAW_P_T_F16 834
5.90.1 Source File 834
5.90.2 Compound Type Members 834
5.91 GFLIB_CONTROLLER_PIAW_P_T_F32 835
5.91.1 Source File 835
5.91.2 Compound Type Members 835
5.92 GFLIB_CONTROLLER_PIAW_P_T_FLT 835
5.92.1 Source File 835
5.92.2 Compound Type Members 835
5.93 GFLIB_CONTROLLER_PIAW_R_T_F16 836
5.93.1 Source File 836
5.93.2 Compound Type Members 836
5.94 GFLIB_CONTROLLER_PIAW_R_T_F32 837
5.94.1 Source File 837
5.94.2 Compound Type Members 837
5.95 GFLIB_CONTROLLER_PIAW_R_T_FLT 837
5.95.1 Source File 837
5.95.2 Compound Type Members 837
5.96 GFLIB_CONTROLLER_PID_P_AW_T_F16 ..838
5.96.1 Source File 838
5.96.2 Compound Type Members 838
5.97 GFLIB_CONTROLLER_PID_P_AW_T_F32 ..839
5.97.1 Source File 839
5.97.2 Compound Type Members 839
5.98 GFLIB_CONTROLLER_PID_P_AW_T_FLT ..840
5.98.1 Source File 840
5.98.2 Compound Type Members 840
5.99 GFLIB_COS_T_F16 840
5.99.1 Source File 840
5.99.2 Compound Type Members 841
5.100 GFLIB_COS_T_F32 841
5.100.1 Source File 841
5.100.2 Compound Type Members 841
5.101 GFLIB_COS_T_FLT 841
5.101.1 Source File 841
5.101.2 Compound Type Members 841
5.102 GFLIB_HYST_T_F16 841
5.102.1 Source File 841
5.102.2 Compound Type Members 842

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.103 GFLIB_HYST_T_F32 842
5.103.1 Source File 842
5.103.2 Compound Type Members 842
5.104 GFLIB_HYST_T_FLT 842
5.104.1 Source File 842
5.104.2 Compound Type Members 843
5.105 GFLIB_INTEGRATOR_TR_T_F16 843
5.105.1 Source File 843
5.105.2 Compound Type Members 843
5.106 GFLIB_INTEGRATOR_TR_T_F32 843
5.106.1 Source File 843
5.106.2 Compound Type Members 843
5.107 GFLIB_INTEGRATOR_TR_T_FLT 844
5.107.1 Source File 844
5.107.2 Compound Type Members 844
5.108 GFLIB_LIMIT_T_F16 844
5.108.1 Source File 844
5.108.2 Compound Type Members 844
5.109 GFLIB_LIMIT_T_F32 844
5.109.1 Source File 845
5.109.2 Compound Type Members 845
5.110 GFLIB_LIMIT_T_FLT 845
5.110.1 Source File 845
5.110.2 Compound Type Members 845
5.111 GFLIB_LOG10_T_FLT 845
5.111.1 Source File 845
5.111.2 Compound Type Members 845
5.112 GFLIB_LOWERLIMIT_T_F16 845
5.112.1 Source File 846
5.112.2 Compound Type Members 846
5.113 GFLIB_LOWERLIMIT_T_F32 846
5.113.1 Source File 846
5.113.2 Compound Type Members 846
5.114 GFLIB_LOWERLIMIT_T_FLT 846
5.114.1 Source File 846
5.114.2 Compound Type Members 846
5.115 GFLIB_LUT1D_T_F16 846
5.115.1 Source File 846
5.115.2 Compound Type Members 847
5.116 GFLIB_LUT1D_T_F32 847
5.116.1 Source File 847
5.116.2 Compound Type Members 847
5.117 GFLIB_LUT1D_T_FLT 847
5.117.1 Source File 847
5.117.2 Compound Type Members 848
5.118 GFLIB_LUT2D_T_F16 848
5.118.1 Source File 848
5.118.2 Compound Type Members 848
5.119 GFLIB_LUT2D_T_F32 848
5.119.1 Source File 848
5.119.2 Compound Type Members 849
5.120 GFLIB_LUT2D_T_FLT 849
5.120.1 Source File 849
5.120.2 Compound Type Members 849
5.121 GFLIB_RAMP_T_F16 850
5.121.1 Source File 850
5.121.2 Compound Type Members 850
5.122 GFLIB_RAMP_T_F32 850
5.122.1 Source File 850
5.122.2 Compound Type Members 850
5.123 GFLIB_RAMP_T_FLT 850
5.123.1 Source File 850
5.123.2 Compound Type Members 850
5.124 GFLIB_SIN_T_F16 851
5.124.1 Source File 851
5.124.2 Compound Type Members 851
5.125 GFLIB_SIN_T_F32 851
5.125.1 Source File 851
5.125.2 Compound Type Members 851
5.126 GFLIB_SIN_T_FLT 851
5.126.1 Source File 851
5.126.2 Compound Type Members 852
5.127 GFLIB_SINCOS_T_F16 852
5.127.1 Source File 852
5.127.2 Compound Type Members 852
5.128 GFLIB_SINCOS_T_F32 852
5.128.1 Source File 852
5.128.2 Compound Type Members 852
5.129 GFLIB_SINCOS_T_FLT 852
5.129.1 Source File 852
5.129.2 Compound Type Members 853
5.130 GFLIB_TAN_T_F16 853
5.130.1 Source File 853
5.130.2 Compound Type Members 853
5.131 GFLIB_TAN_T_F32 853
5.131.1 Source File 853
5.131.2 Compound Type Members 854
5.132 GFLIB_TAN_T_FLT 854
5.132.1 Source File 854
5.132.2 Compound Type Members 854
5.133 GFLIB_TAN_TAYLOR_COEF_T_F16 854
5.133.1 Source File 854
5.133.2 Description 854
5.133.3 Compound Type Members 854
5.134 GFLIB_TAN_TAYLOR_COEF_T_F32 855
5.134.1 Source File 855
