Automotive Math and Motor Control Library Set for NXP MPC574xP devices

User's Guide

Document Number: MPC574XPMCLUG
Rev. 12

Contents

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

Chapter 1
Revision History

Chapter 2

Chapter 3
Introduction

Chapter 4

Chapter 5
API References

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Chapter 6

Section number	Title	Page

Chapter 7
Compound Data Types

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Chapter 8

Chapter 9
Macro References

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Chapter 1
Revision History

Table 1-1. Revision History

Revision	Date	Author	Description
1.0	19/02/2013	Jiri Kuhn	Initial version.
2.0	23/07/2013	Jiri Kuhn	The user guide was updated to reflect the change of the installation process.
3.0	08/11/2013	Jiri Kuhn	Changed Matlab Integration chapter (added 64-bit BAM models). Added functions precision table.
4.0	25/04/2014	Petr Zelinka	Added missing descriptions in some functions, formatting changes. Added revision history table.
5.0	31/03/2015	Jiri Kuhn	The Matlab revision, used for testing, changed to Matlab 2014a.
6.0	31/07/2015	Petr Fajmon	The Qorivva brand name has been retired and was removed. All compilers versions were increased. Number of the significant digits of the calculated IIR1/IIR2 coefficients in the MATLAB examples for GDFLIB_IIR1/GDFLIB_IIR2 functions was increased.
7.0	30/09/2015	Jiri Kuhn	Adding the chapters describing the GFLIB_SinCos function.
8.0	31/12/2015	Petr Fajmon	The Service Release v1.1.3. The example code for GFLIB_Lut1D_FLT function was corrected. The GreenHills compiler version was increased to the latest v2015.1.4. Missing equations for GFLIB_Sin_FLT, GFLIB_Cos_FLT and some other functions were added.
9.0	31/03/2016	Jiri Kuhn	Change from Freescale to NXP entity, Service release 1.1.4.
10.0	30/06/2016	Petr Fajmon	The Service Release v1.1.5. Added support of the S32 Design Studio for Power Architecture. Added new MLIB_RndSat_F16F32 function. Corrected examples for GFLIB_ControllerPIrAW_FLT/ GFLIB_ControllerPIr_FLT functions. Added chapter about possible exceptions.
11.0	30/09/2016	Jiri Kuhn	The Service Release v1.1.6. Correction of S32 Design Studio for Power Architecture integration chapter. The WindRiver Diab compiler version was increased to 5.9.4.8.

Table continues on the next page...

Table 1-1. Revision History (continued)

Revision	Date	Author	Description
12.0	31/12/2016	Petr Fajmon	The Service Release v1.1.7. Added AMCLIB_BemfObsrvDQ and AMCLIB_TrackObsrv functions. Software License Agreement updated.
13.0	31/03/2017	Petr Fajmon	The Service Release v1.1.8. Repeated sections were merged into a common chapters in the User Guide. This approach was applied to all AMMCLIB functions.

Chapter 2

2.1 ATTACHMENT A - NXP Automotive Software License Agreement v1.6

IMPORTANT. Read the following NXP Software License Agreement ("Agreement") completely. By selecting the "I Accept" button at the end of this page, you indicate that you accept the terms of this Agreement. You may then install the software.

NXP SOFTWARE LICENSE AGREEMENT This is a legal agreement between you (either as an individual or as an authorized representative of your employer) ("you" or "Licensee") and NXP USA, Inc. ("NXP"). It concerns your rights to use this file and any accompanying written materials (the "Software"). In consideration for NXP allowing you to access the Software, you are agreeing to be bound by the terms of this Agreement. If you do not agree to all of the terms of this Agreement, do not download the Software. If you change your mind later, stop using the Software and delete all copies of the Software in your possession or control. Any copies of the Software that you have already distributed, where permitted, and do not destroy will continue to be governed by this Agreement. Your prior use will also continue to be governed by this Agreement. For the avoidance of doubt, any third party software indicated in the Quotation Document as being provided is not Software licensed under this Agreement and is governed solely by the terms and conditions of the third party licensor. In addition, the parties acknowledge that the Software may contain Open Source Software components. For the avoidance of doubt, all Open Source Software components are governed solely by the applicable Open Source Software licenses.

LICENSE GRANT: Your rights to use this Software vary depending upon the type of license model you ordered from NXP. The type of license will be set forth in the NXP Quotation document which formed the basis of your order or, with respect to evaluation licenses, in the correspondence with NXP confirming your request for a copy of the Software for evaluation. If you do not remember this information, you may contact the party from whom you obtained the Software or NXP at www.NXP.com.

Definitions: For purposes of this Agreement the following terms are defined as set forth below: "Authorized System" means the Authorized System set forth in the Quotation Documents or, if none is specified, Licensee's ECU product containing a NXP processor or other semiconductor product supplied directly or indirectly from NXP. "Confidential Information" means any information disclosed by NXP to the Licensee and, if disclosed in writing or in some other tangible form, that is marked at the time of disclosure as being "Confidential" or "Proprietary" or with words of similar import; provided, that Software provided in source code format will be deemed Confidential Information whether or not identified as such in writing or otherwise. Confidential Information does not include any information that: (a) is, or becomes, publicly known through no wrongful act on the Licensee's part; (b) is already known to the Licensee, or becomes lawfully known to the Licensee without restriction on disclosure; or (c) is independently developed by the Licensee. "Customer Product Line" means the customer product line specified in the Quotation Document. "Customer Target Project" means the one customer Target Project for one automotive vehicle manufacturer specified in the Quotation Document. "Limited Derivative Works" means Licensee may make derivative works only to the extent the copyrighted material provided by NXP is not modified. For the avoidance of doubt, Licensee explicitly has the right to: 1) add source code to source code provided by NXP without modification to the original source code; 2) compile any source code into object code, or 3) integrate the NXP source code into its ECU without modifying the original source code. "Quotation Document" means the NXP or NXP authorized reseller document offering the goods and/or services which formed the basis of your order and which is incorporated herein by this reference. "DISM Package" is a commercial offering bundling a Development License with software support and maintenance services as described in detail in the Quotation document. "NXP Target Product" means the NXP processor or other semiconductor product set forth in the Quotation Document. "NXP Target Product Family" means the NXP processor family or other semiconductor product family set forth in the Quotation Document. "Open Source Software" means any software that is subject to terms that, as a condition of use, copying, modification or redistribution, require such software and/or derivative works thereof to be disclosed or distributed in source code form, to be licensed for the purpose of making derivative works, or to be redistributed free of charge, including without limitation software distributed under the GNU General Public License or GNU Lesser/Library and General Public License, Apache or BSD.

I. Non-Production License Models:

a. Evaluation License: NXP grants to you, free of charge, a personal, non-transferable, non-exclusive, revocable, royalty-free, license to use the Software for ninety (90) consecutive days from the date of your acceptance as indicated by clicking the "I accept" button below, internally, solely for the purpose of performing evaluation and testing of the Software, solely for automotive use, and solely on a NXP Target Product. This license does not include the right to reproduce, distribute, sell, lease, sublicense or transfer all or any part of the Software, or to use the Software on non-NXP integrated circuit devices, or to use the Software for any production purposes whatsoever. NXP reserves all rights not expressly granted in this Agreement. If you violate any of the terms or restrictions of this Agreement, NXP may immediately terminate this Agreement, and require that you stop using and delete all copies of the Software in your possession or control. b. Development License as part of the DISM Package: Subject to payment of the fees set forth in the Quotation Document, NXP grants to you a personal, non-transferable, non-exclusive, revocable, license for 12 months beginning on the date of your registration or, if you are renewing, the anniversary date of your registration or prior renewal as indicated by clicking the "I accept" button below, unless a different license term is specified in the Quotation Document ("Term"), (1) to use the Software on one Target Product Family for execution on a maximum of 500 sample Authorized Systems, (2) to reproduce the Software, (3) to prepare Limited Derivative Works of the Licensed Software (4) to distribute the Software and Limited Derivative Works thereof to automotive customers in object code only, (5) to sublicense to automotive customers the right to use the distributed Software in object code only as included within the Authorized System solely for purposes of testing the Authorized System during the Term, and (6) to disclose one copy of the Software and Limited Derivative Works thereof in source code to direct automotive customers solely for internal evaluation and testing or archiving (and not for further use, reproduction, modification or redistribution) only where such disclosure is necessary in order to achieve the purpose of the license because the automotive customer requires disclosure of source code in conjunction with the testing or purchase of an Authorized System incorporating a NXP Target Product, and where the direct automotive customer has executed written agreements obligating itself to observe the limitations set out in this subsection (6) and to protect such Confidential Information on terms at least as strict as those contained in this Agreement. NXP reserves all rights not expressly granted in this Agreement. You must prohibit your sublicensees from translating, reverse engineering, decompiling, or disassembling the Software except to the extent applicable law specifically prohibits such restriction. If you violate any of the terms or restrictions of this Agreement, NXP may immediately terminate this Agreement, and require that you stop using and delete all copies of the Software in your possession or control. You are solely responsible for systems you design using the Software.

II. Production License Models:

a. Project License: Subject to payment of the fees set forth in the Quotation Document, exclusively in conjunction with Licensee's development and sale of an Authorized System, NXP grants to you the non-exclusive, non-transferable right (1) to use the Software in the Customer Target Project on the Target Product, (2) to reproduce the Software, (3) to prepare or have prepared through multiple layers of subcontractors

Limited Derivative Works of the Software, (4) to distribute the Software and Limited Derivative Works thereof in object code only as part of an Authorized System, (5) to sublicense to others the right to use the distributed Software in object code only as included within the Authorized System, and (6) to disclose one copy of the Software and Limited Derivative Works thereof in source code to direct automotive customers solely for internal evaluation and testing or archiving (and not for further use, reproduction, modification or redistribution) only where such disclosure is necessary in order to achieve the purpose of the license because the automotive customer requires disclosure of source code in conjunction with the testing or purchase of an Authorized System incorporating a NXP Target Product, and where the direct automotive customer has executed written agreements obligating itself to observe the limitations set out in this subsection (6) and to protect such Confidential Information on terms at least as strict as those contained in this Agreement. Licensee will remain liable for such subcontractor's adherence to the terms of this Agreement and for any and all acts and omissions of such subcontractors with respect to this Agreement. Notwithstanding limitation on damages in this Agreement, Licensee will indemnify, defend, and hold harmless NXP against any and all claims, costs, damages, liabilities, judgments and attorneys' fees resulting from or arising out of any breach by the sublicensee, or resulting from or arising out of any action by the sublicensee inconsistent with this Agreement. You must prohibit your sublicensees from translating, reverse engineering, decompiling, or disassembling the Software except to the extent applicable law specifically prohibits such restriction. If you violate any of the terms or restrictions of this Agreement, NXP may immediately terminate this Agreement, and require that you stop using and delete all copies of the Software in your possession or control.

b. Product Line License: Subject to payment of the fees set forth in the Quotation Document, exclusively in conjunction with Licensee's development and sale of an Authorized System, NXP grants to you, the non-exclusive, non-transferable right (1) to use the Software in one Target Product Family in one Customer Product Line, (2) to reproduce the Software, (3) to prepare or have prepared through multiple layers of subcontractors Limited Derivative Works of the Software, (4) to distribute the Software and Limited Derivative Works thereof in object code only as part of an Authorized System, (5) to sublicense to others the right to use the distributed Software in object code only as included within the Authorized System and (6) to disclose one copy of the Software and Limited Derivative Works thereof in source code to direct automotive customers solely for internal evaluation and testing or archiving (and not for further use, reproduction, modification or redistribution) only where such disclosure is necessary in order to achieve the purpose of the license because the automotive customer requires disclosure of source code in conjunction with the testing or purchase of an Authorized System incorporating a NXP Target Product, and where the direct automotive customer has executed written agreements obligating itself to observe the limitations set out in this subsection (6) and to protect such Confidential Information on terms at least as strict as those contained in this Agreement. Licensee will remain liable for such subcontractor's adherence to the terms of this Agreement and for any and all acts and omissions of such subcontractors with respect to this Agreement. Notwithstanding limitation on damages in this Agreement, Licensee will indemnify, defend, and hold harmless NXP against any and all claims, costs, damages, liabilities, judgments and attorneys' fees resulting from or arising out of any breach by the sublicensee, or resulting from or arising out of any action by the sublicensee inconsistent with this Agreement. You must prohibit your sublicensees from translating, reverse engineering, decompiling, or disassembling the Software except to the extent applicable law specifically prohibits such restriction. If you violate any of the terms or restrictions of this Agreement, NXP may immediately terminate this Agreement, and require that you stop using and delete all copies of the Software in your possession or control.

c. Family Multi-Project License: Subject to payment of the fees set forth in the Quotation Document, exclusively in conjunction with Licensee's development and sale of an Authorized System, NXP grants to you, the non-exclusive, non-transferable right (1) to use the Software in one Target Product Family, (2) to reproduce the Software, (3) to prepare or have prepared through multiple layers of subcontractors Limited Derivative Works of the Software, (4) to distribute the Software and Limited Derivative Works thereof in object code only as part of an Authorized System, (5) to sublicense to others the right to use the distributed Software in object code only as included within the Authorized System and (6) to disclose one copy of the Software and Limited Derivative Works thereof in source code to direct automotive customers solely for internal evaluation and testing or archiving (and not for further use, reproduction, modification or redistribution) only where such disclosure is necessary in order to achieve the purpose of the license because the automotive customer requires disclosure of source code in conjunction with the testing or purchase of an Authorized System incorporating a NXP Target Product, and where the direct automotive customer has executed written agreements obligating itself to observe the limitations set out in this subsection (6) and to protect such Confidential Information on terms at least as strict as those contained in this Agreement. Licensee will remain liable for such subcontractor's adherence to the terms of this Agreement and for any and all acts and omissions of such subcontractors with respect to this Agreement. Notwithstanding limitation on damages in this Agreement, Licensee will indemnify, defend, and hold harmless NXP against any and all claims, costs, damages, liabilities, judgments and attorneys' fees resulting from or arising out of any breach by the sublicensee, or resulting from or arising out of any action by the sublicensee inconsistent with this Agreement. You must prohibit your sublicensees from translating, reverse engineering, decompiling, or disassembling the Software except to the extent applicable law specifically prohibits such restriction. If you violate any of the terms or restrictions of this Agreement, NXP may immediately terminate this Agreement, and require that you stop using and delete all copies of the Software in your possession or control.

d. Volume License: Subject to payment of the fees set forth in the Quotation Document., exclusively in conjunction with Licensee's development and sale of a Authorized System, NXP grants to you, the non-exclusive, non-transferable right (1) to use the Software in one Target Product Family on a maximum of the number of Authorized Systems set forth in the Quotation Document, (2) to reproduce the Software, (3) to prepare or have prepared through multiple layers of subcontractors Limited Derivative Works of the Software, (4) to distribute the Software and Limited Derivative Works thereof in object code only as part of an Authorized System, (5) to sublicense to others the right to use the distributed Software in object code only as included within the Authorized System and (6) to disclose one copy of the Software and Limited Derivative Works thereof in source code to direct automotive customers solely for internal evaluation and testing or archiving (and not for further use, reproduction, modification or redistribution) only where such disclosure is necessary in order to achieve the purpose of the license because the automotive customer requires disclosure of source code in conjunction with the testing or purchase of an Authorized System incorporating a NXP Target Product, and where the direct automotive customer has executed written agreements obligating itself to observe the limitations set out in this subsection (6) and to protect such Confidential Information on terms at least as strict as those contained in this Agreement. Licensee will remain liable for such subcontractor's adherence to the terms of this Agreement and for any and all acts and omissions of such subcontractors with respect to this Agreement. Notwithstanding limitation on damages in this Agreement, Licensee will indemnify, defend, and hold harmless NXP against any and all claims, costs, damages, liabilities, judgments and attorneys' fees resulting from or arising out of any breach by the sublicensee, or resulting from or arising out of any action by the sublicensee inconsistent with this Agreement. You must prohibit your sublicensees from translating, reverse engineering, decompiling, or disassembling the Software except to the extent applicable law specifically prohibits such restriction. If you violate any of the terms or restrictions of this Agreement, NXP may immediately terminate this Agreement, and require that you stop using and delete all copies of the Software in your possession or control.

COPYRIGHT. The Software is licensed to you, not sold. NXP owns the Software, and United States copyright laws and international treaty provisions protect the Software. Therefore, you must treat the Software like any other copyrighted material (e.g. a book or musical recording). You may not use or copy the Software for any other purpose than what is described in this Agreement. Except as expressly provided herein, NXP does not grant to you any express or implied rights under any NXP or third party patents, copyrights, trademarks, or trade secrets. Additionally, you must reproduce and apply any copyright or other proprietary rights notices included on or embedded in the Software to any copies or derivative works made thereof, in whole or in part, if any.

SUPPORT. Except as otherwise agreed by the parties under separate agreement or as specified in the Quotation Document as part of the DISM Package, NXP is NOT obligated to provide any support, upgrades or new releases of the Software. If you wish, you may contact NXP and report problems and provide suggestions regarding the Software. NXP has no obligation whatsoever to respond in any way to such a problem report or suggestion. NXP may make changes to the Software at any time, without any obligation to notify or provide updated versions of the Software to you.

NO WARRANTY. TO THE MAXIMUM EXTENT PERMITTED BY LAW, NXP EXPRESSLY DISCLAIMS ANY WARRANTY FOR THE SOFTWARE. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. YOU ASSUME THE ENTIRE RISK ARISING OUT OF THE USE OR PERFORMANCE OF THE SOFTWARE, OR ANY SYSTEMS YOU DESIGN USING THE SOFTWARE (IF ANY). NOTHING IN THIS AGREEMENT MAY BE CONSTRUED AS A WARRANTY OR REPRESENTATION BY NXP THAT THE SOFTWARE OR ANY DERIVATIVE WORK DEVELOPED WITH OR INCORPORATING THE SOFTWARE WILL BE FREE FROM INFRINGEMENT OF THE INTELLECTUAL PROPERTY RIGHTS OF THIRD PARTIES.

INDEMNITY. You agree to fully defend and indemnify NXP from any and all claims, liabilities, and costs (including reasonable attorney's fees) related to (1) your use (including your sublicensee's use, if permitted) of the Software or (2) your violation of the terms and conditions of this Agreement.

LIMITATION OF LIABILITY. IN NO EVENT WILL NXP BE LIABLE, WHETHER IN CONTRACT, TORT, OR OTHERWISE, FOR ANY INCIDENTAL, SPECIAL, INDIRECT, CONSEQUENTIAL OR PUNITIVE DAMAGES, INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR ANY LOSS OF USE, LOSS OF TIME, INCONVENIENCE, COMMERCIAL LOSS, OR LOST PROFITS, SAVINGS, OR REVENUES TO THE FULL EXTENT SUCH MAY BE DISCLAIMED BY LAW.

COMPLIANCE WITH LAWS. You must use the Software in accordance with all applicable laws, regulations and statutes.

EXPORT CONTROLS. Each party shall comply with all applicable export and import control laws and regulations including but not limited to the US Export Administration Regulations (including prohibited party lists issued by other federal governments), Catch-all regulations and all national and international embargoes. Each party further agrees that it will not knowingly transfer, divert, export or re-export, directly or indirectly, any product, software, including software source code, or technology restricted by such regulations or by other applicable national regulations, received from the other party under this Agreement, or any direct product of such software or technical data to any person, firm, entity, country or destination to which such transfer, diversion, export or re-export is restricted or prohibited, without obtaining prior written authorization from the applicable competent government authorities to the extent required by those laws. This provision shall survive termination or expiration of this Agreement.

GOVERNMENT USE. Use of the Software and any corresponding documentation, if any, is provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(l) and (2) of the Commercial Computer Software--Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is NXP Semiconductors, 6501 William Cannon Drive West, Austin, TX, 78735.

HIGH RISK ACTIVITIES. You acknowledge that the Software is not fault tolerant and is not designed, manufactured or intended by NXP for incorporation into products intended for use or resale in on-line control equipment in hazardous, dangerous to life or potentially life-threatening environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems, in which the failure of products could lead directly to death, personal injury or severe physical or environmental damage ("High Risk Activities"). You specifically represent and warrant that you will not use the Software or any derivative work of the Software for High Risk Activities.

CHOICE OF LAW; VENUE; LIMITATIONS. You agree that the statutes and laws of the United States and the State of Texas, USA, without regard to conflicts of laws principles, will apply to all matters relating to this Agreement or the Software, and you agree that any litigation will be subject to the exclusive jurisdiction of the state or federal courts in Texas, USA. You agree that regardless of any statute or law to the contrary, any claim or cause of action arising out of or related to this Agreement or the Software must be filed within one (1) year after such claim or cause of action arose or be forever barred.

PRODUCT LABELING. You are not authorized to use any NXP trademarks, brand names, or logos.

ENTIRE AGREEMENT. This Agreement constitutes the entire agreement between you and NXP regarding the subject matter of this Agreement, and supersedes all prior communications, negotiations, understandings, agreements or representations, either written or oral, if any. This Agreement may only be amended in written form, executed by you and NXP.

SEVERABILITY. If any provision of this Agreement is held for any reason to be invalid or unenforceable, then the remaining provisions of this Agreement will be unimpaired and, unless a modification or replacement of the invalid or unenforceable provision is further held to deprive you or NXP of a material benefit, in which case the Agreement will immediately terminate, the invalid or unenforceable provision will be replaced with a provision that is valid and enforceable and that comes closest to the intention underlying the invalid or unenforceable provision.

NO WAIVER. The waiver by NXP of any breach of any provision of this Agreement will not operate or be construed as a waiver of any other or a subsequent breach of the same or a different provision.

CONFIDENTIAL INFORMATION. The Licensee agrees to retain the Confidential Information in confidence perpetually, with respect to Software in source code form (human readable), or for a period of 3 years from the date of receipt of the Confidential Information, with respect to all other Confidential Information. During this period the Licensee may not disclose NXP's Confidential Information to any third party except where such disclosure is necessary in order to achieve the purpose of the license because the automotive customer requires disclosure in conjunction with the testing or purchase of an Authorized System and who have executed written agreements obligating them to protect such Confidential Information. During this period Licensee will also limit dissemination of the Confidential Information to its employees or contractors who have a need to know of the Confidential Information and who have executed written agreements obligating them to protect such Confidential Information, and may not use the NXP's Confidential Information for any purpose or in any manner not authorized by this Agreement. The Licensee further agrees to use the same degree of care, but no less than a reasonable degree of care, with NXP's Confidential Information as it would with its own confidential information. The Licensee may disclose Confidential Information as required by a court or under operation of law or order provided that the Licensee notifies NXP of such requirement prior to disclosure, that the Licensee discloses only that information required, and that the Licensee allows NXP the opportunity to object to such court or other legal body requiring such disclosure.