5.134.2 Description 855
5.134.3 Compound Type Members 855
5.135 GFLIB_UPPERLIMIT_T_F16 855
5.135.1 Source File 855
5.135.2 Compound Type Members 855
5.136 GFLIB_UPPERLIMIT_T_F32 855
5.136.1 Source File 855
5.136.2 Compound Type Members 855
5.137 GFLIB_UPPERLIMIT_T_FLT 856
5.137.1 Source File 856
5.137.2 Compound Type Members 856
5.138 GFLIB_VECTORLIMIT_T_F16 856
5.138.1 Source File 856
5.138.2 Compound Type Members 856
5.139 GFLIB_VECTORLIMIT_T_F32 856
5.139.1 Source File 856
5.139.2 Compound Type Members 856
5.140 GFLIB_VECTORLIMIT_T_FLT 857
5.140.1 Source File 857
5.140.2 Compound Type Members 857
5.141 GFLIB_VLOG10_T_FLT 857
5.141.1 Source File 857

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

5.141.2 Compound Type Members 857
5.142 GMCLIB_CLARK_6PH_T_F16 857
5.142.1 Source File 857
5.142.2 Compound Type Members 857
5.143 GMCLIB_CLARK_6PH_T_F32 858
5.143.1 Source File 858
5.143.2 Compound Type Members 858
5.144 GMCLIB_CLARK_6PH_T_FLT 858
5.144.1 Source File 858
5.144.2 Compound Type Members 858
5.145 GMCLIB_CLARKINV_6PH_T_F16 859
5.145.1 Source File 859
5.145.2 Compound Type Members 859
5.146 GMCLIB_CLARKINV_6PH_T_F32 859
5.146.1 Source File 859
5.146.2 Compound Type Members 859
5.147 GMCLIB_CLARKINV_6PH_T_FLT 859
5.147.1 Source File 859
5.147.2 Compound Type Members 860
5.148 GMCLIB_DECOUPLINGPMSM_T_F16 860
5.148.1 Source File 860
5.148.2 Compound Type Members 860
5.149 GMCLIB_DECOUPLINGPMSM_T_F32 860
5.149.1 Source File 860
5.149.2 Compound Type Members 860
5.150 GMCLIB_DECOUPLINGPMSM_T_FLT 861
5.150.1 Source File 861
5.150.2 Compound Type Members 861
5.151 GMCLIB_DOUBLESWITCHING_3PH_ EDGES_T_F16 861
5.151.1 Source File 861
5.151.2 Compound Type Members 861
5.152 GMCLIB_DOUBLESWITCHING_EDGES_ T_F16 861
5.152.1 Source File 861
5.152.2 Compound Type Members 861
5.153 GMCLIB_DOUBLESWITCHING_T_F16 862
5.153.1 Source File 862
5.153.2 Compound Type Members 862
5.154 GMCLIB_DOUBLESWITCHING_ TRIGGERS_T_F16 862
5.154.1 Source File 862
5.154.2 Compound Type Members 862
5.155 GMCLIB_ELIMDCBUSRIP_T_F16 863
5.155.1 Source File 863
5.155.2 Compound Type Members 863
5.156 GMCLIB_ELIMDCBUSRIP_T_F32 863
5.156.1 Source File 863
5.156.2 Compound Type Members 863
5.157 GMCLIB_ELIMDCBUSRIP_T_FLT 863
5.157.1 Source File 863
5.157.2 Compound Type Members 864
5.158 GMCLIB_SPREADPWM_T 864
5.158.1 Source File 864
5.158.2 Compound Type Members 864
5.159 GMCLIB_SVMSTD_6PH_T_F16 864
5.159.1 Source File 864
5.159.2 Compound Type Members 865
5.160 GMCLIB_SVMSTD_6PH_T_F32 865
5.160.1 Source File 865
5.160.2 Compound Type Members 865
5.161 GMCLIB_SVMSTD_6PH_T_FLT 865
5.161.1 Source File 865
5.161.2 Compound Type Members 865
5.162 SWLIBS_2Syst_F16 866
5.162.1 Source File 866
5.162.2 Compound Type Members 866
5.163 SWLIBS_2Syst_F32 866
5.163.1 Source File 866
5.163.2 Compound Type Members 866
5.164 SWLIBS_2Syst_FLT 866
5.164.1 Source File 866
5.164.2 Compound Type Members 866
5.165 SWLIBS_2Syst_U16 867
5.165.1 Source File 867
5.165.2 Compound Type Members 867
5.166 SWLIBS_2Syst_U32 867
5.166.1 Source File 867
5.166.2 Compound Type Members 867
5.167 SWLIBS_3Syst_F16 867
5.167.1 Source File 867
5.167.2 Compound Type Members 867
5.168 SWLIBS_3Syst_F32 868
5.168.1 Source File 868
5.168.2 Compound Type Members 868
5.169 SWLIBS_3Syst_FLT 868
5.169.1 Source File 868
5.169.2 Compound Type Members 868
5.170 SWLIBS_3Syst_U16 868
5.170.1 Source File 868
5.170.2 Compound Type Members 868
5.171 SWLIBS_6Syst_F16 869
5.171.1 Source File 869
5.171.2 Compound Type Members 869
5.172 SWLIBS_6Syst_F32 869
5.172.1 Source File 869
5.172.2 Compound Type Members 869
5.173 SWLIBS_6Syst_FLT 869
5.173.1 Source File 869
5.173.2 Compound Type Members 870
5.174 SWLIBS_VERSION_T 870
5.174.1 Source File 870
5.174.2 Compound Type Members 870
6 Macros 870
6.1 AMCLIB_BEMF_OBSRV_DQ_DEFAULT_ F32 870
6.1.1 Source File 870
6.1.2 Macro Definition 870
6.2 AMCLIB_BEMF_OBSRV_DQ_DEFAULT_ F16 871
6.2.1 Source File 871
6.2.2 Macro Definition 871
6.3 AMCLIB_BEMF_OBSRV_DQ_DEFAULT_ FLT 871
6.3.1 Source File 871