ASSIGNMENT. Licensee may not assign this Agreement, by operation of law or otherwise, in whole or in part, or any of its rights, interests, duties or obligations, without the prior written approval of NXP. NXP may assign this Agreement without the consent or approval of Licensee. For clarity, the proposed acquisition transaction involving Qualcomm Inc. ("Qualcomm"), which was publicly announced on October 27, 2016, shall not constitute an assignment by operation of law, and NXP may assign this Agreement to Qualcomm or any subsidiary thereof in connection with or following completion of the acquisition.

Chapter 3
Introduction

The aim of this document is to describe the Automotive Math and Motor Control Library Set for NXP MPC574xP devices. It describes the components of the library, its behavior and interaction, the API and steps needed to integrate the library into the customer project.

3.1 Architecture Overview

The Automotive Math and Motor Control Library Set for NXP MPC574xP devices consists of several sub-libraries, functionally connected as depicted in Figure 3-1.

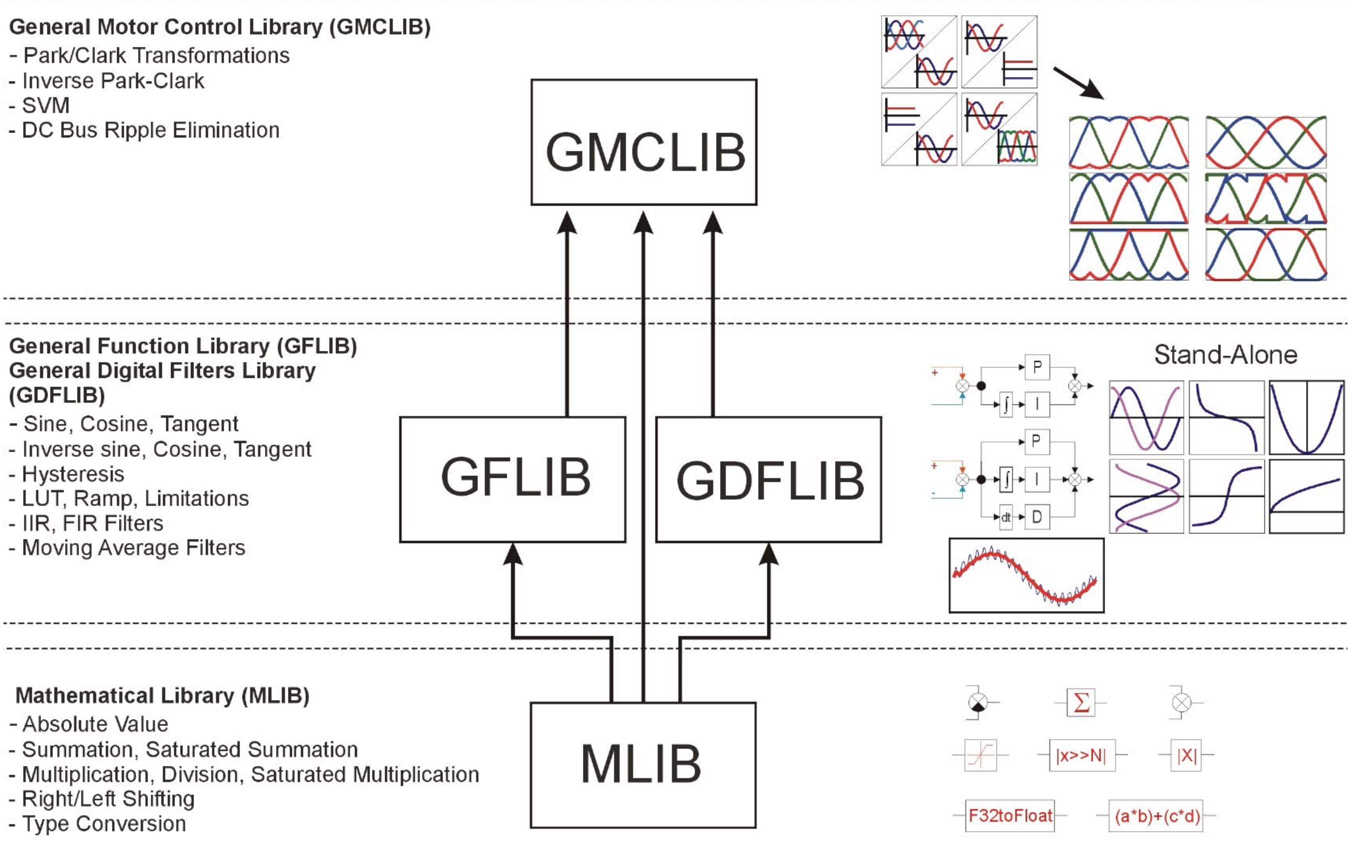

Figure 3-1. AMMCLIB components structure

The Automotive Math and Motor Control Library Set for NXP MPC574xP devices 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 3-1, the Automotive Math and Motor Control Library Set for NXP MPC574xP devices libraries form the layer architecture where all upper libraries utilize the functions from MLIB library. This concept is a key factor for mathematical operations abstractions allowing to support the highly target-optimized variants.

3.2 General Information

The Automotive Math and Motor Control Library Set for NXP MPC574xP devices 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}.

Also those functions which are not relevant for particular implementation, e.g. saturated functions or shifting for single precision floating point implementation, are not delivered with this package. For detailed information about available functions please refer to Function Index.

NOTE

> The fixed-point 32-bit fractional and fixed-point 16-bit fractional functions are implemented based on the unity model. Which means that the input and output numbers are normalized to fit between the $<-1; 1-2^{-31}>$ or $<-1; 1-2^{-15}>$ range representing the Q1.31 or Q1.15 format.

The Automotive Math and Motor Control Library Set for NXP MPC574xP devices was tested using two different test methods. To test the precision of each function implementation, the testing based on Matlab reference models was used. This release was tested using the Matlab R2014a version. To test the implementation on the embedded side, the target-in-loop testing was performed on the Automotive Math and Motor Control Library Set for NXP MPC574xP devices. This release was tested using the SFIO toolbox version 2.2 and Matlab R2014a version.

3.3 Multiple Implementation Support

In order to allow the user to utilize arbitrary implementation within one user application without any limitations, three different function call methods are supported in the Automotive Math and Motor Control Library Set for NXP MPC574xP devices:

- Global configuration option
- Additional parameter option
- API postfix option

Each of these method calls the API postfix function. Thus, for each implementation (32-bit fixed-point, 16-bit fixed point and single precision floating point) only one function is available within the package. This approach is based on ANSI-C99 ISO/IEC 9899:1999 function overloading.

By default the support of all implementations is turned off, thus the error message *"Define at least one supported implementation in SWLIBS_Config.h file."* is displayed during the compilation if no implementation is selected, preventing the user application building. Following are the macro definitions enabling or disabling the implementation support:

- **SWLIBS_SUPPORT_F32** for 32-bit fixed-point implementation support selection
- **SWLIBS_SUPPORT_F16** for 16-bit fixed-point implementation support selection
- **SWLIBS_SUPPORT_FLT** for single precision floating-point implementation support selection

These macros are defined in the SWLIBS_Config.h file located in Common directory of the Automotive Math and Motor Control Library Set for NXP MPC574xP devices installation destination. To enable the support of each individual implementation the relevant macro definition has to be set to *SWLIBS_STD_ON*.

3.3.1 Global Configuration Option

This function call supports the user legacy applications, which were based on older version of Motor Control Library. Prior to any Automotive Math and Motor Control Library Set for NXP MPC574xP devices function call using the Global configuration option, the *SWLIBS_DEFAULT_IMPLEMENTATION* macro definition has to be setup properly. This macro definition is not defined by default thus the error message "*Define default implementation in SWLIBS_Config.h file.*" is displayed during the compilation, preventing the user application building.

The *SWLIBS_DEFAULT_IMPLEMENTATION* macro is defined in the *SWLIBS_Config.h* file located in Common directory of the Automotive Math and Motor Control Library Set for NXP MPC574xP devices installation destination. The *SWLIBS_DEFAULT_IMPLEMENTATION* can be defined as the one of the following supported implementations:

- **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

After proper definition of *SWLIBS_DEFAULT_IMPLEMENTATION* macro the Automotive Math and Motor Control Library Set for NXP MPC574xP devices functions can be called using standard legacy API convention, e.g. *GFLIB_Sin(x)*.

For example if the *SWLIBS_DEFAULT_IMPLEMENTATION* macro definition is set to *SWLIBS_DEFAULT_IMPLEMENTATION_F32*, the 32-bit fixed-point implementation of sine function is invoked after the *GFLIB_Sin(x)* API call. Note that all standard legacy API calls will invoke the 32-bit fixed-point implementation in this example.

NOTE

As the Automotive Math and Motor Control Library Set for NXP MPC574xP devices supports the global configuration option, it is highly recommended to copy the SWLIBS_Config.h file to your local structure and refer the configuration to this local copy. This approach will prevent the incorrect setup of default configuration option, in case multiple projects with different default configuration are used.

3.3.2 Additional Parameter Option

In order to support the free selection of used implementation in the user application while keeping the function name same as in standard legacy API approach, the additional parameter option is implemented in the Automotive Math and Motor Control Library Set for NXP MPC574xP devices. In this option the additional parameter is used to distinguish which implementation shall be invoked. There are the following possible switches selecting the implementation:

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

For example, if the user application needs to invoke the 16-bit fixed-point implementation of sine function, the *GFLIB_Sin(x, F16)* API call needs to be used. Note that there is a possibility to call any implementation of the functions in user application without any limitation.

3.3.3 API Postfix Option

In order to support the free selection of used implementation in the user application while keeping the number of parameters same as in standard legacy API approach, the API postfix option is implemented in the Automotive Math and Motor Control Library Set for NXP MPC574xP devices. In this option the implementation postfix is used to distinguish which implementation shall be invoked. There are the following possible API postfixes selecting the implementation:

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

For example, if the user application needs to invoke the 32-bit implementation of sine function, the *GFLIB_Sin_F32* API call needs to be used. Note that there is a possibility to call any implementation of the functions in user application without any limitation.

3.4 Supported Compilers

The Automotive Math and Motor Control Library Set for NXP MPC574xP devices 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 version 2015.1.4
2. Wind River Compiler version 5.9.4.8
3. S32 Design Studio for Power Architecture (GCC) v1.2

The library is delivered in a library module "*MPC574xP_AMMCLIB.a*" for the Green Hills compiler. The library module is located in "*C:\Freescale\AMMCLIB\MPC574xP_AMMCLIB_v1.1.9\lib\ghs*" folder (considering the default installation path).

For the Wind River compiler, the "*MPC574xP_AMMCLIB.a*" library module is delivered within the installation package. The library module is located in "*C:\Freescale\AMMCLIB\MPC574xP_AMMCLIB_v1.1.9\lib\diab*" folder (considering the default installation path).

The library is delivered in a library module "*MPC574xP_AMMCLIB.a*" for the S32 Design Studio for Power Architecture IDE. The library module is located in "*C:\Freescale\AMMCLIB\MPC574xP_AMMCLIB_v1.1.9\lib\s32ds_ppc*" folder (considering the default installation path).

Together with the pre-compiled library modules and compiler-supplied headers (see library integration section), 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.

3.5 Matlab Integration

In addition to the Automotive Math and Motor Control Library Set for NXP MPC574xP devices library modules, the Bit Accurate Models (BAM) are delivered in the installation package. These models can be used in the Matlab Simulink Toolbox to model the behavior of each function in real implementation. As there are two versions of Matlab environment depending on the operation system, both 32-bit and 64-bit Bit Accurate Models are supported. Each model consists of these four files:

1. <libID>_<funcID>_BAM_<implementation>.mdl (e.g. *GFLIB_Acos_BAM_F32.mdl*): Contains the Bit Accurate Model (BAM) which can be included in the user Matlab Simulink model and refers to the S-function C file, and the S-function executable file.
2. <libID>_<funcID>_SF_<implementation>.c (e.g. *GFLIB_Acos_SF_F32.c*): The S-function C file that calls a simple legacy function during the simulation.
3. <libID>_<funcID>_SF_<implementation>.mexw32 (e.g. *GFLIB_Acos_SF_F32.mexw32*): Contains the compiled MATLAB S-function executable file created from the function C source file compiled for 32-bit Matlab environment.
4. <libID>_<funcID>_SF_<implementation>.mexw64 (e.g. *GFLIB_Acos_SF_F32.mexw64*): Contains the compiled MATLAB S-function executable file created from the function C source file compiled for 64-bit Matlab environment.

All delivered functions are provided with Bit Accurate Models in all applicable implementation versions and were compiled using Matlab 2014a and Windows SDK v7.1 in case of 64-bit models and using LCC compiler in case of 32-bit Bit Accurate Models. The user is thus responsible for selecting appropriate version of the Bit Accurate Model considering also the version of the Matlab environment. To include the Bit Accurate Model in the user Simulink scheme, simply copy the BAM model into the Simulink scheme. All Matlab/Simulink models are available as a standalone model in the appropriate library in the "bam" folder or, to make work easier, in one single model library "ammclib_bam.mdl" file which can be used for Matlab/Simulink scheme creation. The BAM models can be pulled out from this unified model and can be easily placed into your Matlab/Simulink scheme. Note that the BAM parameters need to be properly set up. The structure of the Matlab files delivered with the Automotive Math and Motor Control Library Set for NXP MPC574xP devices is depicted in Figure 3-2.

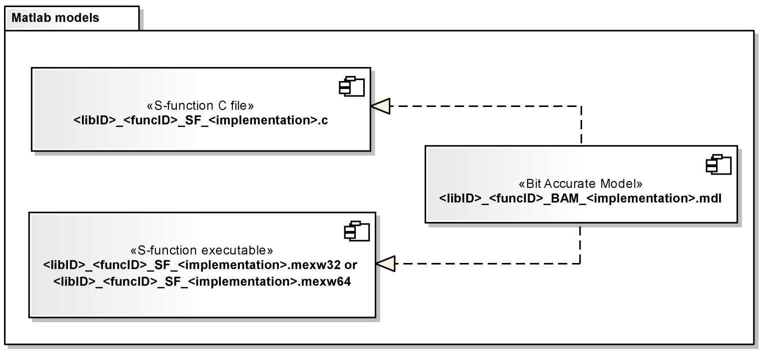

Figure 3-2. AMMCLib Matlab file structure

An example of Matlab Simulink model utilizing the Automotive Math and Motor Control Library Set for NXP MPC574xP devices Bit Accurate Models is depicted in Figure 3-3. Note that this schema is for the illustration only.

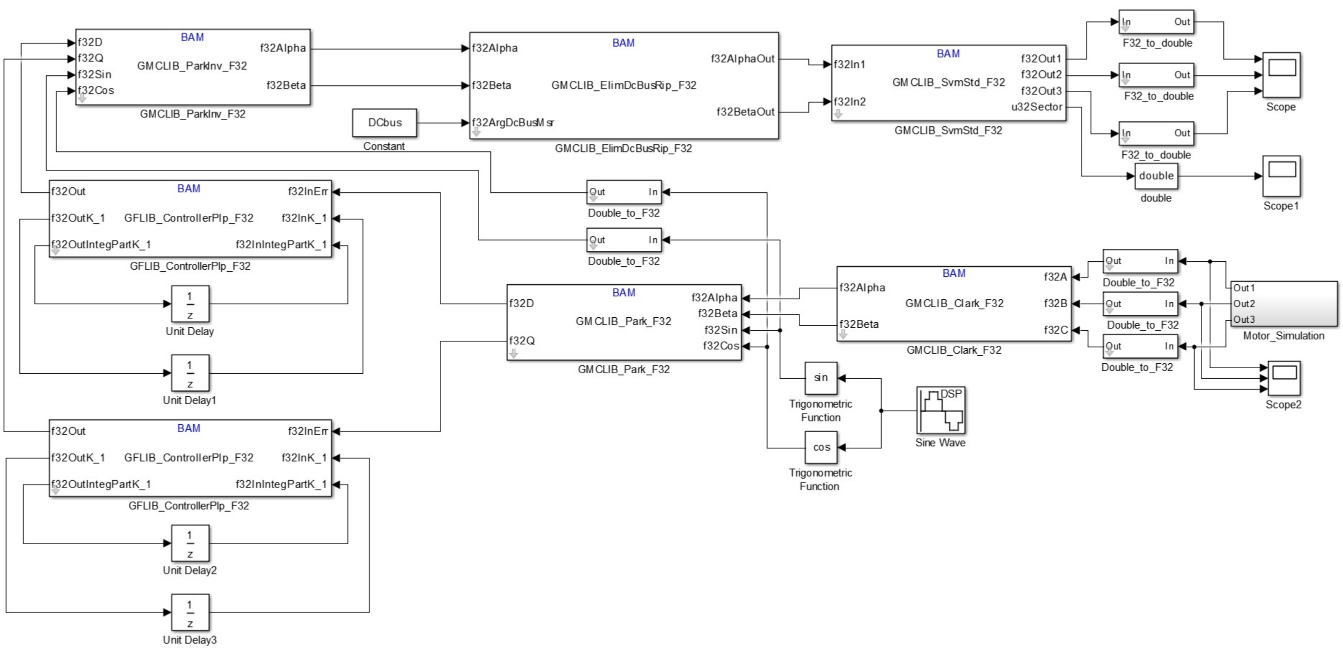

Figure 3-3. Example of Matlab Simulink model utilizing the Bit Accurate Models

3.6 Installation

The Automotive Math and Motor Control Library Set for NXP MPC574xP devices is delivered as a single executable file. To install the Automotive Math and Motor Control Library Set for NXP MPC574xP devices on a user computer, it is necessary to run the installation file and follow these steps:

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

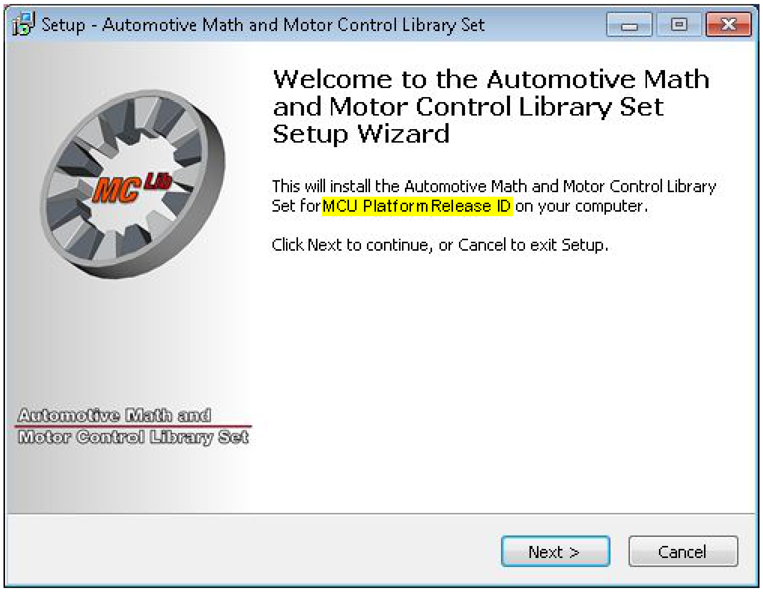

Figure 3-4. AMMCLib installation - step 1. Highlighted "MCU Platform" and "ReleaseID" identifies the actual release, which is the MPC574xP_AMMCLIB_v1.1.9

2. Accept the license agreement

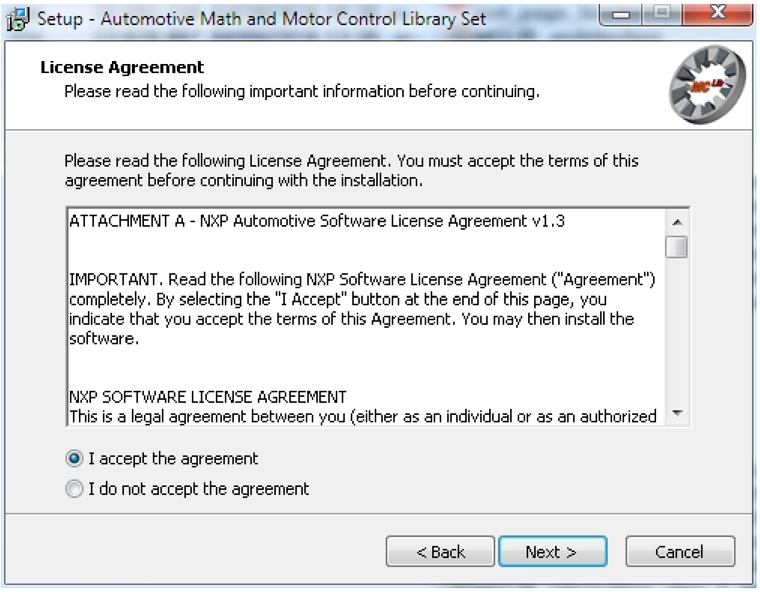

Figure 3-5. AMMCLib installation - step 2

3. Select the destination directory

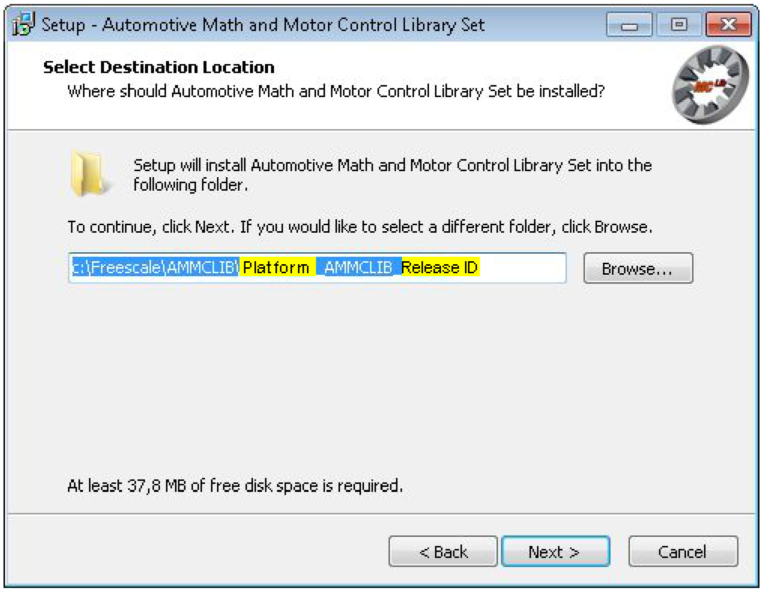

Figure 3-6. AMMCLib installation - step 3. Highlighted "Platform" and "ReleaseID" identifies the actual release installation path, which is the MPC574xP_AMMCLIB_v1.1.9

4. Check the destination directory and confirm the installation

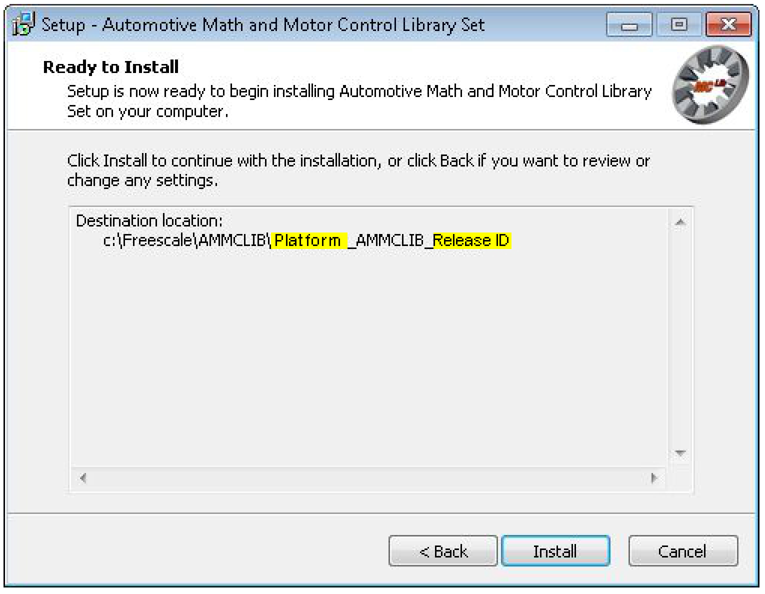

Figure 3-7. AMMCLib installation - step 4. Highlighted "Platform" and "ReleaseID" identifies the actual release installation path, which is the MPC574xP_AMMCLIB_v1.1.9

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

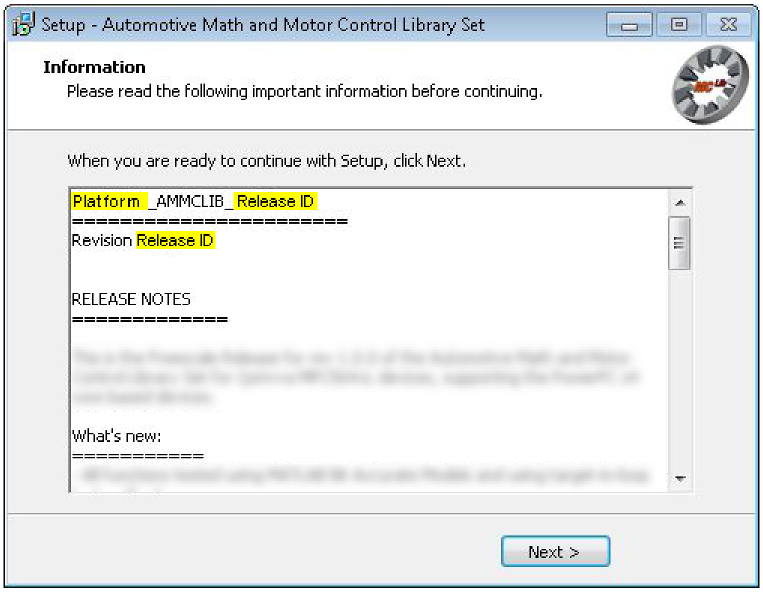

Figure 3-8. AMMCLib installation - step 5. Highlighted "Platform" and "ReleaseID" identifies the actual release, which is the MPC574xP_AMMCLIB_v1.1.9

6. Select Finish to end the installation

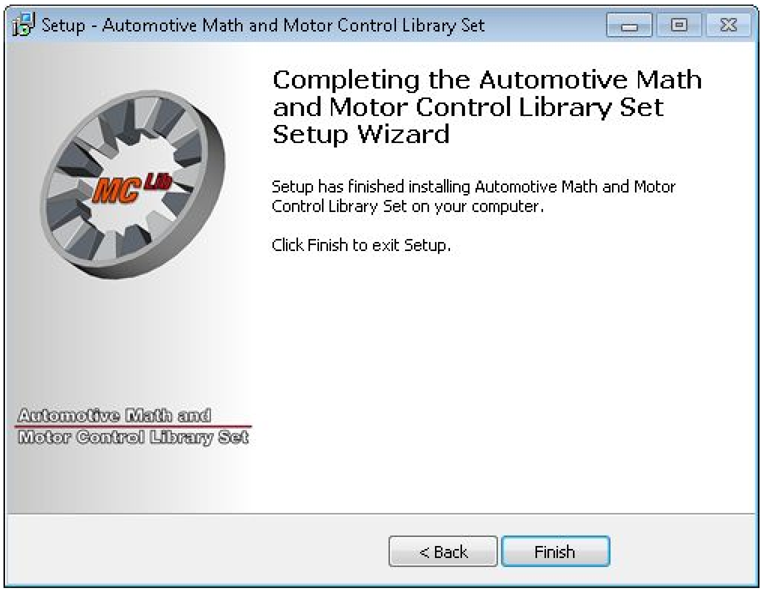

Figure 3-9. MCLib installation - step 6

3.7 Library File Structure

After a successful installation, the Automotive Math and Motor Control Library Set for NXP MPC574xP devices is added by default into the "*C:\Freescale\AMMCLIB \MPC574xP_AMMCLIB_v1.1.9*" subfolder. This folder will contain other nested subfolders and files required by the Automotive Math and Motor Control Library Set for NXP MPC574xP devices, as shown in Figure 3-10.

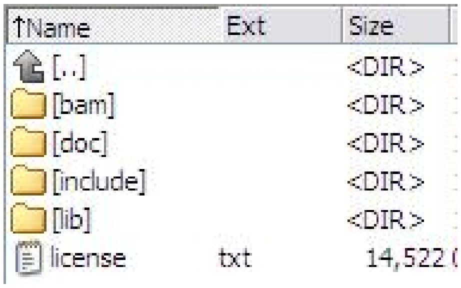

Figure 3-10. AMMCLIB directory structure

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

In order to integrate the Automotive Math and Motor Control Library Set for NXP MPC574xP devices into a new Green Hills compiler based project, the steps described in Library Integration into a Green Hills Multi Development Environment section must be performed.

In order to integrate the Automotive Math and Motor Control Library Set for NXP MPC574xP devices into a new Wind River Compiler based project, the steps described in section Library Integration into a Wind River Compiler Environment must be performed.

For integration of the Automotive Math and Motor Control Library Set for NXP MPC574xP devices into a new S32 Design Studio for Power Architecture based project, the steps described in section Library Integration into a S32 Design Studio for Power Architecture IDE must be performed.

The header files structure of the Automotive Math and Motor Control Library Set for NXP MPC574xP devices is depicted in Figure 3-11.

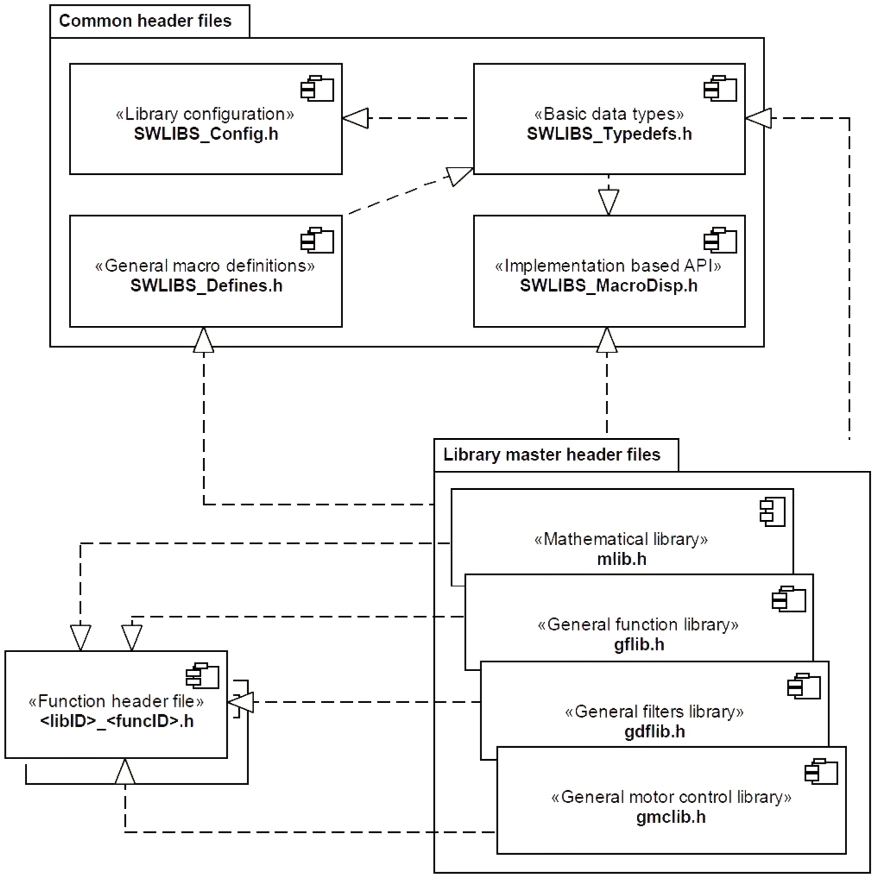

Figure 3-11. AMMCLIB header file structure

3.8 Integration Assumption

In order to successfully integrate the Automotive Math and Motor Control Library Set for NXP MPC574xP devices 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).
3. The pre-compiled library of the Automotive Math and Motor Control Library Set for NXP MPC574xP devices was compiled using the VLE option, thus the VLE has to be enabled in case the pre-compiled version of the Automotive Math and Motor Control Library Set for NXP MPC574xP devices is used in the user application and the VLE option is available on target MCU.

3.9 Library Integration into a Green Hills Multi Development Environment

The Automotive Math and Motor Control Library Set for NXP MPC574xP devices 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 3-12

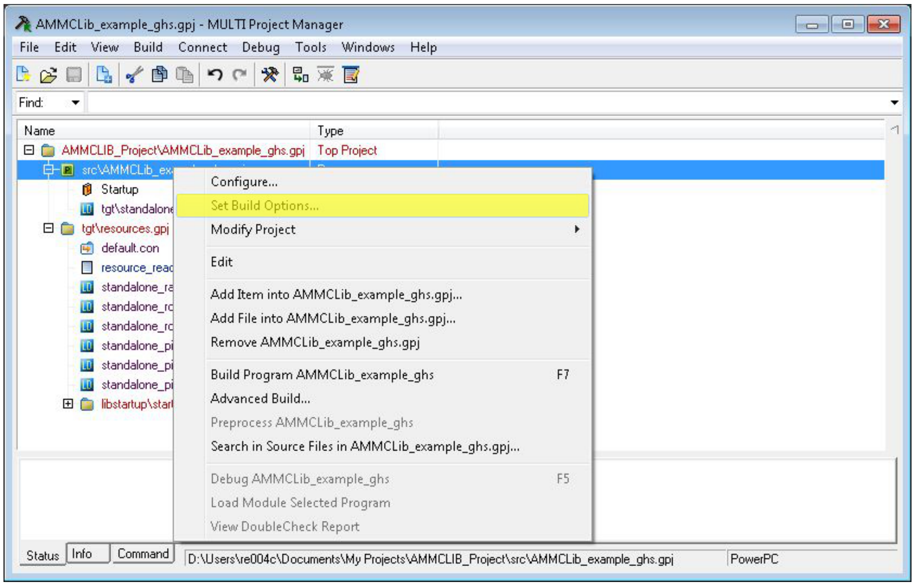

Figure 3-12. Project build options

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 builder should look for all the project header files, including the library header files, shall be specified as shown in Figure 3-13. Considering default settings, the following path shall be added: "*C:\Freescale\AMMCLIB \MPC574xP_AMMCLIB_v1.1.9\include.* "

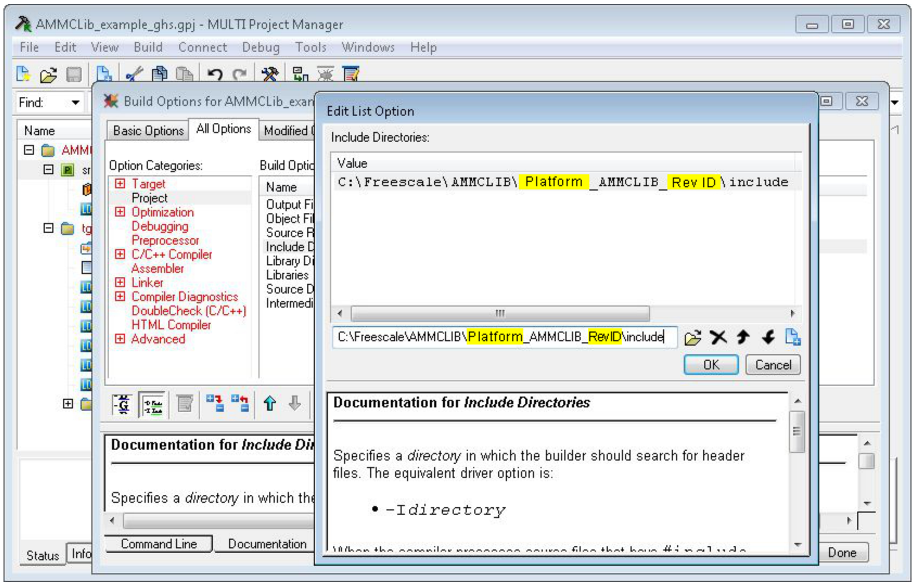

Figure 3-13. Adding a path to the library header files

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 3-14.

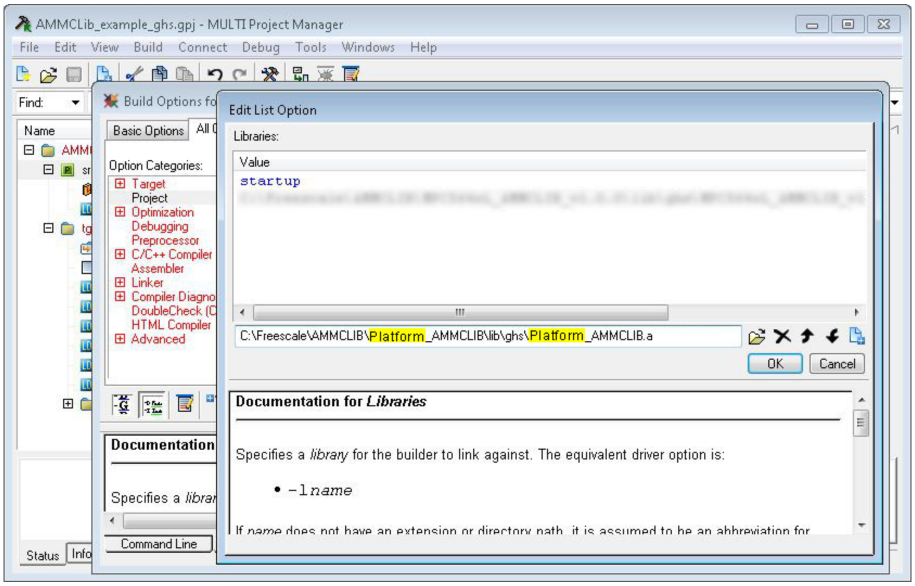

Figure 3-14. Adding a path to the library object files

Considering default settings, you should add *"C:\Freescale\AMMCLIB \MPC574xP_AMMCLIB_v1.1.9\lib\ghs\MPC574xP_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 *amclibgdflib,gflib,gmclib* depending on which library is to be employed.

The master header files contain several additional header files that are needed for the Automotive Math and Motor Control Library Set for NXP MPC574xP devices 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 implementation based API call.

NOTE

Remember that by default there is no default implementation selected in the "*SWLIBS_Config.h*" thus the error message will be displayed during the compilation requesting the default implementation selection.

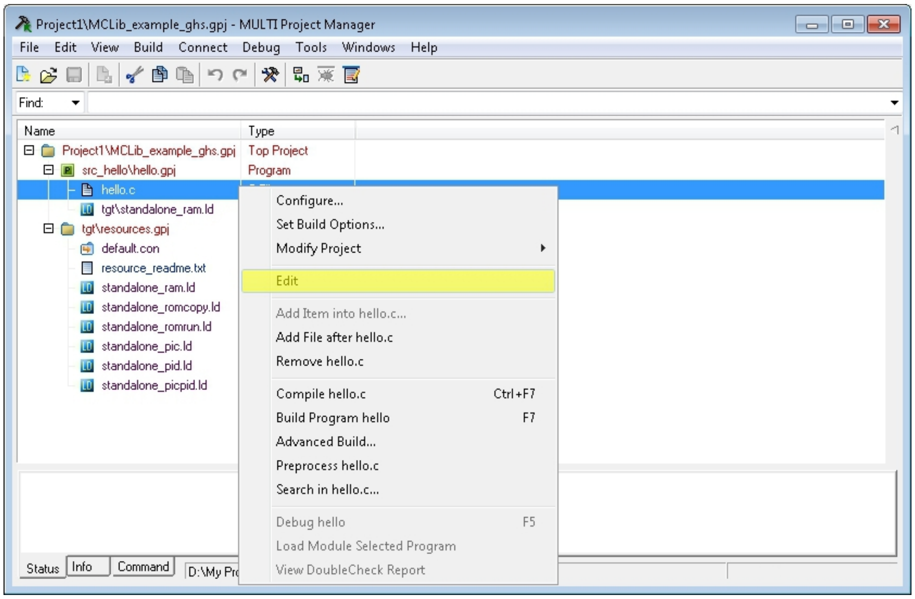

Figure 3-15. Opening the C editor for editing the application source code

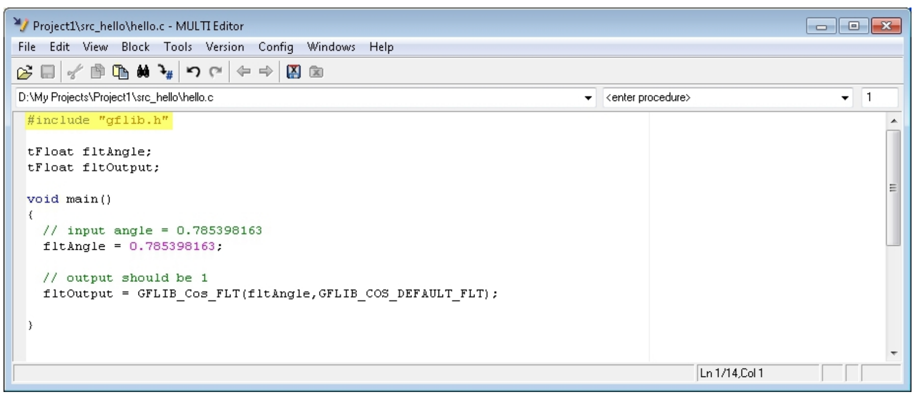

Figure 3-16. Using the pre-processor directive to include library master header files

At this point, the Automotive Math and Motor Control Library Set for NXP MPC574xP devices is linked with the user project file, and hence the library functions can be exploited and flawlessly compiled/linked with the user application.

3.10 Library Integration into a Wind River Compiler Environment

As the Wind River Compiler is provided with the Eclipse based IDE called Wind River Workbench, the following instruction will describe the steps needed to add the Automotive Math and Motor Control Library Set for NXP MPC574xP devices into a new project. All described steps assume that there is an existing Wind River Workbench project ("*MCLib_example_diab*" name is used in the pictures below).

In order to integrate the Automotive Math and Motor Control Library Set for NXP MPC574xP devices into an existing Wind River Workbench project, it is necessary to provide the Wind River Workbench IDE with access paths to the Automotive Math and Motor Control Library Set for NXP MPC574xP devices files. The following files shall be added to the user project:

- Library binary files located in the directory *"C:\Freescale\AMMCLIB \MPC574xP_AMMCLIB_v1.1.9\lib\diab"* (note: this is the default location and may be modified during library installation)
- Header files located in the directory *"C:\Freescale\AMMCLIB \MPC574xP_AMMCLIB_v1.1.9\include"* (note: this is the default location and may be modified during library installation)
- Compiler header files located in the directory *"C:\WindRiver\diab\5.9.4.8\include \diab"* (note: this is the default location and may be modified during compiler installation)

To add these files, choose your project in <*Project Explorer*> in the left tab, and from the <*Project*> menu select <*Properties*> as described in Figure 3-17.

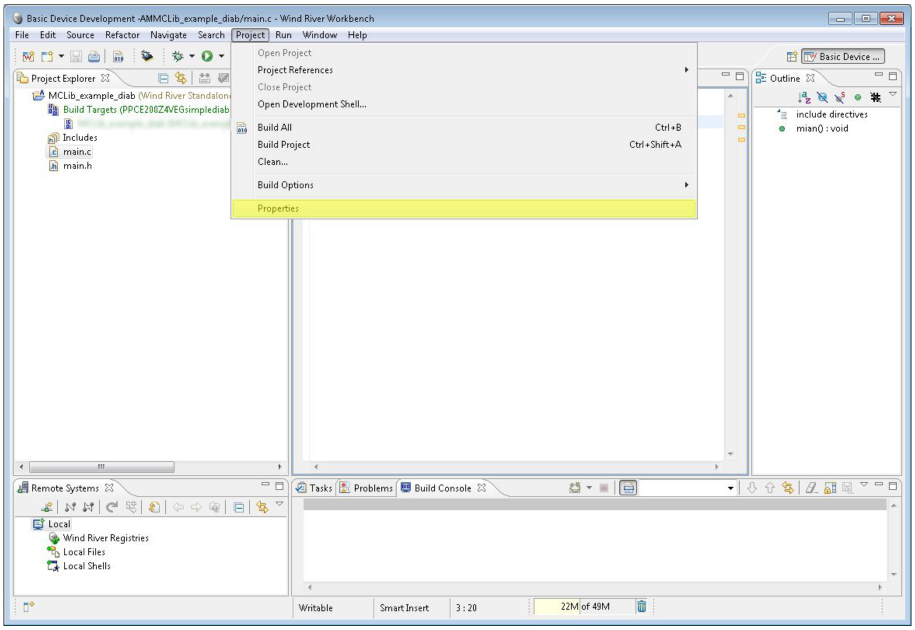

Figure 3-17. Selection of the project property in the Wind River Workbench

In the *Properties* window, select *Build Properties* from the left-hand list and choose the *Build Path* tab as described in Figure 3-18.