6.3.2 Macro Definition 871
6.4 AMCLIB_CURRENT_LOOP_DEFAULT_ F32 871

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.4.1 Source File 871
6.4.2 Macro Definition 871
6.5 AMCLIB_CURRENT_LOOP_DEFAULT_ F16 871
6.5.1 Source File 872
6.5.2 Macro Definition 872
6.6 AMCLIB_CURRENT_LOOP_DEFAULT_ FLT 872
6.6.1 Source File 872
6.6.2 Macro Definition 872
6.7 ESTIMBJ_INIT_DEFAULT_FLT 872
6.7.1 Source File 872
6.7.2 Macro Definition 872
6.7.3 Arguments 872
6.8 ESTIMRL_INIT_DEFAULT_FLT 873
6.8.1 Source File 873
6.8.2 Macro Definition 873
6.8.3 Arguments 873
6.9 ESTIMRL_INIT_CFG1_FLT 873
6.9.1 Source File 873
6.9.2 Macro Definition 873
6.9.3 Arguments 873
6.10 ESTIMRL_INIT_CFG2_FLT 874
6.10.1 Source File 874
6.10.2 Macro Definition 874
6.10.3 Arguments 874
6.11 ESTIMRL_INIT_CFG3_FLT 874
6.11.1 Source File 874
6.11.2 Macro Definition 874
6.12 ESTIMRL_RUN_DEFAULT_FLT 874
6.12.1 Source File 874
6.12.2 Macro Definition 875
6.13 ESTIMRL_RUN_CFG1_FLT 875
6.13.1 Source File 875
6.13.2 Macro Definition 875
6.13.3 Arguments 875
6.14 ESTIMRL_RUN_CFG2_FLT 875
6.14.1 Source File 875
6.14.2 Macro Definition 875
6.14.3 Arguments 876
6.15 ESTIMRL_RUN_CFG3_FLT 876
6.15.1 Source File 876
6.15.2 Macro Definition 876
6.15.3 Arguments 876
6.16 AMCLIB_FW_DEFAULT_F32 876
6.16.1 Source File 876
6.16.2 Macro Definition 877
6.17 AMCLIB_FW_DEFAULT_F16 877
6.17.1 Source File 877
6.17.2 Macro Definition 877
6.18 AMCLIB_FW_DEFAULT_FLT 877
6.18.1 Source File 877
6.18.2 Macro Definition 877
6.19 AMCLIB_FW_SPEED_LOOP_DEFAULT_ F32 877
6.19.1 Source File 877
6.19.2 Macro Definition 877
6.20 AMCLIB_FW_SPEED_LOOP_DEFAULT_ F16 878
6.20.1 Source File 878
6.20.2 Macro Definition 878
6.21 AMCLIB_FW_SPEED_LOOP_DEFAULT_ FLT 878
6.21.1 Source File 878
6.21.2 Macro Definition 878
6.22 AMCLIB_SPEED_LOOP_DEFAULT_F32 878
6.22.1 Source File 878
6.22.2 Macro Definition 878
6.23 AMCLIB_SPEED_LOOP_DEFAULT_F16 878
6.23.1 Source File 879
6.23.2 Macro Definition 879
6.24 AMCLIB_SPEED_LOOP_DEFAULT_FLT 879
6.24.1 Source File 879
6.24.2 Macro Definition 879
6.25 AMCLIB_TRACK_OBSRV_T 879
6.25.1 Source File 879
6.25.2 Macro Definition 879
6.26 AMCLIB_TRACK_OBSRV_DEFAULT 879
6.26.1 Source File 879
6.26.2 Macro Definition 879
6.27 AMCLIB_TRACK_OBSRV_DEFAULT_F32 ... 880
6.27.1 Source File 880
6.27.2 Macro Definition 880
6.28 AMCLIB_TRACK_OBSRV_DEFAULT_F16 ... 880
6.28.1 Source File 880
6.28.2 Macro Definition 880
6.29 AMCLIB_TRACK_OBSRV_DEFAULT_FLT ... 880
6.29.1 Source File 880
6.29.2 Macro Definition 880
6.30 GDFLIB_FILTERFIR_PARAM_T 880
6.30.1 Source File 880
6.30.2 Macro Definition 881
6.31 GDFLIB_FILTERFIR_STATE_T 881
6.31.1 Source File 881
6.31.2 Macro Definition 881
6.32 GDFLIB_FILTER_IIR1_T 881
6.32.1 Source File 881
6.32.2 Macro Definition 881
6.33 GDFLIB_FILTER_IIR1_DEFAULT 881
6.33.1 Source File 881
6.33.2 Macro Definition 881
6.34 GDFLIB_FILTER_IIR1_DEFAULT_F32 881
6.34.1 Source File 881
6.34.2 Macro Definition 882
6.35 GDFLIB_FILTER_IIR1_DEFAULT_F16 882
6.35.1 Source File 882
6.35.2 Macro Definition 882
6.36 GDFLIB_FILTER_IIR1_DEFAULT_FLT 882
6.36.1 Source File 882
6.36.2 Macro Definition 882
6.37 GDFLIB_FILTER_IIR2_T 882
6.37.1 Source File 882
6.37.2 Macro Definition 882
6.38 GDFLIB_FILTER_IIR2_DEFAULT 882
6.38.1 Source File 882
6.38.2 Macro Definition 883
6.39 GDFLIB_FILTER_IIR2_DEFAULT_F32 883
6.39.1 Source File 883

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.39.2 Macro Definition 883
6.40 GDFLIB_FILTER_IIR2_DEFAULT_F16 883
6.40.1 Source File 883
6.40.2 Macro Definition 883
6.41 GDFLIB_FILTER_IIR2_DEFAULT_FLT 883
6.41.1 Source File 883
6.41.2 Macro Definition 883
6.42 GDFLIB_FILTER_MA_T 883
6.42.1 Source File 884
6.42.2 Macro Definition 884
6.43 GDFLIB_FILTER_MA_DEFAULT 884
6.43.1 Source File 884
6.43.2 Macro Definition 884
6.44 GDFLIB_FILTER_MA_DEFAULT_F32 884
6.44.1 Source File 884
6.44.2 Macro Definition 884
6.45 GDFLIB_FILTER_MA_DEFAULT_F16 884
6.45.1 Source File 884
6.45.2 Macro Definition 884
6.46 GDFLIB_FILTER_MA_DEFAULT_FLT 884
6.46.1 Source File 884
6.46.2 Macro Definition 885
6.47 GFLIB_ACOS_T 885
6.47.1 Source File 885
6.47.2 Macro Definition 885
6.48 GFLIB_ACOS_DEFAULT 885
6.48.1 Source File 885
6.48.2 Macro Definition 885
6.49 GFLIB_ACOS_DEFAULT_F32 885
6.49.1 Source File 885
6.49.2 Macro Definition 885
6.50 GFLIB_ACOS_DEFAULT_F16 885
6.50.1 Source File 885
6.50.2 Macro Definition 885
6.51 GFLIB_ACOS_DEFAULT_FLT 886
6.51.1 Source File 886
6.51.2 Macro Definition 886
6.52 GFLIB_ASIN_FLT_MIN 886
6.52.1 Source File 886
6.52.2 Macro Definition 886
6.53 GFLIB_ASIN_FLT_INT1 886
6.53.1 Source File 886
6.53.2 Macro Definition 886
6.54 GFLIB_ASIN_T 886
6.54.1 Source File 886
6.54.2 Macro Definition 886
6.55 GFLIB_ASIN_DEFAULT 886
6.55.1 Source File 887
6.55.2 Macro Definition 887
6.56 GFLIB_ASIN_DEFAULT_F32 887
6.56.1 Source File 887
6.56.2 Macro Definition 887
6.57 GFLIB_ASIN_DEFAULT_F16 887
6.57.1 Source File 887
6.57.2 Macro Definition 887
6.58 GFLIB_ASIN_DEFAULT_FLT 887
6.58.1 Source File 887
6.58.2 Macro Definition 887
6.59 GFLIB_ATAN_T 887
6.59.1 Source File 887
6.59.2 Macro Definition 888
6.60 GFLIB_ATAN_DEFAULT 888
6.60.1 Source File 888
6.60.2 Macro Definition 888
6.61 GFLIB_ATAN_DEFAULT_F32 888
6.61.1 Source File 888
6.61.2 Macro Definition 888
6.62 GFLIB_ATAN_DEFAULT_F16 888
6.62.1 Source File 888
6.62.2 Macro Definition 888
6.63 GFLIB_ATAN_DEFAULT_FLT 888
6.63.1 Source File 888
6.63.2 Macro Definition 888
6.64 GFLIB_ATANYXSHIFTED_T 889
6.64.1 Source File 889
6.64.2 Macro Definition 889
6.65 GFLIB_CONTROLLER_PID_P_AW_T 889
6.65.1 Source File 889
6.65.2 Macro Definition 889
6.66 GFLIB_CONTROLLER_PID_P_AW_ DEFAULT 889
6.66.1 Source File 889
6.66.2 Macro Definition 889
6.67 GFLIB_CONTROLLER_PID_P_AW_ DEFAULT_F32 889
6.67.1 Source File 889
6.67.2 Macro Definition 890
6.68 GFLIB_CONTROLLER_PID_P_AW_ DEFAULT_F16 890
6.68.1 Source File 890
6.68.2 Macro Definition 890
6.69 GFLIB_CONTROLLER_PID_P_AW_ DEFAULT_FLT 890
6.69.1 Source File 890
6.69.2 Macro Definition 890
6.70 GFLIB_CONTROLLER_PI_P_T 890
6.70.1 Source File 890
6.70.2 Macro Definition 890
6.71 GFLIB_CONTROLLER_PI_P_DEFAULT 890
6.71.1 Source File 891
6.71.2 Macro Definition 891
6.72 GFLIB_CONTROLLER_PI_P_DEFAULT_ F32 891
6.72.1 Source File 891
6.72.2 Macro Definition 891
6.73 GFLIB_CONTROLLER_PI_P_DEFAULT_ F16 891
6.73.1 Source File 891
6.73.2 Macro Definition 891
6.74 GFLIB_CONTROLLER_PI_P_DEFAULT_ FLT 891
6.74.1 Source File 891
6.74.2 Macro Definition 891
6.75 GFLIB_CONTROLLER_PIAW_P_T 891
6.75.1 Source File 892
6.75.2 Macro Definition 892
6.76 GFLIB_CONTROLLER_PIAW_P_ DEFAULT 892

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.76.1 Source File ... 892
6.76.2 Macro Definition ... 892
6.77 GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32 ... 892
6.77.1 Source File ... 892
6.77.2 Macro Definition ... 892
6.78 GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16 ... 892
6.78.1 Source File ... 892
6.78.2 Macro Definition ... 892
6.79 GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT ... 893
6.79.1 Source File ... 893
6.79.2 Macro Definition ... 893
6.80 GFLIB_CONTROLLER_PI_R_T ... 893
6.80.1 Source File ... 893
6.80.2 Macro Definition ... 893
6.81 GFLIB_CONTROLLER_PI_R_DEFAULT ... 893
6.81.1 Source File ... 893
6.81.2 Macro Definition ... 893
6.82 GFLIB_CONTROLLER_PI_R_DEFAULT_F32 ... 893
6.82.1 Source File ... 893
6.82.2 Macro Definition ... 894
6.83 GFLIB_CONTROLLER_PI_R_DEFAULT_F16 ... 894
6.83.1 Source File ... 894
6.83.2 Macro Definition ... 894
6.84 GFLIB_CONTROLLER_PI_R_DEFAULT_FLT ... 894
6.84.1 Source File ... 894
6.84.2 Macro Definition ... 894
6.85 GFLIB_CONTROLLER_PIAW_R_T ... 894
6.85.1 Source File ... 894
6.85.2 Macro Definition ... 894
6.86 GFLIB_CONTROLLER_PIAW_R_DEFAULT ... 894
6.86.1 Source File ... 895
6.86.2 Macro Definition ... 895
6.87 GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32 ... 895
6.87.1 Source File ... 895
6.87.2 Macro Definition ... 895
6.88 GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16 ... 895
6.88.1 Source File ... 895
6.88.2 Macro Definition ... 895
6.89 GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT ... 895