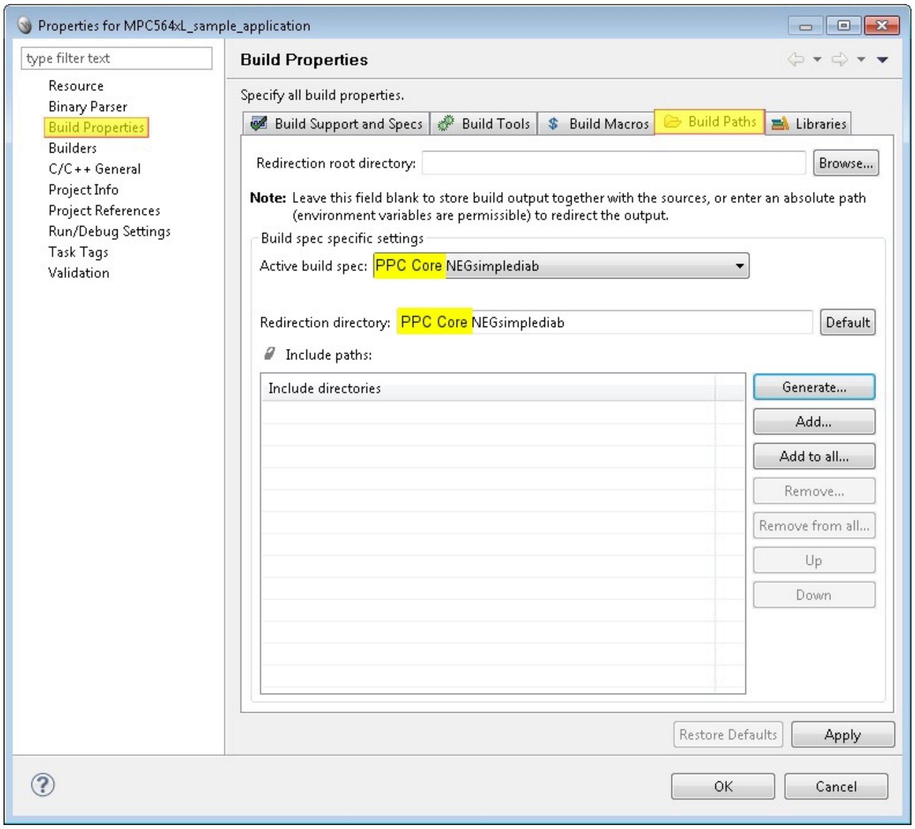

Figure 3-18. Selecting the include paths in the Wind River Workbench

By selecting the *<Add..>* button, add the directories, where the builder should look for all the header files, including the library file. Considering the default settings, the paths shown in Figure 3-19 and Figure 3-20 shall be added.

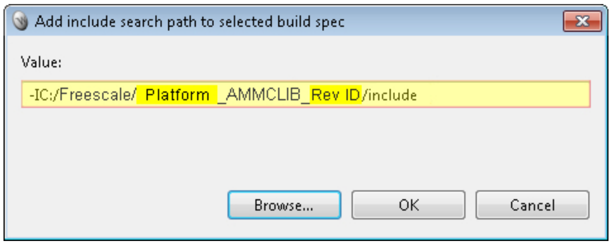

Figure 3-19. Including the header files directory to the Wind River Workbench

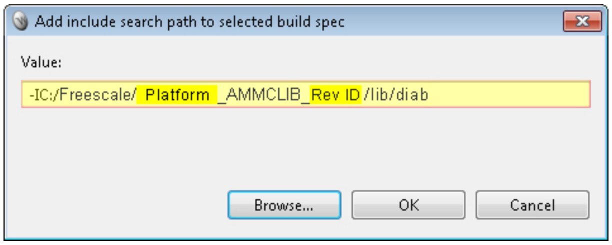

Figure 3-20. Including the library file directory to the Wind River Workbench

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 *amclibgdflib,gflib,gmclib* depending on which library is to be employed.

The master header files contain several additional header files that are needed for the Automotive Math and Motor Control Library Set for NXP MPC574xP devices 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 implementation based API call.

NOTE

Remember that by default there is no default implementation selected in the "*SWLIBS_Config.h*" thus the error message will be displayed during the compilation requesting the default implementation selection.

At this point, the Automotive Math and Motor Control Library Set for NXP MPC574xP devices is linked with the user project file, and hence the library functions can be exploited and flawlessly compiled/linked with the user application.

3.11 Library Integration into a S32 Design Studio for Power Architecture IDE

In order to use the Automotive Math and Motor Control Library Set for NXP MPC574xP devices within a S32 Design Studio for Power Architecture GUI environment, it is necessary to provide the S32 Design Studio GUI with access paths to the Automotive Math and Motor Control Library Set for NXP MPC574xP devices files. The following files shall be added to the user project:

- Library binary files located in the directory *"C:\Freescale\AMMCLIB \MPC574xP_AMMCLIB_v1.1.9\lib\s32ds_ppc"* (note: this is the default location and may be modified during library installation)
- Header files located in the directory *"C:\Freescale\AMMCLIB \MPC574xP_AMMCLIB_v1.1.9\include"* (note: this is the default location and may be modified during library installation)

To add these files, select your project in the <*Project Explorer*> in the left tab, and from the <*Project*> menu select the <*Properties*> option as described in Figure 3-21.

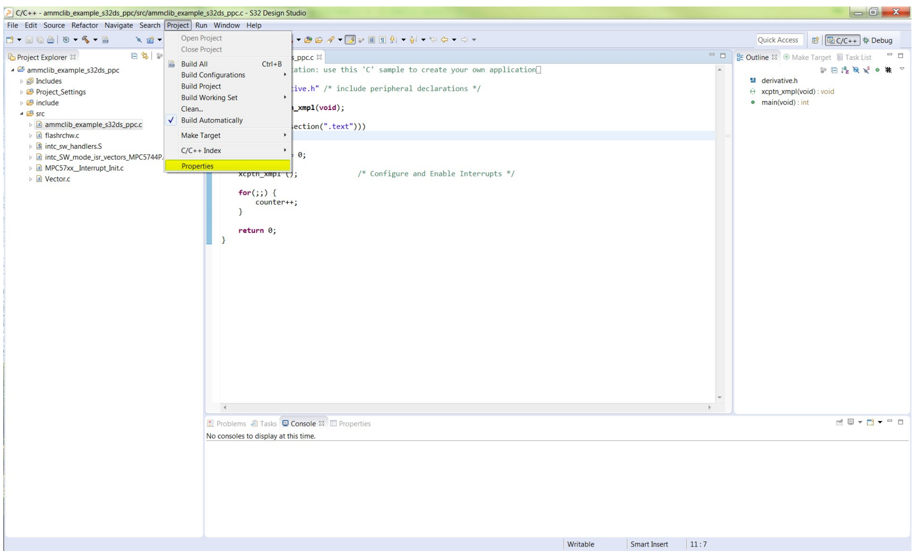

Figure 3-21. Selection of the project property in the S32 Design Studio for Power Architecture IDE

In the <*Properties*> window, select the <*C/C++ General*><*Paths and Symbols*> from the left-hand list and choose the <*Includes*> tab and <*GNU C*> under the <*Languages*> section, as described in Figure 3-22.

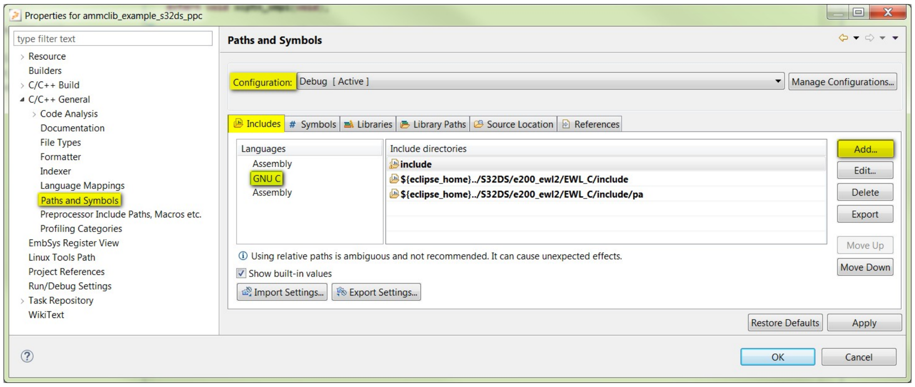

Figure 3-22. Selecting the include paths in the S32 Design Studio for Power Architecture IDE

By selecting the <*Add..*> button, add the directories, where the builder should look for all the header files. Considering the default settings, the path shown in Figure 3-23 shall be added.

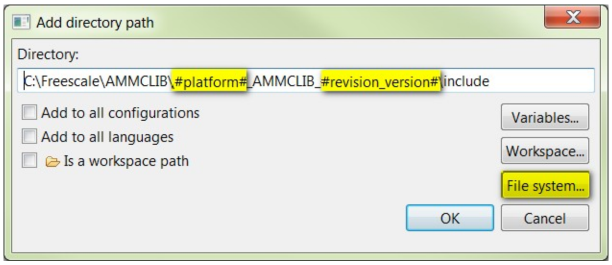

Figure 3-23. Adding the header files path to the S32 Design Studio for Power Architecture IDE project

After adding of the Automotive Math and Motor Control Library Set for NXP MPC574xP devices header files path, the library object file shall be add to the S32 Design Studio for Power Architecture project. In the <*Properties*> window, select the <*C/C++ Build*><<*Standard S32DS C Linker*> from the left-hand list and choose the <*Libraries*>, as described in Figure 3-24. By selecting the <*Add..*> button, add the library object file. Be aware, that colon must be added as a prefix of the library name.

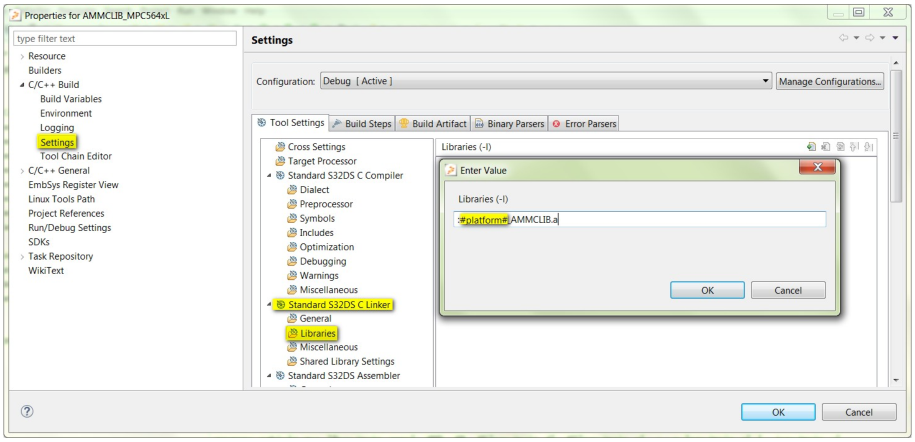

Figure 3-24. Adding the library object file to the S32 Design Studio for Power Architecture IDE project

By selecting the <*Add..*> button, add the library search path folder. Considering the default settings, the path shown in Figure 3-25 shall be added.

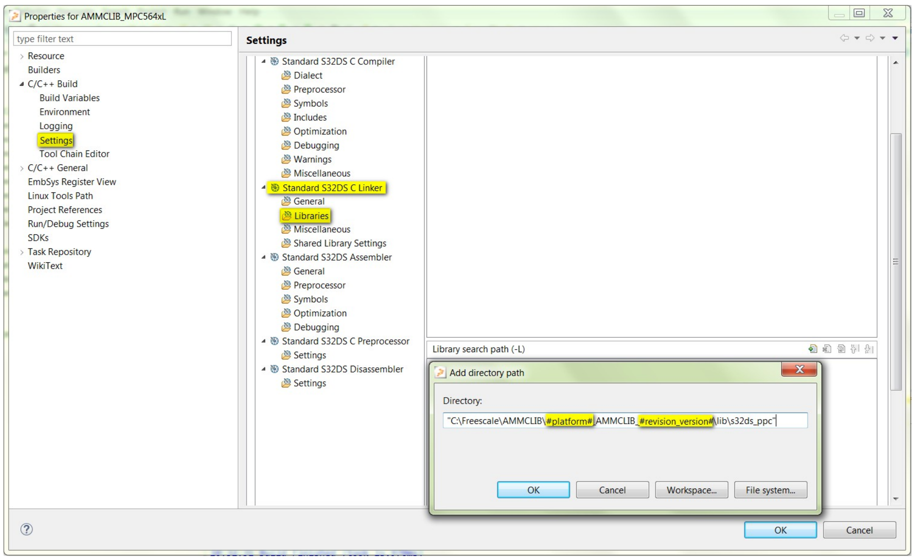

Figure 3-25. Adding the library search path folder to the S32 Design Studio for Power Architecture IDE project

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 *amclibgdflib,gflib,gmclibmlib,* depending on which library is to be employed.

The master header files contain several additional header files that are needed for the Automotive Math and Motor Control Library Set for NXP MPC574xP devices 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 implementation based API call.

NOTE

Remember that, there is no default implementation selected in the "*SWLIBS_Config.h*" by default, thus the error message will be displayed during the compilation requesting the default implementation selection.

At this point, the Automotive Math and Motor Control Library Set for NXP MPC574xP devices is linked with the user project file, and hence the library functions can be exploited and flawlessly compiled/linked with the user application.

3.12 Library Testing

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

- Matlab Simulink Toolbox based testing (refer to #d63e97a1310 for more details).
- Target-in-loop based testing (refer to #d63e123a1310 for detailed information).

The Figure 3-26 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 equivalent precision is the same for all supported implementations of the . For the 32-bit and 16-bit fixed-point implementations, the output precision of non-approximation function is greater or equal to +/-1 LSB in 16-bit arithmetic, and is greater or equal to +/-3 LSB in 16-bit arithmetic for approximation functions.

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

- Approximation function with precision +/-3 LSB:
    - **32-bit fixed point:** <PRECISE_VALUE> +/- 3*2^-15
    - **16-bit fixed point:** <PRECISE_VALUE> +/- 3*2^-15
- Non-approximation function with precision +/-1 LSB:
    - **32-bit fixed point:** <PRECISE_VALUE> +/- 2^-15
    - **16-bit fixed point:** <PRECISE_VALUE> +/- 2^-15

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 MPC574XPMCFLTACC.pdf.

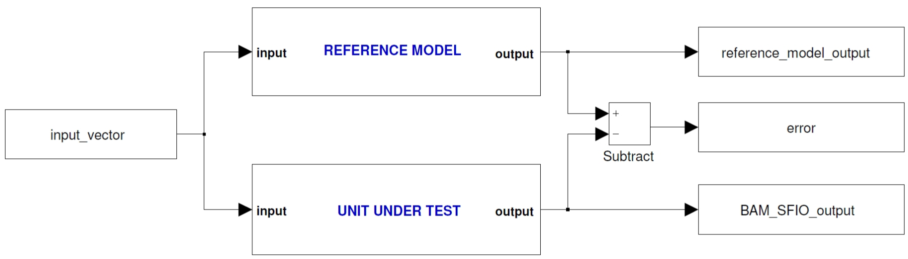

Figure 3-26. Principle of AMMCLIB testing

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 #d63e97a1310 and #d63e123a1310 describe the differences between AMMCLIB testing based on BAM models and target-in-loop testing based on SFIO models.

3.12.1 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 3-27). 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 3-28. 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 3-29.

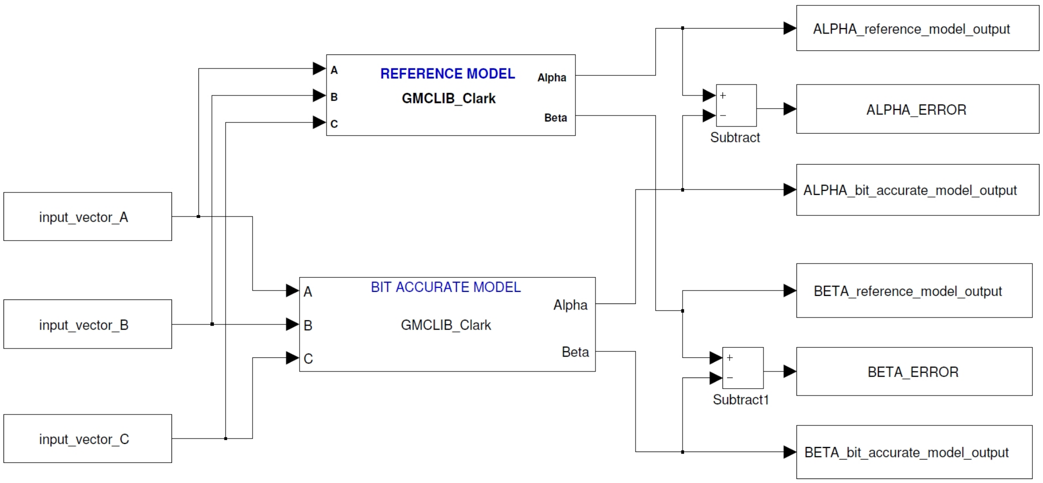

Figure 3-27. Testing of the GMCLIB_Clark function based on the MATLAB Simulink Toolbox

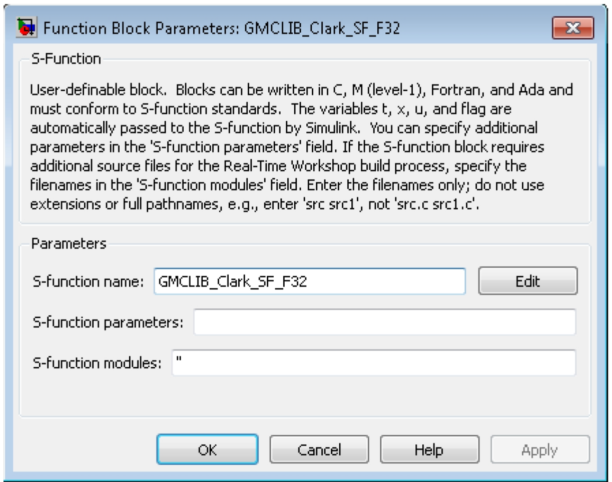

Figure 3-28. Bit Accurate Model parameters of the Clark transformation

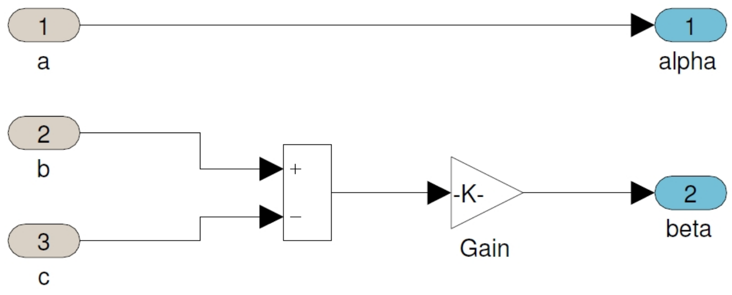

Figure 3-29. Reference Model of the GMCLIB_Clark function

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

The testing method in Figure 3-30 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 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.

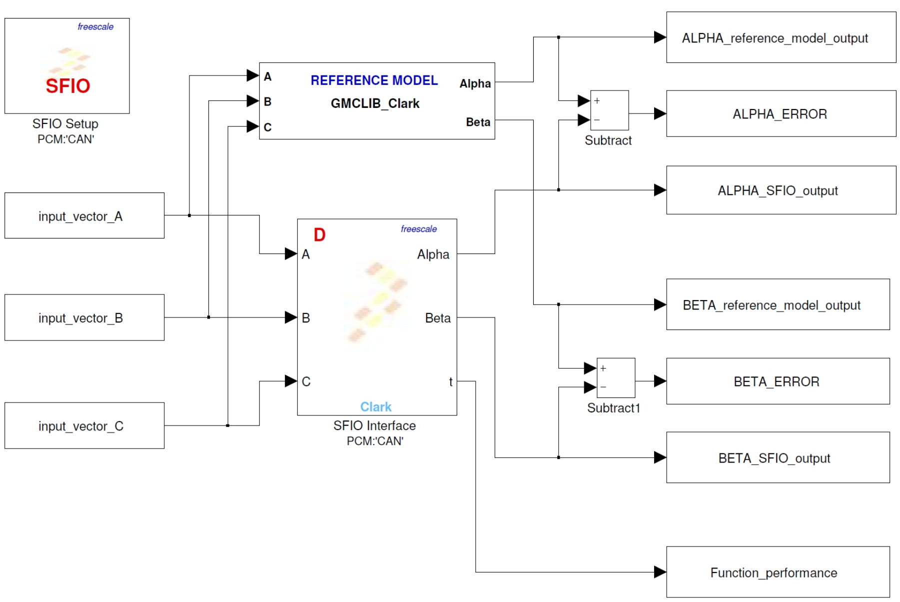

Figure 3-30. Target-in-loop testing example based on the SFIO Toolbox

3.13 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 Library Testing 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 MPC574XPMCFLTACC.pdf.

The following table describes the maximum absolute error of the fixed-point functions measured in the 16-bit LSB multiples:

Table 3-1. Maximum absolute error

Function name	F32 variant	F16 variant
GDFLIB_FilterFIR	3	3
GDFLIB_FilterIIR1	1	1
GDFLIB_FilterIIR2	1	1
GDFLIB_FilterMA	1	1
GFLIB_Acos	3	3
GFLIB_Asin	3	3
GFLIB_Atan	3	3
GFLIB_AtanYX	3	3
GFLIB_AtanYXShifted	3	3
GFLIB_ControllerPIp	3	3
GFLIB_ControllerPIpAW	3	3
GFLIB_ControllerPIr	3	3
GFLIB_ControllerPIrAW	3	3
GFLIB_Cos	3	3
GFLIB_Hyst	1	1
GFLIB_IntegratorTR	3	3
GFLIB_Limit	1	1
GFLIB_LowerLimit	1	1
GFLIB_Lut1D	3	3
GFLIB_Lut2D	3	3
GFLIB_Ramp	1	1
GFLIB_Sign	1	1

Table continues on the next page...

Table 3-1. Maximum absolute error (continued)

Function name	F32 variant	F16 variant
GFLIB_Sin	3	3
GFLIB_SinCos	3	3
GFLIB_Sqrt	1	3
GFLIB_Tan	3	3
GFLIB_UpperLimit	1	1
GFLIB_VectorLimit	3	3
GMCLIB_Clark	1	3
GMCLIB_ClarkInv	1	3
GMCLIB_DecouplingPMSM	1	3
GMCLIB_ElimDcBusRip	3	3
GMCLIB_Park	1	1
GMCLIB_ParkInv	1	1
GMCLIB_SvmStd	1	3
MLIB_Abs	1	1
MLIB_AbsSat	1	1
MLIB_Add	1	1
MLIB_AddSat	1	1
MLIB_Convert_F16F32	N/A	1
MLIB_Convert_F16FLT	N/A	1
MLIB_Convert_F32F16	1	N/A
MLIB_Convert_F32FLT	1	N/A
MLIB_Convert_FLTF16	N/A	N/A
MLIB_Convert_FLTF32	N/A	N/A
MLIB_ConvertPU_F16F32	N/A	1
MLIB_ConvertPU_F16FLT	N/A	1
MLIB_ConvertPU_F32F16	1	N/A
MLIB_ConvertPU_F32FLT	1	N/A
MLIB_ConvertPU_FLTF16	N/A	N/A
MLIB_ConvertPU_FLTF32	N/A	N/A
MLIB_Div	3	1
MLIB_DivSat	3	1
MLIB_Mac	1	1
MLIB_MacSat	1	1
MLIB_Mnac	1	1
MLIB_Msu	1	1
MLIB_Mul	1	1
MLIB_MulSat	1	1
MLIB_Neg	1	1
MLIB_NegSat	1	1

Table continues on the next page...