6.89.1 Source File ... 895
6.89.2 Macro Definition ... 895
6.90 GFLIB_COS_T ... 895
6.90.1 Source File ... 896
6.90.2 Macro Definition ... 896
6.91 GFLIB_COS_DEFAULT ... 896
6.91.1 Source File ... 896
6.91.2 Macro Definition ... 896
6.92 GFLIB_COS_DEFAULT_F32 ... 896
6.92.1 Source File ... 896
6.92.2 Macro Definition ... 896
6.93 GFLIB_COS_DEFAULT_F16 ... 896
6.93.1 Source File ... 896
6.93.2 Macro Definition ... 896
6.94 GFLIB_COS_DEFAULT_FLT ... 896
6.94.1 Source File ... 896
6.94.2 Macro Definition ... 897
6.95 GFLIB_HYST_T ... 897
6.95.1 Source File ... 897
6.95.2 Macro Definition ... 897
6.96 GFLIB_HYST_DEFAULT ... 897
6.96.1 Source File ... 897
6.96.2 Macro Definition ... 897
6.97 GFLIB_HYST_DEFAULT_F32 ... 897
6.97.1 Source File ... 897
6.97.2 Macro Definition ... 897
6.98 GFLIB_HYST_DEFAULT_F16 ... 897
6.98.1 Source File ... 897
6.98.2 Macro Definition ... 898
6.99 GFLIB_HYST_DEFAULT_FLT ... 898
6.99.1 Source File ... 898
6.99.2 Macro Definition ... 898
6.100 GFLIB_INTEGRATOR_TR_T ... 898
6.100.1 Source File ... 898
6.100.2 Macro Definition ... 898
6.101 GFLIB_INTEGRATOR_TR_DEFAULT ... 898
6.101.1 Source File ... 898
6.101.2 Macro Definition ... 898
6.102 GFLIB_INTEGRATOR_TR_DEFAULT_F32 ... 898
6.102.1 Source File ... 898
6.102.2 Macro Definition ... 899
6.103 GFLIB_INTEGRATOR_TR_DEFAULT_F16 ... 899
6.103.1 Source File ... 899
6.103.2 Macro Definition ... 899
6.104 GFLIB_INTEGRATOR_TR_DEFAULT_FLT ... 899
6.104.1 Source File ... 899
6.104.2 Macro Definition ... 899
6.105 GFLIB_LIMIT_T ... 899
6.105.1 Source File ... 899
6.105.2 Macro Definition ... 899
6.106 GFLIB_LIMIT_DEFAULT ... 899
6.106.1 Source File ... 899
6.106.2 Macro Definition ... 900
6.107 GFLIB_LIMIT_DEFAULT_F32 ... 900
6.107.1 Source File ... 900
6.107.2 Macro Definition ... 900
6.108 GFLIB_LIMIT_DEFAULT_F16 ... 900
6.108.1 Source File ... 900
6.108.2 Macro Definition ... 900
6.109 GFLIB_LIMIT_DEFAULT_FLT ... 900
6.109.1 Source File ... 900
6.109.2 Macro Definition ... 900
6.110 GFLIB_LOG10_DEFAULT_FLT ... 900
6.110.1 Source File ... 900
6.110.2 Macro Definition ... 900
6.111 GFLIB_LOWERLIMIT_T ... 901
6.111.1 Source File ... 901
6.111.2 Macro Definition ... 901
6.112 GFLIB_LOWERLIMIT_DEFAULT ... 901

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.112.1 Source File 901
6.112.2 Macro Definition 901
6.113 GFLIB_LOWERLIMIT_DEFAULT_F32 901
6.113.1 Source File 901
6.113.2 Macro Definition 901
6.114 GFLIB_LOWERLIMIT_DEFAULT_F16 901
6.114.1 Source File 901
6.114.2 Macro Definition 901
6.115 GFLIB_LOWERLIMIT_DEFAULT_FLT 902
6.115.1 Source File 902
6.115.2 Macro Definition 902
6.116 GFLIB_LUT1D_T 902
6.116.1 Source File 902
6.116.2 Macro Definition 902
6.117 GFLIB_LUT1D_DEFAULT 902
6.117.1 Source File 902
6.117.2 Macro Definition 902
6.118 GFLIB_LUT1D_DEFAULT_F32 902
6.118.1 Source File 902
6.118.2 Macro Definition 902
6.119 GFLIB_LUT1D_DEFAULT_F16 902
6.119.1 Source File 903
6.119.2 Macro Definition 903
6.120 GFLIB_LUT1D_DEFAULT_FLT 903
6.120.1 Source File 903
6.120.2 Macro Definition 903
6.121 GFLIB_LUT2D_T 903
6.121.1 Source File 903
6.121.2 Macro Definition 903
6.122 GFLIB_LUT2D_DEFAULT 903
6.122.1 Source File 903
6.122.2 Macro Definition 903
6.123 GFLIB_LUT2D_DEFAULT_F32 903
6.123.1 Source File 903
6.123.2 Macro Definition 904
6.124 GFLIB_LUT2D_DEFAULT_F16 904
6.124.1 Source File 904
6.124.2 Macro Definition 904
6.125 GFLIB_LUT2D_DEFAULT_FLT 904
6.125.1 Source File 904
6.125.2 Macro Definition 904
6.126 GFLIB_RAMP_T 904
6.126.1 Source File 904
6.126.2 Macro Definition 904
6.127 GFLIB_RAMP_DEFAULT 904
6.127.1 Source File 904
6.127.2 Macro Definition 904
6.128 GFLIB_RAMP_DEFAULT_F32 905
6.128.1 Source File 905
6.128.2 Macro Definition 905
6.129 GFLIB_RAMP_DEFAULT_F16 905
6.129.1 Source File 905
6.129.2 Macro Definition 905
6.130 GFLIB_RAMP_DEFAULT_FLT 905
6.130.1 Source File 905
6.130.2 Macro Definition 905
6.131 GFLIB_SIN_FLT_MIN 905
6.131.1 Source File 905
6.131.2 Macro Definition 905
6.132 GFLIB_SIN_T 905
6.132.1 Source File 906
6.132.2 Macro Definition 906
6.133 GFLIB_SIN_DEFAULT 906
6.133.1 Source File 906
6.133.2 Macro Definition 906
6.134 GFLIB_SIN_DEFAULT_F32 906
6.134.1 Source File 906
6.134.2 Macro Definition 906
6.135 GFLIB_SIN_DEFAULT_F16 906
6.135.1 Source File 906
6.135.2 Macro Definition 906
6.136 GFLIB_SIN_DEFAULT_FLT 906
6.136.1 Source File 906
6.136.2 Macro Definition 907
6.137 GFLIB_SINCOS_FLT_MIN 907
6.137.1 Source File 907
6.137.2 Macro Definition 907
6.138 GFLIB_SINCOS_T 907
6.138.1 Source File 907
6.138.2 Macro Definition 907
6.139 GFLIB_SINCOS_DEFAULT 907
6.139.1 Source File 907
6.139.2 Macro Definition 907
6.140 GFLIB_SINCOS_DEFAULT_F32 907
6.140.1 Source File 907
6.140.2 Macro Definition 907
6.141 GFLIB_SINCOS_DEFAULT_F16 908
6.141.1 Source File 908
6.141.2 Macro Definition 908
6.142 GFLIB_SINCOS_DEFAULT_FLT 908
6.142.1 Source File 908
6.142.2 Macro Definition 908
6.143 GFLIB_TAN_FLT_MIN 908
6.143.1 Source File 908
6.143.2 Macro Definition 908
6.144 GFLIB_TAN_FLT_3P4 908
6.144.1 Source File 908
6.144.2 Macro Definition 908
6.145 GFLIB_TAN_FLT_PI 908
6.145.1 Source File 909
6.145.2 Macro Definition 909
6.146 GFLIB_TAN_FLT_CORR1 909
6.146.1 Source File 909
6.146.2 Macro Definition 909
6.147 GFLIB_TAN_FLT_PI4 909
6.147.1 Source File 909
6.147.2 Macro Definition 909
6.148 GFLIB_TAN_FLT_PI2 909
6.148.1 Source File 909
6.148.2 Macro Definition 909
6.149 GFLIB_TAN_FLT_CORR2 909
6.149.1 Source File 909
6.149.2 Macro Definition 910
6.150 GFLIB_TAN_T 910
6.150.1 Source File 910
6.150.2 Macro Definition 910
6.151 GFLIB_TAN_DEFAULT 910
6.151.1 Source File 910

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.151.2 Macro Definition 910
6.152 GFLIB_TAN_DEFAULT_F32 910
6.152.1 Source File 910
6.152.2 Macro Definition 910
6.153 GFLIB_TAN_DEFAULT_F16 910
6.153.1 Source File 910
6.153.2 Macro Definition 910
6.154 GFLIB_TAN_DEFAULT_FLT 911
6.154.1 Source File 911
6.154.2 Macro Definition 911
6.155 GFLIB_UPPERLIMIT_T 911
6.155.1 Source File 911
6.155.2 Macro Definition 911
6.156 GFLIB_UPPERLIMIT_DEFAULT 911
6.156.1 Source File 911
6.156.2 Macro Definition 911
6.157 GFLIB_UPPERLIMIT_DEFAULT_F32 911
6.157.1 Source File 911
6.157.2 Macro Definition 911
6.158 GFLIB_UPPERLIMIT_DEFAULT_F16 912
6.158.1 Source File 912
6.158.2 Macro Definition 912
6.159 GFLIB_UPPERLIMIT_DEFAULT_FLT 912
6.159.1 Source File 912
6.159.2 Macro Definition 912
6.160 GFLIB_VECTORLIMIT_T 912
6.160.1 Source File 912
6.160.2 Macro Definition 912
6.161 GFLIB_VECTORLIMIT_DEFAULT 912
6.161.1 Source File 912
6.161.2 Macro Definition 912
6.162 GFLIB_VECTORLIMIT_DEFAULT_F32 913
6.162.1 Source File 913
6.162.2 Macro Definition 913
6.163 GFLIB_VECTORLIMIT_DEFAULT_F16 913
6.163.1 Source File 913
6.163.2 Macro Definition 913
6.164 GFLIB_VECTORLIMIT_DEFAULT_FLT 913
6.164.1 Source File 913
6.164.2 Macro Definition 913
6.165 GFLIB_VLOG10_DEFAULT_FLT 913
6.165.1 Source File 913
6.165.2 Macro Definition 913
6.166 GMCLIB_DECOUPLINGPMSM_T 913
6.166.1 Source File 914
6.166.2 Macro Definition 914
6.167 GMCLIB_DECOUPLINGPMSM_DEFAULT ...914
6.167.1 Source File 914
6.167.2 Macro Definition 914
6.168 GMCLIB_DECOUPLINGPMSM_ DEFAULT_F32 914
6.168.1 Source File 914
6.168.2 Macro Definition 914
6.169 GMCLIB_DECOUPLINGPMSM_ DEFAULT_F16 914
6.169.1 Source File 914
6.169.2 Macro Definition 914
6.170 GMCLIB_DECOUPLINGPMSM_ DEFAULT_FLT 914
6.170.1 Source File 915
6.170.2 Macro Definition 915
6.171 GMCLIB_ELIMDCBUSRIP_T 915
6.171.1 Source File 915
6.171.2 Macro Definition 915
6.172 GMCLIB_ELIMDCBUSRIP_DEFAULT 915
6.172.1 Source File 915
6.172.2 Macro Definition 915
6.173 GMCLIB_ELIMDCBUSRIP_DEFAULT_F32 .. 915
6.173.1 Source File 915
6.173.2 Macro Definition 915
6.174 GMCLIB_ELIMDCBUSRIP_DEFAULT_F16 .. 915
6.174.1 Source File 916
6.174.2 Macro Definition 916
6.175 GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT .. 916
6.175.1 Source File 916
6.175.2 Macro Definition 916
6.176 SWLIBS_SUPPORT_F32 916
6.176.1 Source File 916
6.176.2 Macro Definition 916
6.177 SWLIBS_SUPPORT_F16 916
6.177.1 Source File 916
6.177.2 Macro Definition 916
6.178 SWLIBS_SUPPORT_FLT 916