Table 3-1. Maximum absolute error (continued)

Function name	F32 variant	F16 variant
MLIB_Norm	1	1
MLIB_RndSat	1	1
MLIB_Round	1	1
MLIB_ShBi	1	1
MLIB_ShBiSat	1	1
MLIB_ShL	1	1
MLIB_ShLSat	1	1
MLIB_ShR	1	1
MLIB_Sub	1	1
MLIB_SubSat	1	1
MLIB_VMac	1	2
AMCLIB_BemfObsrvDQ	24	30
AMCLIB_TrackObsrv	3	3

Chapter 4

4.1 Function Index

Table 4-1. 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
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_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_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

Table continues on the next page...

Table 4-1. Quick function reference (continued)

Type	Name	Arguments
void	GDFLIB_FilterFIRInit_F16	const GDFLIB_FILTERFIR_PARAM_T_F16 *const pParam GDFLIB_FILTERFIR_STATE_T_F16 *const pState tFrac16 * pInBuf
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
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
tFrac16	GDFLIB_FilterMA_F16	tFrac16 f16In GDFLIB_FILTER_MA_T_F16 * pParam

Table continues on the next page...

Table 4-1. Quick function reference (continued)

Type	Name	Arguments
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
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
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
tFrac16	**GFLIB_ControllerPIp_F16**	tFrac16 f16InErr

Table continues on the next page...

Table 4-1. Quick function reference (continued)

Type	Name	Arguments
		GFLIB_CONTROLLER_PI_P_T_F16 *const pParam
tFrac32	**GFLIB_ControllerPIp_F32**	tFrac32 f32InErr GFLIB_CONTROLLER_PI_P_T_F32 *const pParam
tFloat	**GFLIB_ControllerPIp_FLT**	tFloat fltInErr GFLIB_CONTROLLER_PI_P_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
tFrac16	**GFLIB_ControllerPIr_F16**	tFrac16 f16InErr GFLIB_CONTROLLER_PI_R_T_F16 *const pParam
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
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
tFloat	**GFLIB_Log10_FLT**	tFloat fltIn const GFLIB_LOG10_T_FLT *const pParam

Table continues on the next page...

Table 4-1. Quick function reference (continued)

Type	Name	Arguments
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
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

Table continues on the next page...

Table 4-1. Quick function reference (continued)

Type	Name	Arguments
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
tBool	**GFLIB_VectorLimit_F16**	const SWLIBS_2Syst_F16 *const pIn SWLIBS_2Syst_F16 *const pOut const GFLIB_VECTORLIMIT_T_F16 *const pParam
tBool	**GFLIB_VectorLimit_F32**	const SWLIBS_2Syst_F32 *const pIn SWLIBS_2Syst_F32 *const pOut const GFLIB_VECTORLIMIT_T_F32 *const pParam
tBool	**GFLIB_VectorLimit_FLT**	const SWLIBS_2Syst_FLT *const pIn SWLIBS_2Syst_FLT *const pOut const GFLIB_VECTORLIMIT_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
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

Table continues on the next page...

Table 4-1. Quick function reference (continued)

Type	Name	Arguments
		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_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
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_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
INLINE tFrac16	**MLIB_AbsSat_F16**	register tFrac16 f16In
INLINE tFrac32	**MLIB_AbsSat_F32**	register tFrac32 f32In
INLINE tFrac16	**MLIB_Abs_F16**	register tFrac16 f16In

Table continues on the next page...

Table 4-1. Quick function reference (continued)

Type	Name	Arguments
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
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 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

Table continues on the next page...

Table 4-1. Quick function reference (continued)

Type	Name	Arguments
		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
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

Table continues on the next page...

Table 4-1. Quick function reference (continued)

Type	Name	Arguments
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
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

Table continues on the next page...

Table 4-1. Quick function reference (continued)

Type	Name	Arguments
		register tFrac32 f32In2
INLINE tFloat	**MLIB_Sub_FLT**	register tFloat fltIn1 register tFloat 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
const SWLIBS_VERSION_T *	**SWLIBS_GetVersion**	void

Chapter 5
API References

This section describes in details the Application Interface for all functions available in Automotive Math and Motor Control Library Set for NXP MPC574xP devices.

5.1 Function AMCLIB_BemfObsrvDQInit

This function initializes the AMCLIB_BemfObsrvDQ function.

5.1.1 Description

This function resets all state variables to zero.

Note

The input/output pointers must contain valid addresses, otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.1.2 Re-entrancy

The function is re-entrant for a different pCtrl.

5.1.3 Function AMCLIB_BemfObsrvDQInit_F32

5.1.3.1 Declaration

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

5.1.3.2 Arguments

Table 5-1. 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 coefficients.

5.1.3.3 Code Example

```
#include "amclib.h"

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

void main (void)
{
    // Initialize BEMF observer state variables
    AMCLIB_BemfObsrvDQInit_F32 (&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;

    while(1);
}
```

5.1.4 Function AMCLIB_BemfObsrvDQInit_F16

5.1.4.1 Declaration

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

5.1.4.2 Arguments

Table 5-2. 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 coefficients.

5.1.4.3 Code Example

```
#include "amclib.h"

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

void main (void)
{
    // Initialize BEMF observer state variables
    AMCLIB_BemfObsrvDQInit_F16 (&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;

    while(1);
}
```

5.1.5 Function AMCLIB_BemfObsrvDQInit_FLT

5.1.5.1 Declaration

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

5.1.5.2 Arguments

Table 5-3. 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 coefficients.

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

5.1.5.3 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;

void main (void)
{
    // Initialize BEMF observer state variables
    AMCLIB_BemfObsrvDQInit_FLT (&BemfObsrv);

    // Set BEMF observer parameters
    BemfObsrv.pParamD.fltCC1sc      =  Kp+Ki*Ts/2;
    BemfObsrv.pParamD.fltCC2sc      = -Kp+Ki*Ts/2;
    BemfObsrv.pParamD.fltUpperLimit =  1000.0F;
    BemfObsrv.pParamD.fltLowerLimit = -1000.0F;
    BemfObsrv.pParamQ.fltCC1sc      =  Kp+Ki*Ts/2;
    BemfObsrv.pParamQ.fltCC2sc      = -Kp+Ki*Ts/2;
    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);

    while(1);
}
```

5.2 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.

5.2.1 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). The BEMF observer is suitable for medium to high motor speeds.

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 5-1.

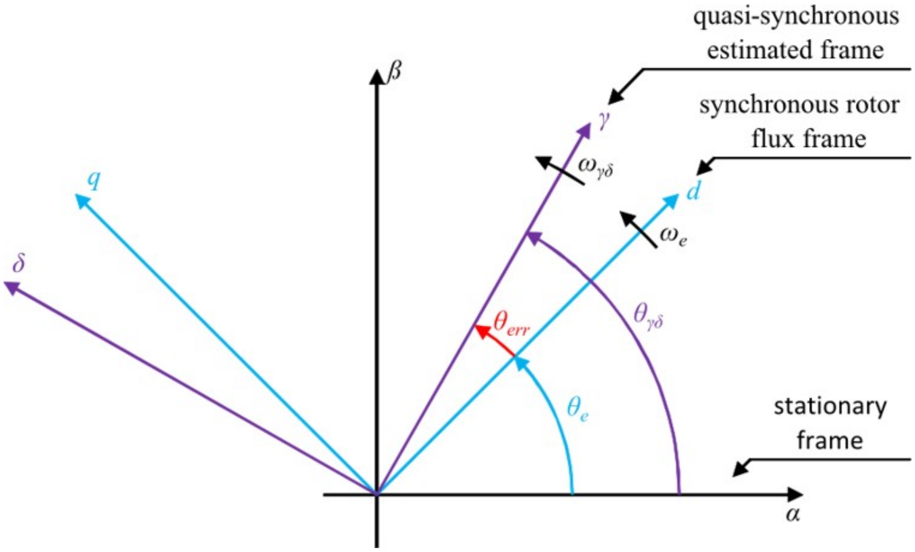

Figure 5-1. Rotor reference frames

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:

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

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.

Only the BEMF term depends on the phase error θ_{err} between the quasi-synchronous reference frame γ/δ and the synchronous rotor flux frame d/q. Since the saliency-based BEMF term is not modeled in the BEMF observer, the observer acts as a BEMF state filter. The extracted BEMF value is 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 5-2.

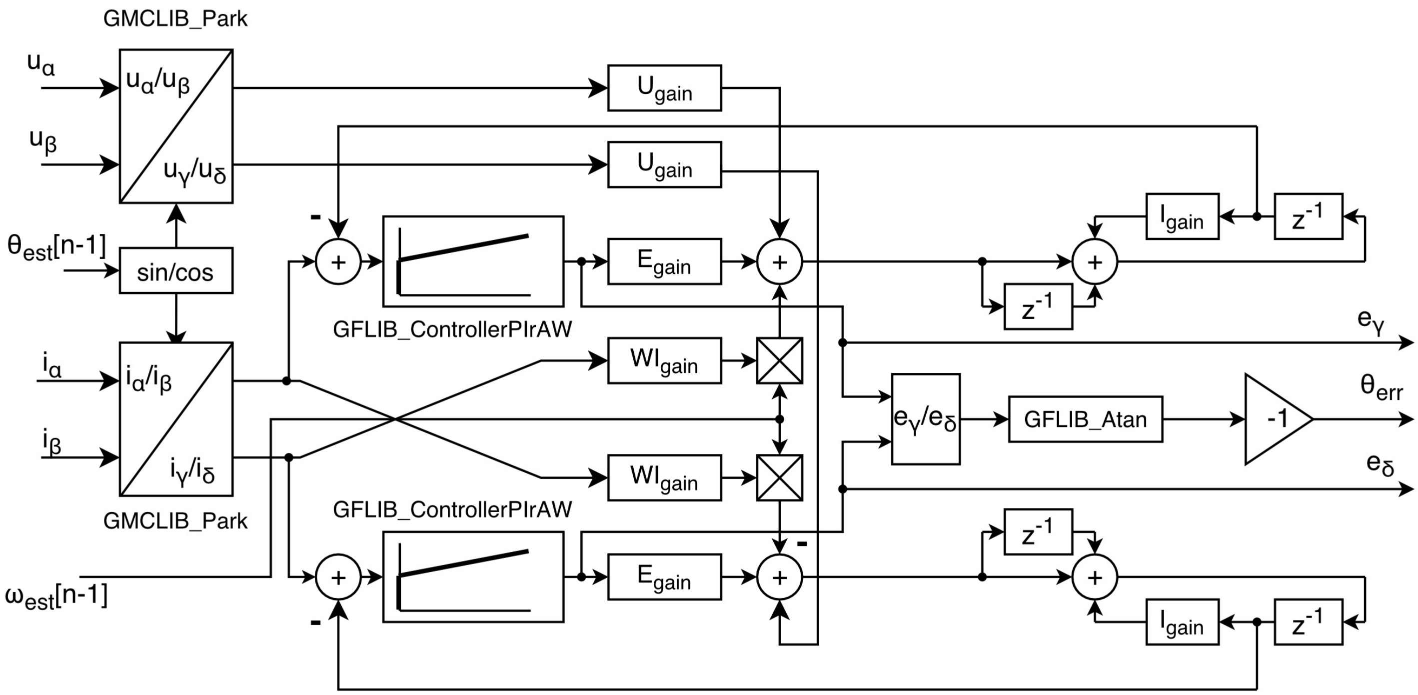

Figure 5-2. BEMF observer structure

The BEMF observer loop consists of a proportional-integral controller and an R-L circuit, which simulates a motor winding without the BEMF. Refer to the GFLIB_ControllerPIrAW_F16 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 by the following difference equations:

$$i_{dSC}(n) = (UGain \cdot u_{dSC}(n) + WIGain \cdot \omega_{SC}(n) i_{qSC}(n) + EGain \cdot e_{dSC}(n)) \cdot 2^{s16Shift} + IGain \cdot i_{dSC}(n-1) + x_d(n-1)$$

$$x_d(n-1) = (UGain \cdot u_{dSC}(n-1) + WIGain \cdot \omega_{SC}(n-1) i_{qSC}(n-1) + EGain \cdot e_{dSC}(n-1)) \cdot 2^{s16Shift}$$

$$i_{qSC}(n) = (UGain \cdot u_{qSC}(n) - WIGain \cdot \omega_{SC}(n) i_{dSC}(n) + EGain \cdot e_{qSC}(n)) \cdot 2^{s16Shift} + IGain \cdot i_{qSC}(n-1) + x_q(n-1)$$

$$x_q(n-1) = (UGain \cdot u_{qSC}(n-1) - WIGain \cdot \omega_{SC}(n-1) i_{dSC}(n-1) + EGain \cdot e_{qSC}(n-1)) \cdot 2^{s16Shift}$$

Equation **AMCLIB_BemfObsrvDQ_Eq2**

where the subscript *SC* indicates values scaled to the fractional range [-1, 1).

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).

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
- MC_TOOLBOX - Motor Control Development Toolbox
- FREEMASTER - FreeMASTER Run-Time Debugging Tool

Note

The input/output pointers must contain valid addresses, otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.2.2 Re-entrancy

The function is re-entrant for a different pCtrl.

5.2.3 Function AMCLIB_BemfObsrvDQ_F32

5.2.3.1 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);
```

5.2.3.2 Arguments

Table 5-4. 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 coefficients.

5.2.3.3 Return

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

5.2.3.4 Variant Specifics

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). The incorrect setting of the scaling constants may lead to an undesirable overflow or saturation during the computation. 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 5-5. Scaling constants

Scaling constant	Symbol	Calculation
Maximum stator phase voltage [V]	U_{MAX}	$U_{MAX}=U_{DC\_BUS}/\sqrt{3}$
Maximum phase current [A]	I_{MAX}	Maximum current depends on power stage capabilities.
Maximum required 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.

Parameters of the PIrAW controllers inside the BEMF observer can be calculated using the following equations:

$$pParamD.f32CC1sc = \left(K_P + \frac{K_I T_S}{2}\right) \cdot \frac{I_{MAX}}{E_{MAX}} \cdot 2^{-NShift}$$

$$pParamD.f32CC2sc = \left(-K_P + \frac{K_I T_S}{2}\right) \cdot \frac{I_{MAX}}{E_{MAX}} \cdot 2^{-NShift}$$

$$pParamD.u16NShift = NShift$$

$$pParamQ.f32CC1sc = \left(K_P + \frac{K_I T_S}{2}\right) \cdot \frac{I_{MAX}}{E_{MAX}} \cdot 2^{-NShift}$$

$$pParamQ.f32CC2sc = \left(-K_P + \frac{K_I T_S}{2}\right) \cdot \frac{I_{MAX}}{E_{MAX}} \cdot 2^{-NShift}$$

$$pParamQ.u16NShift = NShift$$

Equation **AMCLIB_BemfObsrvDQ_Eq3**

where T_S is the sampling interval, K_P is the proportional gain, and K_I is the integral gain. The upper and lower limits of the PIrAW controller should be set based on the expected dynamics of the system. *NShift* is the smallest nonnegative integer value that ensures that the controller coefficients fit in the fractional range [-1, 1). The gains 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_Eq4**

where ξ is the current loop attenuation, and ω_0 is the current loop natural frequency [rad/s]. Coefficients ξ and ω_0 should correspond to the values chosen for the FOC current loop.

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

$$f32IGain = \frac{2 \cdot L_d - T_S \cdot R_S}{2 \cdot L_d + T_S \cdot R_S}$$
$$f32UGain = \frac{T_S}{2 \cdot L_d + T_S \cdot R_S} \cdot \frac{U_{MAX}}{I_{MAX}} \cdot 2^{-NShiftRL}$$
$$f32WIGain = \frac{T_S \cdot L_q}{2 \cdot L_d + T_S \cdot R_S} \cdot \Omega_{MAX} \cdot 2^{-NShiftRL}$$
$$f32EGain = \frac{T_S}{2 \cdot L_d + T_S \cdot R_S} \cdot \frac{E_{MAX}}{I_{MAX}} \cdot 2^{-NShiftRL}$$
$$s16Shift = NShiftRL$$

Equation **AMCLIB_BemfObsrvDQ_Eq5**

NShiftRL is set to ensure that the gains fit in the fractional range [-1, 1).

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.e-4; % inductance in d-axis [H]
Lq = 3.e-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 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_F32 coefficients ---')
% PIrAW controller parameters
maxCoeff = max(abs([( Kp + Ki*Ts/2)*Imax/Umax,...
                    (-Kp + Ki*Ts/2)*Imax/Umax]));
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/Umax*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/Umax*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;
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 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.

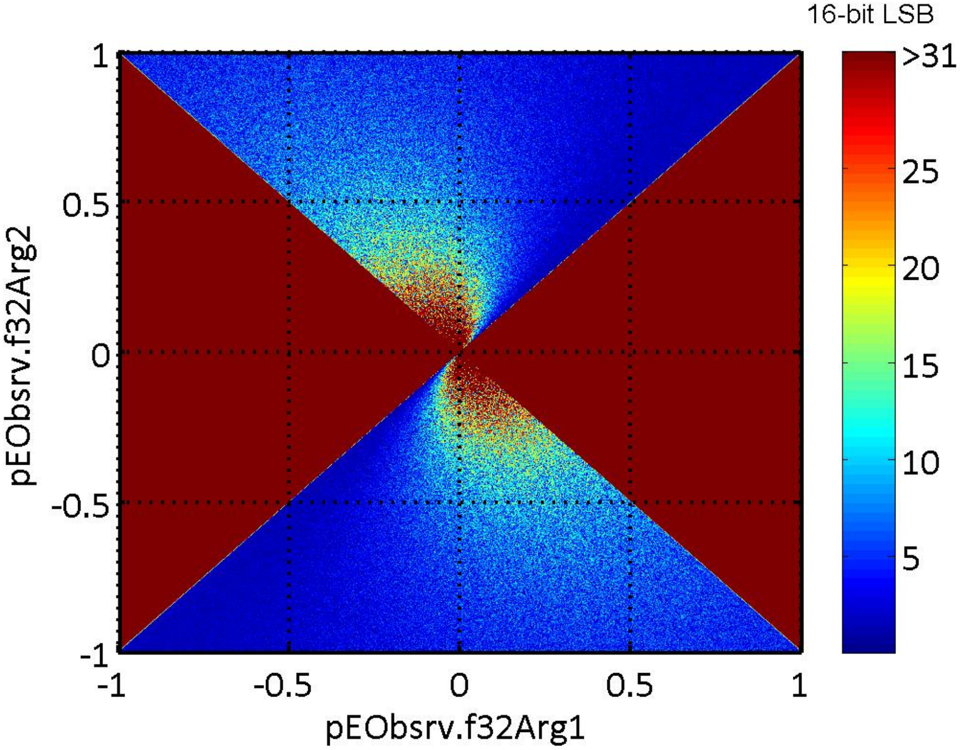

Figure 5-3. Absolute error of the returned value, assuming all shifts are 1

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.

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.

5.2.3.5 Code Example

```
#include "amclib.h"

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

void main (void)
{
    // Initialize BEMF observer state variables
    AMCLIB_BemfObsrvDQInit_F32 (&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;

    while(1);
}

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

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

    // Only one function call shall be placed in the periodical
    // function or interrupt
    f32PhaseErr = AMCLIB_BemfObsrvDQ_F32 (&mcIab, &mcUab, f32Velocity,
                                         f32Phase, &BemfObsrv);


    // Only one function call shall be placed in the periodical
    // function or interrupt
    f32PhaseErr = AMCLIB_BemfObsrvDQ (&mcIab, &mcUab, f32Velocity,
                                     f32Phase, &BemfObsrv,F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // Only one function call shall be placed in the periodical
    // function or interrupt
    f32PhaseErr = AMCLIB_BemfObsrvDQ (&mcIab, &mcUab, f32Velocity,
```

```
                                   f32Phase, &BemfObsrv);

            // Pass f32PhaseErr to the tracking observer
            // (...)
        }
```