6.178.1 Source File 916
6.178.2 Macro Definition 917
6.179 SWLIBS_SUPPORTED_ IMPLEMENTATION 917
6.179.1 Source File 917
6.179.2 Macro Definition 917
6.180 SFRACT_MIN 917
6.180.1 Source File 917
6.180.2 Macro Definition 917
6.181 SFRACT_MAX 917
6.181.1 Source File 917
6.181.2 Macro Definition 917
6.182 FRACT_MIN 917
6.182.1 Source File 917
6.182.2 Macro Definition 918
6.183 FRACT_MAX 918
6.183.1 Source File 918
6.183.2 Macro Definition 918
6.184 FRAC32_0_5 918
6.184.1 Source File 918
6.184.2 Macro Definition 918
6.185 FRAC16_0_5 918
6.185.1 Source File 918
6.185.2 Macro Definition 918
6.186 FRAC32_0_25 918
6.186.1 Source File 918
6.186.2 Macro Definition 918
6.187 FRAC16_0_25 919
6.187.1 Source File 919
6.187.2 Macro Definition 919
6.188 UINT16_MAX 919
6.188.1 Source File 919
6.188.2 Macro Definition 919
6.189 INT16_MAX 919
6.189.1 Source File 919

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.189.2 Macro Definition 919
6.190 INT16_MIN 919
6.190.1 Source File 919
6.190.2 Macro Definition 919
6.191 UINT32_MAX 920
6.191.1 Source File 920
6.191.2 Macro Definition 920
6.192 INT32_MAX 920
6.192.1 Source File 920
6.192.2 Macro Definition 920
6.193 INT32_MIN 920
6.193.1 Source File 920
6.193.2 Macro Definition 920
6.194 FLOAT_MIN 920
6.194.1 Source File 920
6.194.2 Macro Definition 920
6.195 FLOAT_MAX 921
6.195.1 Source File 921
6.195.2 Macro Definition 921
6.196 INT16TOINT32 921
6.196.1 Source File 921
6.196.2 Macro Definition 921
6.197 INT32TOINT16 921
6.197.1 Source File 921
6.197.2 Macro Definition 921
6.198 INT32TOINT64 921
6.198.1 Source File 921
6.198.2 Macro Definition 921
6.199 INT64TOINT32 921
6.199.1 Source File 922
6.199.2 Macro Definition 922
6.200 F16TOINT16 922
6.200.1 Source File 922
6.200.2 Macro Definition 922
6.201 F32TOINT16 922
6.201.1 Source File 922
6.201.2 Macro Definition 922
6.202 F64TOINT16 922
6.202.1 Source File 922
6.202.2 Macro Definition 922
6.203 F16TOINT32 922
6.203.1 Source File 923
6.203.2 Macro Definition 923
6.204 F32TOINT32 923
6.204.1 Source File 923
6.204.2 Macro Definition 923
6.205 F64TOINT32 923
6.205.1 Source File 923
6.205.2 Macro Definition 923
6.206 F16TOINT64 923
6.206.1 Source File 923
6.206.2 Macro Definition 923
6.207 F32TOINT64 923
6.207.1 Source File 924
6.207.2 Macro Definition 924
6.208 F64TOINT64 924
6.208.1 Source File 924
6.208.2 Macro Definition 924
6.209 INT16TOF16 924
6.209.1 Source File 924
6.209.2 Macro Definition 924
6.210 INT16TOF32 924
6.210.1 Source File 924
6.210.2 Macro Definition 924
6.211 INT32TOF16 924
6.211.1 Source File 924
6.211.2 Macro Definition 925
6.212 INT32TOF32 925
6.212.1 Source File 925
6.212.2 Macro Definition 925
6.213 INT64TOF16 925
6.213.1 Source File 925
6.213.2 Macro Definition 925
6.214 INT64TOF32 925
6.214.1 Source File 925
6.214.2 Macro Definition 925
6.215 F16_1_DIVBY_SQRT3 925
6.215.1 Source File 925
6.215.2 Macro Definition 925
6.216 F32_1_DIVBY_SQRT3 926
6.216.1 Source File 926
6.216.2 Macro Definition 926
6.217 F16_SQRT3_DIVBY_2 926
6.217.1 Source File 926
6.217.2 Macro Definition 926
6.218 F16_MINUS_SQRT3_DIVBY_2 926
6.218.1 Source File 926
6.218.2 Macro Definition 926
6.219 F16_SQRT3_DIVBY_4 926
6.219.1 Source File 926
6.219.2 Macro Definition 926
6.220 F16_SQRT3_DIVBY_6 926
6.220.1 Source File 927
6.220.2 Macro Definition 927
6.221 F32_SQRT3_DIVBY_2 927
6.221.1 Source File 927
6.221.2 Macro Definition 927
6.222 F32_MINUS_SQRT3_DIVBY_2 927
6.222.1 Source File 927
6.222.2 Macro Definition 927
6.223 F32_SQRT3_DIVBY_4 927
6.223.1 Source File 927
6.223.2 Macro Definition 927
6.224 F32_SQRT3_DIVBY_6 927
6.224.1 Source File 927
6.224.2 Macro Definition 928
6.225 F16_SQRT2_DIVBY_2 928
6.225.1 Source File 928
6.225.2 Macro Definition 928
6.226 F32_SQRT2_DIVBY_2 928
6.226.1 Source File 928
6.226.2 Macro Definition 928
6.227 F16_1_DIVBY_3 928
6.227.1 Source File 928
6.227.2 Macro Definition 928
6.228 F32_1_DIVBY_3 928
6.228.1 Source File 928
6.228.2 Macro Definition 928

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2023. All rights reserved.

6.229 F16_MINUS_1_DIVBY_3 929
6.229.1 Source File 929
6.229.2 Macro Definition 929
6.230 F32_MINUS_1_DIVBY_3 929
6.230.1 Source File 929
6.230.2 Macro Definition 929
6.231 F16_MINUS_1_DIVBY_2 929
6.231.1 Source File 929
6.231.2 Macro Definition 929
6.232 F32_MINUS_1_DIVBY_2 929
6.232.1 Source File 929
6.232.2 Macro Definition 929
6.233 F16_2_DIVBY_3 929
6.233.1 Source File 930
6.233.2 Macro Definition 930
6.234 F32_2_DIVBY_3 930
6.234.1 Source File 930
6.234.2 Macro Definition 930
6.235 FRAC16 930
6.235.1 Source File 930
6.235.2 Macro Definition 930
6.236 FRAC32 930
6.236.1 Source File 930
6.236.2 Macro Definition 930
6.237 FLOAT_DIVBY_SQRT3 930
6.237.1 Source File 931
6.237.2 Macro Definition 931
6.238 FLOAT_SQRT3 931
6.238.1 Source File 931
6.238.2 Macro Definition 931
6.239 FLOAT_SQRT3_DIVBY_2 931
6.239.1 Source File 931
6.239.2 Macro Definition 931
6.240 FLOAT_MINUS_SQRT3_DIVBY_2 931
6.240.1 Source File 931
6.240.2 Macro Definition 931
6.241 FLOAT_SQRT3_DIVBY_4 931
6.241.1 Source File 931
6.241.2 Macro Definition 932
6.242 FLOAT_SQRT3_DIVBY_6 932
6.242.1 Source File 932
6.242.2 Macro Definition 932
6.243 FLOAT_SQRT3_DIVBY_4_CORRECTION ... 932
6.243.1 Source File 932
6.243.2 Macro Definition 932
6.244 FLOAT_2_PI 932
6.244.1 Source File 932
6.244.2 Macro Definition 932
6.245 FLOAT_2_PI_DIVBY_3 932
6.245.1 Source File 932
6.245.2 Macro Definition 932
6.246 FLOAT_PI 933
6.246.1 Source File 933
6.246.2 Macro Definition 933
6.247 FLOAT_PI_DIVBY_2 933
6.247.1 Source File 933
6.247.2 Macro Definition 933
6.248 FLOAT_TAN_PI_DIVBY_6 933
6.248.1 Source File 933
6.248.2 Macro Definition 933
6.249 FLOAT_TAN_PI_DIVBY_12 933
6.249.1 Source File 933
6.249.2 Macro Definition 933
6.250 FLOAT_PI_DIVBY_6 933
6.250.1 Source File 934
6.250.2 Macro Definition 934
6.251 FLOAT_PI_SINGLE_CORRECTION 934
6.251.1 Source File 934
6.251.2 Macro Definition 934
6.252 FLOAT_PI_CORRECTION 934
6.252.1 Source File 934
6.252.2 Macro Definition 934
6.253 FLOAT_PI_DIVBY_4 934
6.253.1 Source File 934
6.253.2 Macro Definition 934
6.254 FLOAT_4_DIVBY_PI 934
6.254.1 Source File 934
6.254.2 Macro Definition 935
6.255 FLOAT_0_5 935
6.255.1 Source File 935
6.255.2 Macro Definition 935
6.256 FLOAT_MINUS_0_5 935
6.256.1 Source File 935
6.256.2 Macro Definition 935
6.257 FLOAT_PLUS_1 935
6.257.1 Source File 935
6.257.2 Macro Definition 935
6.258 FLOAT_MINUS_1 935
6.258.1 Source File 935
6.258.2 Macro Definition 935
6.259 FLOAT_MIN_NORM 936
6.259.1 Source File 936
6.259.2 Macro Definition 936
6.260 SWLIBS_2Syst 936
6.260.1 Source File 936
6.260.2 Macro Definition 936
6.261 SWLIBS_3Syst 936
6.261.1 Source File 936
6.261.2 Macro Definition 936
6.262 SWLIBS_6Syst 936
6.262.1 Source File 936
6.262.2 Macro Definition 936
6.263 SWLIBS_ID 937
6.263.1 Source File 937
6.263.2 Macro Definition 937
7 Legal information 938

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

© NXP B.V. 2023. **All rights reserved.**

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 15 September 2023
Document identifier: S32K3XXMCLUG
Document number: S32K3XXMCLUG