5.2.4 Function AMCLIB_BemfObsrvDQ_F16

5.2.4.1 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);
```

5.2.4.2 Arguments

Table 5-6. 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 coefficients.

5.2.4.3 Return

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

5.2.4.4 Variant Specifics

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). The incorrect setting of the scaling constants may lead to an undesirable overflow or saturation during the computation. 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 5-7. Scaling constants

Scaling constant	Symbol	Calculation
Maximum stator phase voltage [V]	U_{MAX}	$U_{MAX}=U_{DC\_BUS}/\sqrt{3}$
Maximum phase current [A]	I_{MAX}	Maximum current depends on power stage capabilities.
Maximum required 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.

Parameters of the PIrAW controllers inside the BEMF observer can be calculated using the following equations:

$$pParamD.f16CC1sc = \left(K_P + \frac{K_I T_S}{2}\right) \cdot \frac{I_{MAX}}{E_{MAX}} \cdot 2^{-NShift}$$

$$pParamD.f16CC2sc = \left(-K_P + \frac{K_I T_S}{2}\right) \cdot \frac{I_{MAX}}{E_{MAX}} \cdot 2^{-NShift}$$

$$pParamD.u16NShift = NShift$$

$$pParamQ.f16CC1sc = \left(K_P + \frac{K_I T_S}{2}\right) \cdot \frac{I_{MAX}}{E_{MAX}} \cdot 2^{-NShift}$$

$$pParamQ.f16CC2sc = \left(-K_P + \frac{K_I T_S}{2}\right) \cdot \frac{I_{MAX}}{E_{MAX}} \cdot 2^{-NShift}$$

$$pParamQ.u16NShift = NShift$$

Equation **AMCLIB_BemfObsrvDQ_Eq3**

where T_S is the sampling interval, K_P is the proportional gain, and K_I is the integral gain. The upper and lower limits of the PIrAW controller should be set based on the expected dynamics of the system. *NShift* is the smallest nonnegative integer value that ensures that the controller coefficients fit in the fractional range [-1, 1). The gains 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_Eq4**

where ξ is the current loop attenuation, and ω_0 is the current loop natural frequency [rad/s]. Coefficients ξ and ω_0 should correspond to the values chosen for the FOC current loop.

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

$$f16IGain = \frac{2 \cdot L_d - T_S \cdot R_S}{2 \cdot L_d + T_S \cdot R_S}$$

$$f16UGain = \frac{T_S}{2 \cdot L_d + T_S \cdot R_S} \cdot \frac{U_{MAX}}{I_{MAX}} \cdot 2^{-NShiftRL}$$

$$f16WIGain = \frac{T_S \cdot L_q}{2 \cdot L_d + T_S \cdot R_S} \cdot \Omega_{MAX} \cdot 2^{-NShiftRL}$$

$$f16EGain = \frac{T_S}{2 \cdot L_d + T_S \cdot R_S} \cdot \frac{E_{MAX}}{I_{MAX}} \cdot 2^{-NShiftRL}$$

$$s16Shift = NShiftRL$$

Equation **AMCLIB_BemfObsrvDQ_Eq5**

NShiftRL is set to ensure that the gains fit in the fractional range [-1, 1).

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.e-4; % inductance in d-axis [H]
Lq = 3.e-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 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/Umax,...
                    (-Kp + Ki*Ts/2)*Imax/Umax]));
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/Umax*2^-NShift;
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/Umax*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 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.

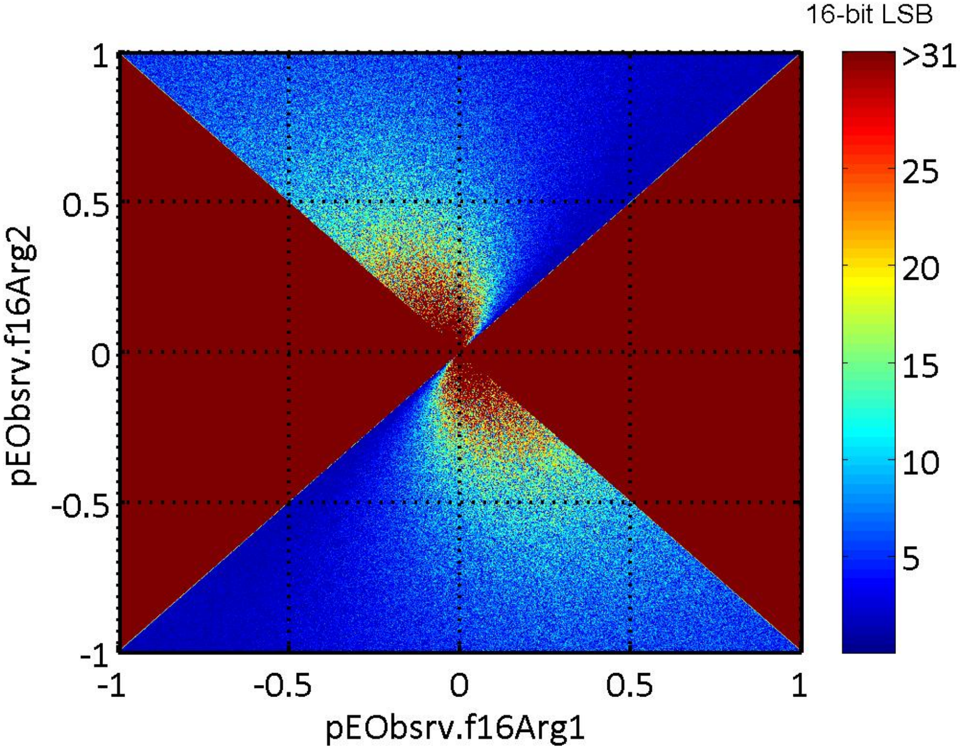

Figure 5-4. Absolute error of the returned value, assuming all shifts are 1

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.

Due to effectivity reasons, this function is implemented using intrinsic functions and inline assembly and is therefore not ANSI-C compliant. Note that some compilers do not support the enhanced features.

5.2.4.5 Code Example

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

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

void main (void)
{
     // Initialize BEMF observer state variables
    AMCLIB_BemfObsrvDQInit_F16 (&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;

    while(1);
}

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

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

    // Only one function call shall be placed in the periodical
    // function or interrupt
    f16PhaseErr = AMCLIB_BemfObsrvDQ_F16 (&mcIab, &mcUab, f16Velocity,
                                         f16Phase, &BemfObsrv);


    // Only one function call shall be placed in the periodical
    // function or interrupt
    f16PhaseErr = AMCLIB_BemfObsrvDQ (&mcIab, &mcUab, f16Velocity,
                                     f16Phase, &BemfObsrv,F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // Only one function call shall be placed in the periodical
    // function or interrupt
    f16PhaseErr = AMCLIB_BemfObsrvDQ (&mcIab, &mcUab, f16Velocity,
                                     f16Phase, &BemfObsrv);

    // Pass f16PhaseErr to the tracking observer
    // (...)
}
```

5.2.5 Function AMCLIB_BemfObsrvDQ_FLT

5.2.5.1 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);
```

5.2.5.2 Arguments

Table 5-8. 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].
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 coefficients.

5.2.5.3 Return

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

5.2.5.4 Variant Specifics

Parameters of the PIrAW controllers inside the BEMF observer can be calculated using the following equations:

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

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

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

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

Equation **AMCLIB_BemfObsrvDQ_Eq3**

where T_S is the sampling interval, K_P is the proportional gain, and K_I is the integral gain. The upper and lower limits of the PIrAW controller should be set based on the expected dynamics of the system. The gains 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_Eq4**

where ξ is the current loop attenuation, and ω_0 is the current loop natural frequency [rad/s]. Coefficients ξ and ω_0 should correspond to the values chosen for the FOC current loop.

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

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

Equation **AMCLIB_BemfObsrvDQ_Eq5**

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.e-4; % inductance in d-axis [H]
Lq = 3.e-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.

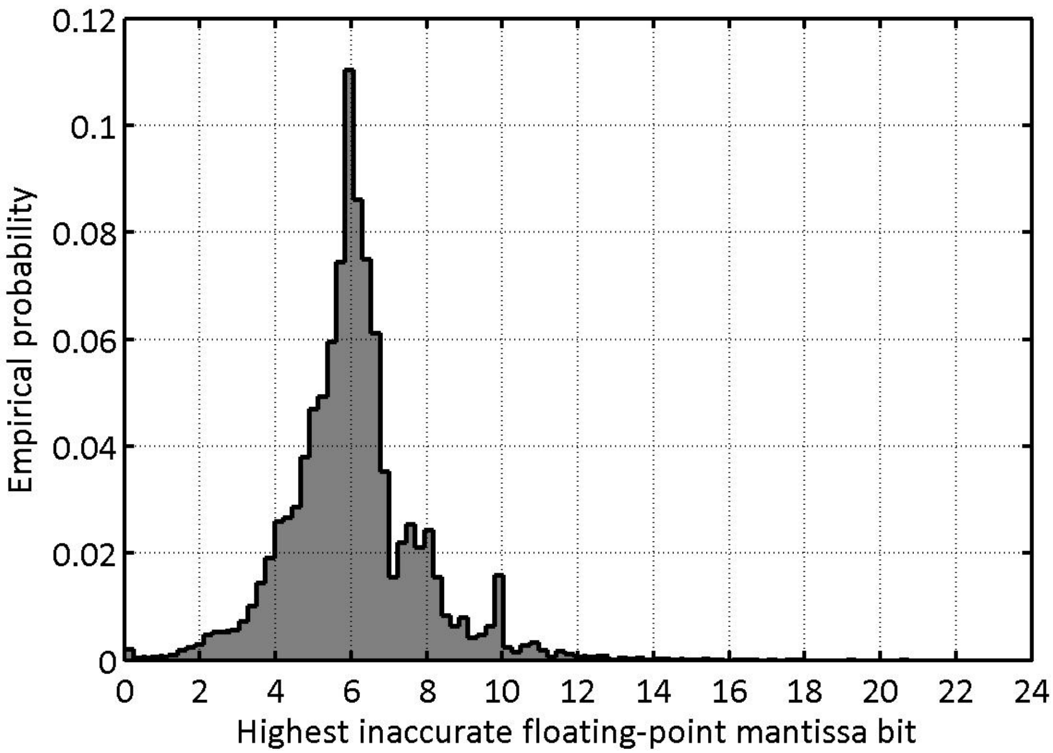

Figure 5-5. Histogram of the returned value error

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

5.2.5.5 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)
{
    // Initialize BEMF observer state variables
    AMCLIB_BemfObsrvDQInit_FLT (&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);

    while(1);
}

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

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

    // Only one function call shall be placed in the periodical
    // function or interrupt
    fltPhaseErr = AMCLIB_BemfObsrvDQ_FLT (&mcIab, &mcUab, fltVelocity,
                                           fltPhase, &BemfObsrv);


    // Only one function call shall be placed in the periodical
    // function or interrupt
    fltPhaseErr = AMCLIB_BemfObsrvDQ (&mcIab, &mcUab, fltVelocity,
                                       fltPhase, &BemfObsrv,FLT);
```

```
        // ############################################################
        // Available only if floating point implementation selected
        // as default
        // ############################################################

        // Only one function call shall be placed in the periodical
        // function or interrupt
        fltPhaseErr = AMCLIB_BemfObsrvDQ (&mcIab, &mcUab, fltVelocity,
                                          fltPhase, &BemfObsrv);

        // Pass fltPhaseErr to the tracking observer
        // (...)
    }
```

5.3 Function AMCLIB_TrackObsrvInit

This function initializes the AMCLIB_TrackObsrv function.

5.3.1 Description

This function clears the internal accumulator and internally-stored previous inputs of a tracking observer. It shall be also called after tracking observer parameter initialization whenever the tracking observer initialization is required.

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 an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.3.2 Re-entrancy:

The function is re-entrant.

5.3.3 Function AMCLIB_TrackObsrvInit_F32

5.3.3.1 Declaration

```
void AMCLIB_TrackObsrvInit_F32(AMCLIB_TRACK_OBSRV_T_F32 *pCtrl);
```

5.3.3.2 Arguments

Table 5-9. 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.

5.3.3.3 Code example

```
#define Wmax (2618F)
#define pi (3.1415927F)
#define Ts (1e-4F)
#define f0 (15F)
#define xi (0.707F)
#define w0 (2F*pi*f0)
#define Ki (w0*w0)
#define Kp (4F*pi*xi*f0)

#include "amclib.h"

AMCLIB_TRACK_OBSRV_T_F32 trMyTrObsrv;
tFrac32 f32PhaseErr;
tFrac32 f32PosEstim;
tFrac32 f32VelEstim;

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);

    // Clearing the tracking observer internal states
    AMCLIB_TrackObsrvInit_F32 (&trMyTrObsrv);

    // output should be f32VelEstim = FRAC32(4.011305e-2)
    // output should be f32PosEstim = FRAC32(1.671381e-3)
    // put this function into interrupt routine or call function periodically
    AMCLIB_TrackObsrv_F32 (f32PhaseErr, &f32PosEstim, &f32VelEstim,
&trMyTrObsrv);
```

```
                    // Clearing the tracking observer internal states
                    AMCLIB_TrackObsrvInit (&trMyTrObsrv, F32);

                    // output should be f32VelEstim = FRAC32(4.011305e-2)
                    // output should be f32PosEstim = FRAC32(1.671381e-3)
                    // put this function into interrupt routine or call function periodically
                    AMCLIB_TrackObsrv (f32PhaseErr, &f32PosEstim, &f32VelEstim, &trMyTrObsrv,
F32);

                    // ##############################################################
                    // Available only if 32-bit fractional implementation selected
                    // as default
                    // ##############################################################

                   // Clearing the tracking observer internal states
                    AMCLIB_TrackObsrvInit (&trMyTrObsrv);

                    // output should be f32VelEstim = FRAC32(4.011305e-2)
                    // output should be f32PosEstim = FRAC32(1.671381e-3)
                    // put this function into interrupt routine or call function periodically
                    AMCLIB_TrackObsrv (f32PhaseErr, &f32PosEstim, &f32VelEstim,
&trMyTrObsrv);
               }
```

5.3.4 Function AMCLIB_TrackObsrvInit_F16

5.3.4.1 Declaration

```
void AMCLIB_TrackObsrvInit_F16(AMCLIB_TRACK_OBSRV_T_F16 *pCtrl);
```

5.3.4.2 Arguments

Table 5-10. 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.

5.3.4.3 Code example

```
#define Wmax (2618F)
#define pi (3.1415927F)
#define Ts (1e-4F)
#define f0 (15F)
#define xi (0.707F)
#define w0 (2F*pi*f0)
#define Ki (w0*w0)
#define Kp (4F*pi*xi*f0)
```

```
#include "amclib.h"

AMCLIB_TRACK_OBSRV_T_F16 trMyTrObsrv;
tFrac16 f16PhaseErr;
tFrac16 f16PosEstim;
tFrac16 f16VelEstim;

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);

    // Clearing the tracking observer internal states
    AMCLIB_TrackObsrvInit_F16 (&trMyTrObsrv);

    // output should be f16VelEstim = FRAC16(4.011305e-2)
    // output should be f16PosEstim = FRAC16(1.671381e-3)
    // put this function into interrupt routine or call function periodically
    AMCLIB_TrackObsrv_F16 (f16PhaseErr, &f16PosEstim, &f16VelEstim,
&trMyTrObsrv);


    // Clearing the tracking observer internal states
    AMCLIB_TrackObsrvInit (&trMyTrObsrv, F16);

    // output should be f16VelEstim = FRAC16(4.011305e-2)
    // output should be f16PosEstim = FRAC16(1.671381e-3)
    // put this function into interrupt routine or call function periodically
    AMCLIB_TrackObsrv (f16PhaseErr, &f16PosEstim, &f16VelEstim, &trMyTrObsrv,
F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

   // Clearing the tracking observer internal states
    AMCLIB_TrackObsrvInit (&trMyTrObsrv);

    // output should be f16VelEstim = FRAC16(4.011305e-2)
    // output should be f16PosEstim = FRAC16(1.671381e-3)
    // put this function into interrupt routine or call function periodically
    AMCLIB_TrackObsrv (f16PhaseErr, &f16PosEstim, &f16VelEstim,
&trMyTrObsrv);
}
```

5.3.5 Function AMCLIB_TrackObsrvInit_FLT

5.3.5.1 Declaration

```
void AMCLIB_TrackObsrvInit_FLT(AMCLIB_TRACK_OBSRV_T_FLT *pCtrl);
```

5.3.5.2 Arguments

Table 5-11. 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 (floating-point overflow, underflow, inexact, invalid operation).

5.3.5.3 Code example:

```
#define pi (3.1415927F)
#define Ts (1e-4F)
#define f0 (15F)
#define xi (0.707F)
#define w0 (2F*pi*f0)
#define Ki (w0*w0)
#define Kp (4F*pi*xi*f0)

#include "amclib.h"

AMCLIB_TRACK_OBSRV_T_FLT trMyTrObsrv;
tFloat fltPhaseErr;
tFloat fltPosEstim;
tFloat fltVelEstim;

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);

    // Clearing the tracking observer internal states
    AMCLIB_TrackObsrvInit_FLT (&trMyTrObsrv);

    // output should be fltVelEstim = (tFloat)(3.342762e-3)
    // output should be fltPosEstim = (tFloat)(1.671381e-3)
    // put this function into interrupt routine or call function periodically
```

```
                    AMCLIB_TrackObsrv_FLT (fltPhaseErr, &fltPosEstim, &fltVelEstim,
&trMyTrObsrv);

                    // Clearing the tracking observer internal states
                    AMCLIB_TrackObsrvInit (&trMyTrObsrv, FLT);

                    // output should be fltVelEstim = (tFloat)(3.342762e-3)
                    // output should be fltPosEstim = (tFloat)(1.671381e-3)
                    // put this function into interrupt routine or call function periodically
                    AMCLIB_TrackObsrv (fltPhaseErr, &fltPosEstim, &fltVelEstim, &trMyTrObsrv,
FLT);

                    // ##############################################
                    // Available only if single precision floating point
                    // implementation selected as default
                    // ##############################################

                    // Clearing the tracking observer internal states
                    AMCLIB_TrackObsrvInit (&trMyTrObsrv);

                    // output should be fltVelEstim = (tFloat)(3.342762e-3)
                    // output should be fltPosEstim = (tFloat)(1.671381e-3)
                    // put this function into interrupt routine or call function periodically
                    AMCLIB_TrackObsrv (fltPhaseErr, &fltPosEstim, &fltVelEstim,
&trMyTrObsrv);
                }
```

5.4 Function AMCLIB_TrackObsrv

This function calculates the position tracking observer algorithm where the phase-locked loop mechanism is adopted. The input of the function is a phase error from the AMCLIB_BemfObsrvDQ function, representing the state filter providing estimates of the saliency based back-EMF in the estimated - reference frame.

5.4.1 Description

This function calculates the position tracking observer algorithm, where the phase-locked-loop mechanism is adopted. The input of the function is the phase error from the AMCLIB_BemfObsrvDQ function, representing the state filter providing the estimates of the saliency based back-EMF in the estimated γ - δ reference frame. Because of the differences between the actual and estimated parameters used in the observer model, the resulting back-EMF estimates can be divided, to extract the information about the displacement between the estimated and rotor flux reference frames, while reducing the effect of the observer parameter variation. The position displacement Θ_{err} is obtained by the following equation:

$$\Theta_{err} = \tan^{-1}\left(\frac{-e_{\gamma}}{e_{\delta}}\right)$$

Equation **AMCLIB_TrackObsrv_Eq0**

The estimated position can then be obtained by driving the position of the estimated reference frame γ - δ, to achieve zero displacement $\Theta_{err}=0$. Therefore a phase locked loop mechanism must be adopted, where the loop compensator ensures the correct tracking of the actual rotor flux position by keeping the error signal $\Theta_{err}=0$. The position tracking observer with a standard PI controller used as the loop compensator is depicted in the following picture:

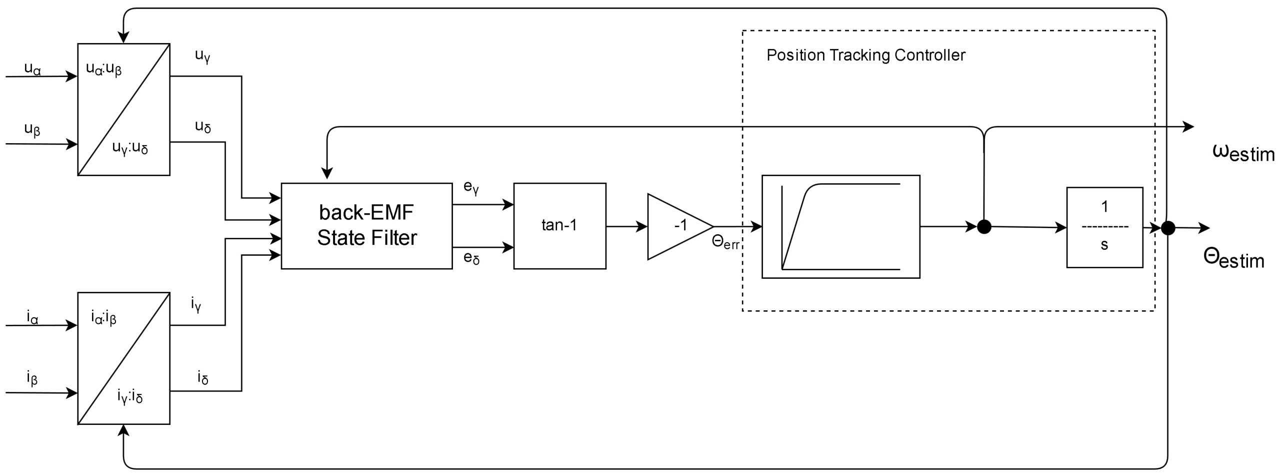

Figure 5-6. Block diagram of proposed PLL scheme for position estimation

The position tracking structure, described in Figure 5-6, can be linearized around the operating point $\Theta_{estim} = \Theta_e$. The linear approximation of the position tracking observer with standard PI controller is shown in the following picture:

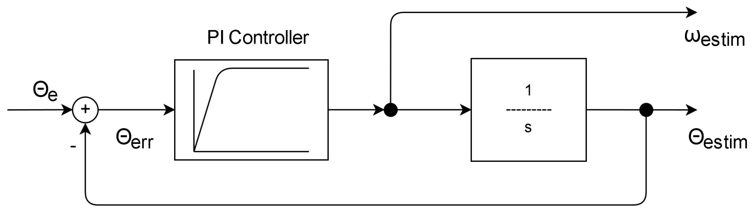

Figure 5-7. Linear approximation of the position tracking observer with standard PI controller

Linearized position tracking observer, depicted on Figure 5-7 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**

Considering the $s*K_p$ term in the nominator as negligible, the controller gains K_p and K_i are calculated by comparing the characteristic polynomial of the resulting transfer function to a standard second order system polynomial:

$$K_p = 4\pi\xi f_0$$
$$K_i = \left(2\pi f_0\right)^2$$

Equation **AMCLIB_TrackObsrv_Eq2**

where:

- ξ is the required attenuation
- f_0 is the required bandwidth of the position tracking 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 estimated and rotor flux frames displacement.

As demonstrated in Figure 5-6, the position tracking controller consists of the standard PI controller and integrator. The output of the ideal standard PI controller is defined by the following equation:

$$\omega_{\mathrm{estim}}\left(t\right) = \Theta_{\mathrm{err}}\left(t\right)K_p + K_i\int_0^{\infty}\Theta_{\mathrm{err}}\left(t\right)\mathrm{dt}$$

Equation **AMCLIB_TrackObsrv_Eq3**

and the output of the ideal integrator as follows:

$$\Theta_{\mathrm{estim}}\left(t\right) = \int_0^{\infty}\omega_{\mathrm{estim}}\left(t\right)\mathrm{dt}$$

Equation **AMCLIB_TrackObsrv_Eq4**

where:

- K_p is the proportional gain
- K_i is integral gain.

Using the Laplace transformation, equations AMCLIB_TrackObsrv_Eq3 and AMCLIB_TrackObsrv_Eq4 are transformed in to the continuous time domain:

$$\omega_{\text{estim}}(s) = K_p \Theta_{\text{err}}(s) + \tfrac{1}{s} K_i \Theta_{\text{err}}(s)$$
$$\Theta_{\text{estim}}(s) = \tfrac{1}{s} \omega_{\text{estim}}(s)$$

Equation **AMCLIB_TrackObsrv_Eq5**

forming two transfer functions:

$$G(s) = \frac{sK_p + K_i}{s}$$
$$H(s) = \frac{1}{s}$$

Equation **AMCLIB_TrackObsrv_Eq6**

where the G(s) is the transfer function of the standard PI controller in a continuous time domain, and H(s) is the transfer function of the integrator in a continuous time domain.

In order to implement the standard PI controller and integrator in the discrete digital control systems, both blocks needs to be transformed into a discrete time domain. Considering the trapezoid discretization method, the equations AMCLIB_TrackObsrv_Eq6 are transformed into the following equations:

$$\omega_{\text{estim}}(k) = \omega_{\text{estim}}(k-1) + \Theta_{\text{err}}(k)\left(K_p + \frac{K_i T_s}{2}\right) + \Theta_{\text{err}}(k-1)\left(-K_p + \frac{K_i T_s}{2}\right)$$
$$\Theta_{\text{estim}}(k) = \Theta_{\text{estim}}(k-1) + \omega_{\text{estim}}(k)\frac{T_s}{2} + \omega_{\text{estim}}(k-1)\frac{T_s}{2}$$

Equation **AMCLIB_TrackObsrv_Eq7**

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]
- $\omega_{estim}(k-1)$ is the estimated angular velocity in the previous calculation step [rad/s].

Using the substitution:

$$CC_1 = K_p + \frac{K_i T_S}{2}$$
$$CC_2 = -K_p + \frac{K_i T_S}{2}$$
$$C_1 = \frac{T_S}{2}$$

Equation **AMCLIB_TrackObsrv_Eq8**

the equation AMCLIB_TrackObsrv_Eq7 can be rewritten into:

$$\omega_{\mathrm{estim}}(k) = \omega_{\mathrm{estim}}(k-1) + \Theta_{\mathrm{err}}(k)\mathrm{CC}_1 + \Theta_{\mathrm{err}}(k-1)\mathrm{CC}_2$$
$$\Theta_{\mathrm{estim}}(k) = \Theta_{\mathrm{estim}}(k-1) + \omega_{\mathrm{estim}}(k)C_1 + \omega_{\mathrm{estim}}(k-1)C_1$$

Equation **AMCLIB_TrackObsrv_Eq9**

Note

The input/output pointers must contain valid addresses, otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.4.2 Re-entrancy

The function is re-entrant.

5.4.3 Function AMCLIB_TrackObsrv_F32

5.4.3.1 Declaration

```
void AMCLIB_TrackObsrv_F32(tFrac32 f32PhaseErr, tFrac32 *pPosEst, tFrac32 *pVelocityEst,
AMCLIB_TRACK_OBSRV_T_F32 *pCtrl);
```

5.4.3.2 Arguments

Table 5-12. 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.

5.4.3.3 Variant Specifics

In order to implement the discretized equations AMCLIB_TrackObsrv_Eq9 of the position tracking controller on the fixed point arithmetic platforms, the maximal values (scales) of the input and output signals are as follows:

- Θ^{MAX} - maximal value of the position tracking controller input phase error
- Ω^{MAX} - maximal value of the position tracking controller output angular velocity must be known. These maximal values are essential for the correct casting of the physical signal values into fixed point values [-1, 1).

Considering the same maximal values for the input phase error Θ_{err} and the output phase Θ_{estim}, fractional representations of the input and both output signals are obtained using these equations:

$$\Theta_f(k) = \frac{\Theta_{err}(k)}{\Theta^{MAX}}$$

$$\omega_f(k) = \frac{\omega_{err}(k)}{\omega^{MAX}}$$

Equation **AMCLIB_TrackObsrv_Eq10**

The resulting position tracking controller, discrete time domain equations in a fixed-point fractional representation, is given as follows:

$$\omega_f^{estim}(k) \bullet \omega^{MAX} = \omega_{estim}(k-1) \bullet \omega^{MAX} + \Theta_{err}(k) \bullet \Theta^{MAX} \bullet CC_1 + \Theta_{err}(k-1) \bullet \Theta^{MAX} \bullet CC_2$$

$$\Theta_f^{estim}(k) \bullet \Theta^{MAX} = \Theta_{estim}(k-1) \bullet \Theta^{MAX} + \omega_{estim}(k) \bullet \omega^{MAX} \bullet C_1 + \omega_{estim}(k-1) \bullet \omega^{MAX} \bullet C_1$$

Equation **AMCLIB_TrackObsrv_Eq11**

which can be rearranged into the following form:

$$\omega_f^{estim}(k) = \omega_{\text{estim}}(k-1) + \Theta_{err}(k)\frac{\Theta^{MAX}}{\omega^{MAX}}CC_1 + \Theta_{err}(k-1)\frac{\Theta^{MAX}}{\omega^{MAX}}CC_2$$
$$\Theta_f^{estim}(k) = \Theta_{\text{estim}}(k-1) + \omega_{\text{estim}}(k)\frac{\omega^{MAX}}{\Theta^{MAX}}C_1 + \omega_{\text{estim}}(k-1)\frac{\omega^{MAX}}{\Theta^{MAX}}C_1$$

Equation **AMCLIB_TrackObsrv_Eq12**

To further simplify the equation AMCLIB_TrackObsrv_Eq12, let's make the substitution:

$$CC_{1f} = CC_1 \cdot \frac{\Theta^{MAX}}{\omega^{MAX}} = \left(K_p + \frac{K_i T_S}{2}\right) \cdot \frac{\Theta^{MAX}}{\omega^{MAX}}$$
$$CC_{2f} = CC_2 \cdot \frac{\Theta^{MAX}}{\omega^{MAX}} = \left(-K_p + \frac{K_i T_S}{2}\right) \cdot \frac{\Theta^{MAX}}{\omega^{MAX}}$$
$$C_{1f} = C_1 \cdot \frac{\omega^{MAX}}{\Theta^{MAX}} = \frac{T_S}{2} \cdot \frac{\omega^{MAX}}{\Theta^{MAX}}$$

Equation **AMCLIB_TrackObsrv_Eq13**

where:

- CC_{1f} and CC_{2f} are the PI controller coefficients adapted according to the input and output scale values
- C_{1f} is the integrator coefficient adapted according to the input and output scale values.

To implement these coefficients as fractional numbers, all three coefficients have to fit in the fractional range [-1,1). However, depending on the CC_{1f}, CC_{2f} and C_{1f} and on Θ^{MAX}, Ω^{MAX} maximum values, calculation of CC_{1f}, CC_{2f} and C_{1f} may result in values outside the fractional [-1, 1) range. Therefore, a scaling of CC_{1f}, CC_{2f} and C_{1f} has to be introduced:

$$f32CC1_{SC} = CC_{1f} \cdot 2^{-u16NShift} = \left(K_p + \frac{K_i T_S}{2}\right) \cdot \frac{\Theta^{MAX}}{\omega^{MAX}} \cdot 2^{-u16NShift}$$
$$f32CC2_{SC} = CC_{2f} \cdot 2^{-u16NShift} = \left(-K_p + \frac{K_i T_S}{2}\right) \cdot \frac{\Theta^{MAX}}{\omega^{MAX}} \cdot 2^{-u16NShift}$$
$$f32C1_{SC} = C_{1f} \cdot 2^{-u16NIntegSh} = \frac{T_S}{2} \cdot \frac{\omega^{MAX}}{\Theta^{MAX}} \cdot 2^{-u16NIntegSh}$$

Equation **AMCLIB_TrackObsrv_Eq14**

Both scaling shifts, u16NShift and u16NIntegSh, are chosen such that all three CC $_{1f}$, CC $_{2f}$ and C $_{1f}$ coefficients reside in the range [-1, 1). To simplify the implementation on the digital controllers, these scaling shifts are chosen to be of a power of 2; thus the final scaling is a simple shift operation on digital controllers. Moreover, the scaling shifts cannot be a negative number, so the scaling operation is always scales the numbers with an absolute value larger than 1 down to fit the range [-1,1). With these discussed requirements, the scaling shifts are calculated as follows:

$$u16NShift = \max\left(ceil\left(\frac{\log(abs(CC_{1f}))}{\log(2)}\right),\ ceil\left(\frac{\log(abs(CC_{2f}))}{\log(2)}\right)\right)$$
$$u16NIntegSh = ceil\left(\frac{\log(abs(C_{1f}))}{\log(2)}\right)$$

Equation **AMCLIB_TrackObsrv_Eq15**

where:

- u16NShift is the scaling shift for the standard PI controller
- u16NIntegSh is the scaling shift for the integrator.

Using AMCLIB_TrackObsrv_Eq13 and AMCLIB_TrackObsrv_Eq14, the AMCLIB_TrackObsrv_Eq12 equations are transformed into a final, scaled, fractional equations of the position tracking controller, represented by the AMCLIB_TrackObsrv_F32 function:

$$\omega_{\mathrm{estim}}(k) = \left(\omega_{\mathrm{estim}}(k-1) + \Theta_{err}(k)\cdot f32CC1_{SC} + \Theta_{err}(k-1)\cdot f32CC2_{SC}\right)\cdot 2^{u16NShift}$$
$$\Theta_{\mathrm{estim}}(k) = \left(\Theta_{\mathrm{estim}}(k-1) + \omega_{\mathrm{estim}}(k)\cdot f32C1_{SC} + \omega_{\mathrm{estim}}(k-1)\cdot f32C1_{SC}\right)\cdot 2^{u16NIntegSh}$$

Equation **AMCLIB_TrackObsrv_Eq16**

where:

- $\omega_{estim}(k)$ is the output estimated angular velocity in the current step
- $\omega_{estim}(k-1)$ is the output estimated angular velocity in the previous calculation step
- $\Theta_{estim}(k)$ is the output estimated position in the current step
- $\Theta_{estim}(k-1)$ is the output estimated position in the previous calculation step
- f32CC1sc is the 1st coefficient of the PI controller
- f32CC2sc is the 2nd coefficient of the PI controller

- u16NShift is the scaling shift of the PI controller
- f32C1sc is the integrator constant
- u16NIntegSh is the scaling shift of the integrator

The output estimated angular velocity signal limitation is implemented in the PI controller. The actual output $\omega_{estim}(k)$ is bounded so as to not exceed the given UpperLimit and LowerLimit values:

$$\omega_{estim}(k) = \begin{cases} f32UpperLimit & => \quad \omega_{estim}(k) \geq f32UpperLimit \\ \omega_{estim}(k) & => \quad f32LowerLimit < \omega_{estim}(k) < f32UpperLimit \\ f32LowerLimit & => \quad \omega_{estim}(k) \leq f32\mathrm{LowerLimit} \end{cases}$$

Equation **AMCLIB_TrackObsrv_Eq17**

Where the bounds are exceeded, the non-linear saturation characteristics will take effect and influence the dynamic behavior. The output limitation is implemented on the output sum; therefore if the limitation occurs, the controller output is clipped to its bounds.

The accuracy of the results is guaranteed for outputs f32PosEstim and f32VelEstim only in cases when (pCtrl.pParamPI.u16NShift + pCtrlpParamInteg.u16NShift) < 15.

Note

If the output of the internal integrator exceeds the fractional range [-1,1) for the Θ_{estim} output, an overflow occurs. This behavior allows the continual integration of the angular velocity of a rotor to obtain the actual rotor position, assuming the output range corresponds to one complete revolution.

5.4.3.4 Code example

```
#define Wmax (2618F)
#define pi (3.1415927F)
#define Ts (1e-4F)
#define f0 (15F)
#define xi (0.707F)
#define w0 (2F*pi*f0)
#define Ki (w0*w0)
#define Kp (4F*pi*xi*f0)

#include "amclib.h"

AMCLIB_TRACK_OBSRV_T_F32 trMyTrObsrv;
tFrac32 f32PhaseErr;
tFrac32 f32PosEstim;
tFrac32 f32VelEstim;
```

```
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);

    // Clearing the tracking observer internal states
    AMCLIB_TrackObsrvInit_F32 (&trMyTrObsrv);

    // output should be f32VelEstim = FRAC32(4.011305e-2)
    // output should be f32PosEstim = FRAC32(1.671381e-3)
    // put this function into interrupt routine or call function periodically
    AMCLIB_TrackObsrv_F32 (f32PhaseErr, &f32PosEstim, &f32VelEstim,
&trMyTrObsrv);


    // Clearing the tracking observer internal states
    AMCLIB_TrackObsrvInit (&trMyTrObsrv, F32);

    // output should be f32VelEstim = FRAC32(4.011305e-2)
    // output should be f32PosEstim = FRAC32(1.671381e-3)
    // put this function into interrupt routine or call function periodically
    AMCLIB_TrackObsrv (f32PhaseErr, &f32PosEstim, &f32VelEstim, &trMyTrObsrv,
F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

   // Clearing the tracking observer internal states
    AMCLIB_TrackObsrvInit (&trMyTrObsrv);

    // output should be f32VelEstim = FRAC32(4.011305e-2)
    // output should be f32PosEstim = FRAC32(1.671381e-3)
    // put this function into interrupt routine or call function periodically
    AMCLIB_TrackObsrv (f32PhaseErr, &f32PosEstim, &f32VelEstim,
&trMyTrObsrv);
}
```

5.4.4 Function AMCLIB_TrackObsrv_F16

5.4.4.1 Declaration

```
void AMCLIB_TrackObsrv_F16(tFrac16 f16PhaseErr, tFrac16 *pPosEst, tFrac16 *pVelocityEst,
AMCLIB_TRACK_OBSRV_T_F16 *pCtrl);
```

5.4.4.2 Arguments

Table 5-13. 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.

5.4.4.3 Variant Specifics

To implement the discretized equations AMCLIB_TrackObsrv_Eq9 of the position tracking controller on the fixed-point arithmetic platforms, the maximal values (scales) of the input and output signals:

- Θ^{MAX} - the maximal value of the position tracking controller input phase error
- Ω^{MAX} - the maximal value of the position tracking controller output angular velocity have to be known. These maximal values are essential for the correct casting of the physical signal values into fixed point values [-1, 1).

Considering the same maximal values for the input phase error Θ_{err} and the output phase Θ_{estim}, the fractional representation input and both output signals are obtained using these equations:

$$\Theta_f(k) = \frac{\Theta_{err}(k)}{\Theta^{MAX}}$$

$$\omega_f(k) = \frac{\omega_{err}(k)}{\omega^{MAX}}$$

Equation **AMCLIB_TrackObsrv_Eq10**

The resulting position tracking controller, discrete time domain equations in fixed-point fractional representation, are given as follows:

$$\omega_f^{estim}(k) \bullet \omega^{MAX} = \omega_{estim}(k-1) \bullet \omega^{MAX} + \Theta_{err}(k) \bullet \Theta^{MAX} \bullet CC_1 + \Theta_{err}(k-1) \bullet \Theta^{MAX} \bullet CC_2$$

$$\Theta_f^{estim}(k) \bullet \Theta^{MAX} = \Theta_{estim}(k-1) \bullet \Theta^{MAX} + \omega_{estim}(k) \bullet \omega^{MAX} \bullet C_1 + \omega_{estim}(k-1) \bullet \omega^{MAX} \bullet C_1$$

Equation **AMCLIB_TrackObsrv_Eq11**

which can be rearranged into this form:

$$\omega_f^{estim}(k) = \omega_{\text{estim}}(k-1) + \Theta_{err}(k)\frac{\Theta^{MAX}}{\omega^{MAX}}CC_1 + \Theta_{err}(k-1)\frac{\Theta^{MAX}}{\omega^{MAX}}CC_2$$
$$\Theta_f^{estim}(k) = \Theta_{\text{estim}}(k-1) + \omega_{\text{estim}}(k)\frac{\omega^{MAX}}{\Theta^{MAX}}C_1 + \omega_{\text{estim}}(k-1)\frac{\omega^{MAX}}{\Theta^{MAX}}C_1$$

Equation **AMCLIB_TrackObsrv_Eq12**

To further simplify the equation AMCLIB_TrackObsrv_Eq12, let's make the substitution:

$$CC_{1f} = CC_1 \cdot \frac{\Theta^{MAX}}{\omega^{MAX}} = \left(K_p + \frac{K_i T_S}{2}\right) \cdot \frac{\Theta^{MAX}}{\omega^{MAX}}$$
$$CC_{2f} = CC_2 \cdot \frac{\Theta^{MAX}}{\omega^{MAX}} = \left(-K_p + \frac{K_i T_S}{2}\right) \cdot \frac{\Theta^{MAX}}{\omega^{MAX}}$$
$$C_{1f} = C_1 \cdot \frac{\omega^{MAX}}{\Theta^{MAX}} = \frac{T_S}{2} \cdot \frac{\omega^{MAX}}{\Theta^{MAX}}$$

Equation **AMCLIB_TrackObsrv_Eq13**

where:

- CC $_{1f}$ and CC $_{2f}$ are the PI controller coefficients adapted according to the input and output scale values
- C $_{1f}$ is the integrator coefficient adapted according to the input and output scale values.

To implement these coefficients as fractional numbers, all three coefficients must fit in the fractional range [-1,1). However, depending on CC $_{1f}$, CC $_{2f}$ and C $_{1f}$ as well as on Θ^{MAX} and Ω^{MAX} maximum values, calculations of CC $_{1f}$, CC $_{2f}$ and C $_{1f}$ may result in values outside the fractional [-1, 1) range. Therefore, a scaling of CC $_{1f}$, CC $_{2f}$ and C $_{1f}$ must be introduced:

$$f16CC1_{SC} = CC_{1f} \cdot 2^{-u16NShift} = \left(K_p + \frac{K_i T_S}{2}\right) \cdot \frac{\Theta^{MAX}}{\omega^{MAX}} \cdot 2^{-u16NShift}$$
$$f16CC2_{SC} = CC_{2f} \cdot 2^{-u16NShift} = \left(-K_p + \frac{K_i T_S}{2}\right) \cdot \frac{\Theta^{MAX}}{\omega^{MAX}} \cdot 2^{-u16NShift}$$
$$f16C1_{SC} = C_{1f} \cdot 2^{-u16NIntegSh} = \frac{T_S}{2} \cdot \frac{\omega^{MAX}}{\Theta^{MAX}} \cdot 2^{-u16NIntegSh}$$

Equation **AMCLIB_TrackObsrv_Eq14**

Both scaling shifts, u16NShift and u16NIntegSh, are chosen such that all three CC $_{1f}$, CC $_{2f}$ and C $_{1f}$ coefficients fit in the range [-1, 1). To simplify the implementation on the digital controllers, these scaling shifts are chosen to be power of the 2; thus the final scaling is a simple shift operation on the digital controllers. Moreover, the scaling shifts cannot be a negative number, so the scaling operation always scales the numbers with an absolute value larger than 1 down to fit the range [-1,1). With these discussed requirements, the scaling shifts are calculated as follows:

$$u16NShift = \max\left(ceil\left(\frac{\log(abs(CC_{1f}))}{\log(2)} \right),\ ceil\left(\frac{\log(abs(CC_{2f}))}{\log(2)} \right) \right)$$
$$u16NIntegSh = ceil\left(\frac{\log(abs(C_{1f}))}{\log(2)} \right)$$

Equation **AMCLIB_TrackObsrv_Eq15**

where:

- u16NShift is the scaling shift for the standard PI controller
- u16NIntegSh is the scaling shift for the integrator.

Using AMCLIB_TrackObsrv_Eq13 and AMCLIB_TrackObsrv_Eq14, the AMCLIB_TrackObsrv_Eq12 equations are transformed into a final, scaled, fractional equations of the position tracking controller, represented by the AMCLIB_TrackObsrv_F16 function:

$$\omega_{\mathrm{estim}}(k) = \left(\omega_{\mathrm{estim}}(k-1) + \Theta_{err}(k) \cdot f16CC1_{SC} + \Theta_{err}(k-1) \cdot f16CC2_{SC} \right) \cdot 2^{u16NShift}$$
$$\Theta_{\mathrm{estim}}(k) = \left(\Theta_{\mathrm{estim}}(k-1) + \omega_{\mathrm{estim}}(k) \cdot f16C1_{SC} + \omega_{\mathrm{estim}}(k-1) \cdot f16C1_{SC} \right) \cdot 2^{u16NIntegSh}$$

Equation **AMCLIB_TrackObsrv_Eq16**

where:

- ω_{estim}(k) is the output estimated angular velocity in the current step
- ω_{estim}(k-1) is the output estimated angular velocity in the previous calculation step
- Θ_{estim}(k) is the output estimated position in the current step
- Θ_{estim}(k-1) is the output estimated position in the previous calculation step
- f16CC1sc is the 1st coefficient of the PI controller
- f16CC2sc is the 2nd coefficient of the PI controller

- u16NShift is the scaling shift of the PI controller
- f16C1sc is the integrator constant
- u16NIntegSh is the scaling shift of the integrator

The output estimated angular velocity signal limitation is implemented in the PI controller. The actual output $\omega_{estim}(k)$ is bounded so as to not exceed the given UpperLimit and LowerLimit limit values:

$$\omega_{estim}(k) = \begin{cases} f16UpperLimit & => \quad \omega_{estim}(k) \geq f16UpperLimit \\ \omega_{estim}(k) & => \quad f16LowerLimit < \omega_{estim}(k) < f16UpperLimit \\ f16LowerLimit & => \quad \omega_{estim}(k) \leq f16\mathrm{LowerLimit} \end{cases}$$

Equation **AMCLIB_TrackObsrv_Eq17**

Where the bounds are exceeded, the non-linear saturation characteristics will take effect and influence the dynamic behavior. The output limitation is implemented on the output sum; therefore if the limitation occurs, the controller output is clipped to its bounds.

The accuracy of f16VelEstim is guaranteed only for cases when pCtrl.pParamPI.u16NShift <= 15. The accuracy of f16PosEstim is guaranteed only for cases when (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 might rise above the guaranteed limits.

Note

If the output of the internal integrator exceeds the fractional range [-1,1) for the Θ_{estim} output, an overflow occurs. This behavior allows the continual integration of an angular velocity of a rotor to obtain the actual rotor position, assuming the output range corresponds to one complete revolution.

5.4.4.4 Code example

```
#define Wmax (2618F)
#define pi (3.1415927F)
#define Ts (1e-4F)
#define f0 (15F)
#define xi (0.707F)
#define w0 (2F*pi*f0)
#define Ki (w0*w0)
#define Kp (4F*pi*xi*f0)

#include "amclib.h"
```

```
AMCLIB_TRACK_OBSRV_T_F16 trMyTrObsrv;
tFrac16 f16PhaseErr;
tFrac16 f16PosEstim;
tFrac16 f16VelEstim;

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);

    // Clearing the tracking observer internal states
    AMCLIB_TrackObsrvInit_F16 (&trMyTrObsrv);

    // output should be f16VelEstim = FRAC16(4.011305e-2)
    // output should be f16PosEstim = FRAC16(1.671381e-3)
    // put this function into interrupt routine or call function periodically
    AMCLIB_TrackObsrv_F16 (f16PhaseErr, &f16PosEstim, &f16VelEstim,
&trMyTrObsrv);


    // Clearing the tracking observer internal states
    AMCLIB_TrackObsrvInit (&trMyTrObsrv, F16);

    // output should be f16VelEstim = FRAC16(4.011305e-2)
    // output should be f16PosEstim = FRAC16(1.671381e-3)
    // put this function into interrupt routine or call function periodically
    AMCLIB_TrackObsrv (f16PhaseErr, &f16PosEstim, &f16VelEstim, &trMyTrObsrv,
F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

   // Clearing the tracking observer internal states
    AMCLIB_TrackObsrvInit (&trMyTrObsrv);

    // output should be f16VelEstim = FRAC16(4.011305e-2)
    // output should be f16PosEstim = FRAC16(1.671381e-3)
    // put this function into interrupt routine or call function periodically
    AMCLIB_TrackObsrv (f16PhaseErr, &f16PosEstim, &f16VelEstim,
&trMyTrObsrv);
}
```

5.4.5 Function AMCLIB_TrackObsrv_FLT

5.4.5.1 Declaration

```
void AMCLIB_TrackObsrv_FLT(tFloat fltPhaseErr, tFloat *pPosEst, tFloat *pVelocityEst,
AMCLIB_TRACK_OBSRV_T_FLT *pCtrl);
```

5.4.5.2 Arguments

Table 5-14. 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.

5.4.5.3 Variant Specifics

The output estimated angular velocity signal limitation is implemented in the PI controller. The actual output $\omega_{estim}(k)$ is bounded so as to not exceed the given UpperLimit and LowerLimit limit values:

$$\omega_{estim}(k) = \begin{cases} \mathrm{flt}UpperLimit & => \quad \omega_{estim}(k) \geq \mathrm{flt}UpperLimit \\ \omega_{estim}(k) & => \quad \mathrm{flt}LowerLimit < \omega_{estim}(k) < \mathrm{flt}UpperLimit \\ \mathrm{flt}LowerLimit & => \quad \omega_{estim}(k) \leq \mathrm{fltLowerLimit} \end{cases}$$

Equation **AMCLIB_TrackObsrv_Eq17**

Where the bounds are exceeded, the non-linear saturation characteristics will take effect and influence the dynamic behavior. The output limitation is implemented on the output sum; therefore, if the limitation occurs, the controller output is clipped to its bounds.

Note

If the output of the internal integrator exceeds the fractional range $[-\pi, \pi)$ for the Θ_{estim} output, an output wraparound occurs. This behavior allows the continual integration of the angular velocity of a rotor to obtain the actual rotor position, assuming the output range corresponds to one complete revolution.

Note

The function may raise the floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

5.4.5.4 Code example:

```
#define pi (3.1415927F)
#define Ts (1e-4F)
#define f0 (15F)
#define xi (0.707F)
#define w0 (2F*pi*f0)
#define Ki (w0*w0)
#define Kp (4F*pi*xi*f0)

#include "amclib.h"

AMCLIB_TRACK_OBSRV_T_FLT trMyTrObsrv;
tFloat fltPhaseErr;
tFloat fltPosEstim;
tFloat fltVelEstim;

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);

    // Clearing the tracking observer internal states
    AMCLIB_TrackObsrvInit_FLT (&trMyTrObsrv);

    // output should be fltVelEstim = (tFloat)(3.342762e-3)
    // output should be fltPosEstim = (tFloat)(1.671381e-3)
    // put this function into interrupt routine or call function periodically
    AMCLIB_TrackObsrv_FLT (fltPhaseErr, &fltPosEstim, &fltVelEstim,
&trMyTrObsrv);


    // Clearing the tracking observer internal states
    AMCLIB_TrackObsrvInit (&trMyTrObsrv, FLT);

    // output should be fltVelEstim = (tFloat)(3.342762e-3)
    // output should be fltPosEstim = (tFloat)(1.671381e-3)
    // put this function into interrupt routine or call function periodically
    AMCLIB_TrackObsrv (fltPhaseErr, &fltPosEstim, &fltVelEstim, &trMyTrObsrv,
FLT);

    // ##################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##################################################
```

```
                    // Clearing the tracking observer internal states
                    AMCLIB_TrackObsrvInit (&trMyTrObsrv);

                    // output should be fltVelEstim = (tFloat)(3.342762e-3)
                    // output should be fltPosEstim = (tFloat)(1.671381e-3)
                    // put this function into interrupt routine or call function periodically
                    AMCLIB_TrackObsrv (fltPhaseErr, &fltPosEstim, &fltVelEstim,
&trMyTrObsrv);
                }
```

5.5 Function GDFLIB_FilterFIRInit

This function initializes the FIR filter buffers.

5.5.1 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 an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

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 behavior of the function might be expected including the incorrect memory access exception.

5.5.2 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.

5.5.3 Function GDFLIB_FilterFIRInit_F32

5.5.3.1 Declaration

```
void GDFLIB_FilterFIRInit_F32(const GDFLIB_FILTERFIR_PARAM_T_F32 *const pParam,
GDFLIB_FILTERFIR_STATE_T_F32 *const pState, tFrac32 *pInBuf);
```

5.5.3.2 Arguments

Table 5-15. 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.

5.5.3.3 Variant Specifics

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.

5.5.3.4 Code Example

```
#include "gdflib.h"

#define FIR_NUMTAPS 16
#define FIR_NUMTAPS_MAX 64
#define FIR_ORDER (FIR_NUMTAPS - 1)

GDFLIB_FILTERFIR_PARAM_T_F32 Param;
GDFLIB_FILTERFIR_STATE_T_F32 State0, State1, State2;

tFrac32 f32InBuf[FIR_NUMTAPS_MAX];
tFrac32 f32CoefBuf[FIR_NUMTAPS_MAX];

#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 = 15;
    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);

                      // ##############################################################
                      // 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}
                }
```

5.5.4 Function GDFLIB_FilterFIRInit_F16

5.5.4.1 Declaration

```
void GDFLIB_FilterFIRInit_F16(const GDFLIB_FILTERFIR_PARAM_T_F16 *const pParam,
GDFLIB_FILTERFIR_STATE_T_F16 *const pState, tFrac16 *pInBuf);
```

5.5.4.2 Arguments

Table 5-16. 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.

5.5.4.3 Code Example

```
#include "gdflib.h"

#define FIR_NUMTAPS 16
#define FIR_NUMTAPS_MAX 64
#define FIR_ORDER (FIR_NUMTAPS - 1)

GDFLIB_FILTERFIR_PARAM_T_F16 Param;
GDFLIB_FILTERFIR_STATE_T_F16 State0, State1, State2;

tFrac16 f16InBuf[FIR_NUMTAPS_MAX];
tFrac16 f16CoefBuf[FIR_NUMTAPS_MAX];

#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 = 15;
    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
                    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}
                }
```

5.5.5 Function GDFLIB_FilterFIRInit_FLT

5.5.5.1 Declaration

```
void GDFLIB_FilterFIRInit_FLT(const GDFLIB_FILTERFIR_PARAM_T_FLT *const pParam,
GDFLIB_FILTERFIR_STATE_T_FLT *const pState, tFloat *pInBuf);
```

5.5.5.2 Arguments

Table 5-17. 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.

5.5.5.3 Code Example

```
#include "gdflib.h"

#define FIR_NUMTAPS 16
#define FIR_NUMTAPS_MAX 64
#define FIR_ORDER (FIR_NUMTAPS - 1)

GDFLIB_FILTERFIR_PARAM_T_FLT Param;
GDFLIB_FILTERFIR_STATE_T_FLT State0, State1, State2;

tFloat fltInBuf[FIR_NUMTAPS_MAX];
tFloat fltCoefBuf[FIR_NUMTAPS_MAX];

#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 = 15;
    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
                    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}
                }
```

5.6 Function GDFLIB_FilterFIR

The function performs a single iteration of an FIR filter.

5.6.1 Description

The function performs the operation of an FIR filter on a sample-by-sample basis. At each new input to the FIR filter, the function should be called, which will return a new filtered value.

The FIR filter is defined by the following formula:

$$y[n] = h_0 x[n] + h_1 x[n-1] + . \; . \; . + h_N x[n-N]$$

Equation **GDFLIB_FilterFIR_Eq1**

where: x[n] is the input signal, y[n] is the output signal, h_i are the filter coefficients, and N is the filter order. It should be noted, that the number of taps of the filter is N + 1 in this case.

The first call to the function must be preceded by an initialization, which can be made through the GDFLIB_FilterFIRInit function. The GDFLIB_FilterFIRInit and then the GDFLIB_FilterFIR functions should be called with the same parameters.

The filter coefficients are stored in the parameter structure, in the structure member pParam->pCoefBuf.

Note

The input/output pointers must contain valid addresses, otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

From the performance point of view, the function is designed to work with filters with a larger number of taps (equal order + 1). As a rule of thumb, if the number of taps is lower than 5, a different algorithm should be considered.

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 behavior of the function might be expected including the incorrect memory access exception.

5.6.2 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.

5.6.3 Function GDFLIB_FilterFIR_F32

5.6.3.1 Declaration

```
tFrac32 GDFLIB_FilterFIR_F32(tFrac32 f32In, const GDFLIB_FILTERFIR_PARAM_T_F32 *const pParam,
GDFLIB_FILTERFIR_STATE_T_F32 *const pState);
```

5.6.3.2 Arguments

Table 5-18. GDFLIB_FilterFIR_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input value.

Table continues on the next page...

Table 5-18. GDFLIB_FilterFIR_F32 arguments (continued)

Type	Name	Direction	Description
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.

5.6.3.3 Return

The value of a filtered signal after processing by an FIR filter.

5.6.3.4 Variant Specifics

The multiply and accumulate operations are performed with 64 accumulation, which means that no saturation is performed during computations. However, if the final value cannot fit in the return data type, saturation may occur. It should be noted, although rather theoretically, that no saturation is performed on the accumulation guard bits and an overflow over the accumulation guard bits may occur.

The function assumes that the filter order is at least one, which is equivalent to two taps. The filter also cannot contain more than 0xffffffff(hexadecimal) taps, which is equivalent to the order of 0xfffffffe (hexadecimal).

The input values are recorded by the function in the provided state structure in a circular buffer, pointed to by the state structure member pState->pInBuf. The buffer index is stored in pState->u32Idx, which points to the buffer element where a new input signal sample will be stored.

5.6.3.5 Code Example

```
#include "gdflib.h"

#define FIR_NUMTAPS 16
#define FIR_NUMTAPS_MAX 64
#define FIR_ORDER (FIR_NUMTAPS - 1)

GDFLIB_FILTERFIR_PARAM_T_F32 Param;
GDFLIB_FILTERFIR_STATE_T_F32 State0, State1, State2;

tFrac32 f32InBuf[FIR_NUMTAPS_MAX];
tFrac32 f32CoefBuf[FIR_NUMTAPS_MAX];
```

```
#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 = 15;
    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);

      // ###############################################################
      // 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}
              }
```

5.6.4 Function GDFLIB_FilterFIR_F16

5.6.4.1 Declaration

```
tFrac16 GDFLIB_FilterFIR_F16(tFrac16 f16In, const GDFLIB_FILTERFIR_PARAM_T_F16 *const pParam,
GDFLIB_FILTERFIR_STATE_T_F16 *const pState);
```

5.6.4.2 Arguments

Table 5-19. GDFLIB_FilterFIR_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input value.
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.

5.6.4.3 Return

The value of a filtered signal after processing by an FIR filter.

5.6.4.4 Variant Specifics

The multiply and accumulate operations are performed with 32 accumulation, which means that no saturation is performed during computations. However, if the final value cannot fit in the return data type, saturation may occur. It should be noted, although rather theoretically, that no saturation is performed on the accumulation guard bits and an overflow over the accumulation guard bits may occur.

The function assumes that the filter order is at least one, which is equivalent to two taps. The filter also cannot contain more than 0xffff(hexadecimal) taps, which is equivalent to the order of 0xfffe (hexadecimal).

The input values are recorded by the function in the provided state structure in a circular buffer, pointed to by the state structure member pState->pInBuf. The buffer index is stored in pState->u16Idx, which points to the buffer element where a new input signal sample will be stored.

5.6.4.5 Code Example

```
#include "gdflib.h"

#define FIR_NUMTAPS 16
#define FIR_NUMTAPS_MAX 64
#define FIR_ORDER (FIR_NUMTAPS - 1)

GDFLIB_FILTERFIR_PARAM_T_F16 Param;
GDFLIB_FILTERFIR_STATE_T_F16 State0, State1, State2;

tFrac16 f16InBuf[FIR_NUMTAPS_MAX];
tFrac16 f16CoefBuf[FIR_NUMTAPS_MAX];

#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 = 15;
            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
            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}
        }
```

5.6.5 Function GDFLIB_FilterFIR_FLT

5.6.5.1 Declaration

```
tFloat GDFLIB_FilterFIR_FLT(tFloat fltIn, const GDFLIB_FILTERFIR_PARAM_T_FLT *const pParam,
GDFLIB_FILTERFIR_STATE_T_FLT *const pState);
```

5.6.5.2 Arguments

Table 5-20. GDFLIB_FilterFIR_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input value.
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.

5.6.5.3 Return

The value of a filtered signal after processing by an FIR filter.

5.6.5.4 Variant Specifics

The function assumes that the filter order is at least one, which is equivalent to two taps. The filter also cannot contain more than 0xffffffff(hexadecimal) taps, which is equivalent to the order of 0xfffffffe (hexadecimal).

The input values are recorded by the function in the provided state structure in a circular buffer, pointed to by the state structure member pState->pInBuf. The buffer index is stored in pState->u32Idx, which points to the buffer element where a new input signal sample will be stored.

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

5.6.5.5 Code Example

```
#include "gdflib.h"

#define FIR_NUMTAPS 16
#define FIR_NUMTAPS_MAX 64
#define FIR_ORDER (FIR_NUMTAPS - 1)

GDFLIB_FILTERFIR_PARAM_T_FLT Param;
GDFLIB_FILTERFIR_STATE_T_FLT State0, State1, State2;

tFloat fltInBuf[FIR_NUMTAPS_MAX];
```

```
tFloat fltCoefBuf[FIR_NUMTAPS_MAX];

#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 = 15;
    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
    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}
              }
```

5.7 Function GDFLIB_FilterIIR1Init

This function initializes the first order IIR filter buffers.

5.7.1 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 an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

Note

This function shall not be called together with the GDFLIB_FilterIIR1 function unless periodic clearing of filter buffers is required.

5.7.2 Re-entrancy

The function is re-entrant.

5.7.3 Function GDFLIB_FilterIIR1Init_F32

5.7.3.1 Declaration

```
void GDFLIB_FilterIIR1Init_F32(GDFLIB_FILTER_IIR1_T_F32 *const pParam);
```

5.7.3.2 Arguments

Table 5-21. 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.

5.7.3.3 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);
}
```

5.7.4 Function GDFLIB_FilterIIR1Init_F16

5.7.4.1 Declaration

```
void GDFLIB_FilterIIR1Init_F16(GDFLIB_FILTER_IIR1_T_F16 *const pParam);
```

5.7.4.2 Arguments

Table 5-22. 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.

5.7.4.3 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);

    // 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);
}
```

5.7.5 Function GDFLIB_FilterIIR1Init_FLT

5.7.5.1 Declaration

```
void GDFLIB_FilterIIR1Init_FLT(GDFLIB_FILTER_IIR1_T_FLT *const pParam);
```

5.7.5.2 Arguments

Table 5-23. 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.

5.7.5.3 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);
}
```

5.8 Function GDFLIB_FilterIIR1

This function implements the first order IIR filter.

5.8.1 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**

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.

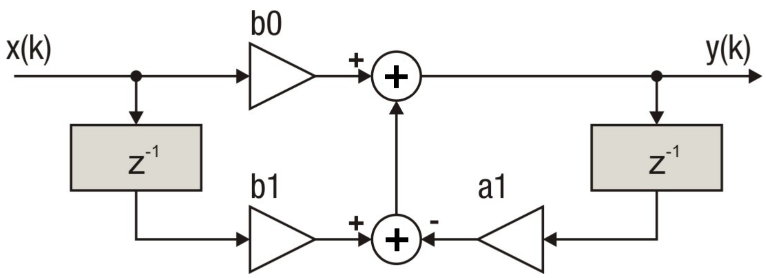

Figure 5-8. Direct Form 1 first order IIR filter

The coefficients of the filter depicted in Figure 5-8 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 an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.8.2 Re-entrancy

The function is re-entrant.

5.8.3 Function GDFLIB_FilterIIR1_F32

5.8.3.1 Declaration

```
tFrac32 GDFLIB_FilterIIR1_F32(tFrac32 f32In, GDFLIB_FILTER_IIR1_T_F32 *const pParam);
```

5.8.3.2 Arguments

Table 5-24. 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 1.31 fractional format.
GDFLIB_FILTER_IIR1_T_F32 *const	pParam	**input, output**	Pointer to the filter structure with a filter buffer and filter parameters.

5.8.3.3 Return

The function returns a 32-bit value in fractional format 1.31, representing the filtered value of the input signal in step (k).

5.8.3.4 Variant Specifics

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.

5.8.3.5 Code Example

```
#include "gdflib.h"

tFrac32 f32In;
tFrac32 f32Out;

GDFLIB_FILTER_IIR1_T_F32 f32trMyIIR1 = GDFLIB_FILTER_IIR1_DEFAULT_F32;

void main(void)
{
    // 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);
}
```

5.8.4 Function GDFLIB_FilterIIR1_F16

5.8.4.1 Declaration

```
tFrac16 GDFLIB_FilterIIR1_F16(tFrac16 f16In, GDFLIB_FILTER_IIR1_T_F16 *const pParam);
```

5.8.4.2 Arguments

Table 5-25. 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 1.15 fractional format.
GDFLIB_FILTER_IIR1_T_F16 *const	pParam	**input, output**	Pointer to the filter structure with a filter buffer and filter parameters.

5.8.4.3 Return

The function returns a 16-bit value in fractional format 1.15, representing the filtered value of the input signal in step (k).

5.8.4.4 Variant Specifics

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)

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.

5.8.4.5 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);
}
```

5.8.5 Function GDFLIB_FilterIIR1_FLT

5.8.5.1 Declaration

```
tFloat GDFLIB_FilterIIR1_FLT(tFloat fltIn, GDFLIB_FILTER_IIR1_T_FLT *const pParam);
```

5.8.5.2 Arguments

Table 5-26. 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.

5.8.5.3 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).

5.8.5.4 Variant Specifics

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 (floating-point overflow, underflow, inexact, invalid operation).

To enumerate the computational error in one calculation step, the internal accumulator is not used for testing purposes and is replaced by an output from the previous calculation step of the Matlab® precise reference model.

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.

5.8.5.5 Code Example

```
#include "gdflib.h"

tFloat fltIn;
tFloat fltOut;

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);
}
```

5.9 Function GDFLIB_FilterIIR2Init

This function initializes the second order IIR filter buffers.

5.9.1 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 an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

Note

This function shall not be called together with the GDFLIB_FilterIIR2 function unless periodic clearing of filter buffers is required.

5.9.2 Re-entrancy

The function is re-entrant.

5.9.3 Function GDFLIB_FilterIIR2Init_F32

5.9.3.1 Declaration

```
void GDFLIB_FilterIIR2Init_F32(GDFLIB_FILTER_IIR2_T_F32 *const pParam);
```

5.9.3.2 Arguments

Table 5-27. 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.

5.9.3.3 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);
}
```

5.9.4 Function GDFLIB_FilterIIR2Init_F16

5.9.4.1 Declaration

```
void GDFLIB_FilterIIR2Init_F16(GDFLIB_FILTER_IIR2_T_F16 *const pParam);
```

5.9.4.2 Arguments

Table 5-28. 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.

5.9.4.3 Code Example

```
#include "gdflib.h"

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);
}
```

5.9.5 Function GDFLIB_FilterIIR2Init_FLT

5.9.5.1 Declaration

```
void GDFLIB_FilterIIR2Init_FLT(GDFLIB_FILTER_IIR2_T_FLT *const pParam);
```

5.9.5.2 Arguments

Table 5-29. 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.

5.9.5.3 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
    GDFLIB_FilterIIR2Init (&flttrMyIIR2);
}
```

5.10 Function GDFLIB_FilterIIR2

This function implements the second order IIR filter.

5.10.1 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).

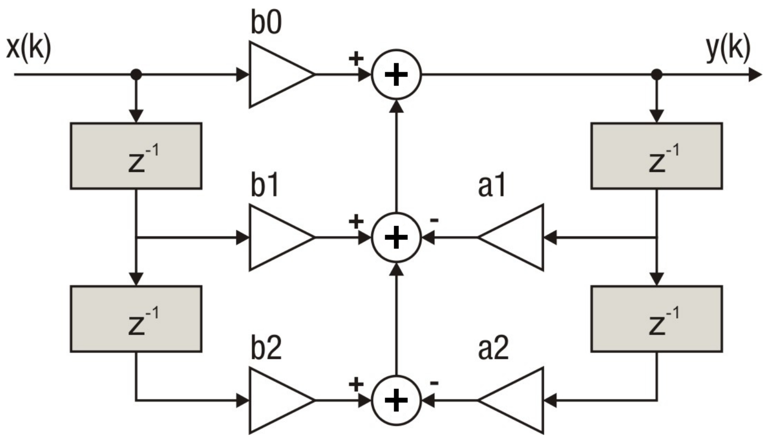

Figure 5-9. Direct Form 1 second order IIR filter

The coefficients of the filter depicted in Figure 5-9 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 an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.10.2 Re-entrancy

The function is re-entrant.

5.10.3 Function GDFLIB_FilterIIR2_F32

5.10.3.1 Declaration

```
tFrac32 GDFLIB_FilterIIR2_F32(tFrac32 f32In, GDFLIB_FILTER_IIR2_T_F32 *const pParam);
```

5.10.3.2 Arguments

Table 5-30. 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 1.31 fractional format.
GDFLIB_FILTER_IIR2_T_F32 *const	pParam	**input, output**	Pointer to the filter structure with a filter buffer and filter parameters.

5.10.3.3 Return

The function returns a 32-bit value in fractional format 1.31, representing the filtered value of the input signal in step (k).

5.10.3.4 Variant Specifics

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.

5.10.3.5 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);
    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);
}
```

5.10.4 Function GDFLIB_FilterIIR2_F16

5.10.4.1 Declaration

```
tFrac16 GDFLIB_FilterIIR2_F16(tFrac16 f16In, GDFLIB_FILTER_IIR2_T_F16 *const pParam);
```

5.10.4.2 Arguments

Table 5-31. 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 1.15 fractional format.
GDFLIB_FILTER_IIR2_T_F16 *const	pParam	**input, output**	Pointer to the filter structure with a filter buffer and filter parameters.

5.10.4.3 Return

The function returns a 16-bit value in fractional format 1.15, representing the filtered value of the input signal in step (k).

5.10.4.4 Variant Specifics

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);
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.

5.10.4.5 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);
}
```

5.10.5 Function GDFLIB_FilterIIR2_FLT

5.10.5.1 Declaration

```
tFloat GDFLIB_FilterIIR2_FLT(tFloat fltIn, GDFLIB_FILTER_IIR2_T_FLT *const pParam);
```

5.10.5.2 Arguments

Table 5-32. 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.

5.10.5.3 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).

5.10.5.4 Variant Specifics

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 (floating-point overflow, underflow, inexact, invalid operation).

To enumerate the computation error in one calculation step, the internal accumulator is not used for testing purposes and is replaced by an output from the previous calculation step of the Matlab precise reference model.

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.

5.10.5.5 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);
}
```

5.11 Function GDFLIB_FilterMAInit

This function clears the internal filter accumulator.

5.11.1 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 an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

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.

5.11.2 Re-entrancy

The function is re-entrant.

5.11.3 Function GDFLIB_FilterMAInit_F32

5.11.3.1 Declaration

```
void GDFLIB_FilterMAInit_F32(GDFLIB_FILTER_MA_T_F32 *pParam);
```

5.11.3.2 Arguments

Table 5-33. 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.

5.11.3.3 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;

  GDFLIB_FilterMAInit_F32 (&f32trMyMA);

  GDFLIB_FilterMAInit (&f32trMyMA,F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  GDFLIB_FilterMAInit (&f32trMyMA);
}
```

5.11.4 Function GDFLIB_FilterMAInit_F16

5.11.4.1 Declaration

```
void GDFLIB_FilterMAInit_F16(GDFLIB_FILTER_MA_T_F16 *pParam);
```

5.11.4.2 Arguments

Table 5-34. 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.

5.11.4.3 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);
}
```

5.11.5 Function GDFLIB_FilterMAInit_FLT

5.11.5.1 Declaration

```
void GDFLIB_FilterMAInit_FLT(GDFLIB_FILTER_MA_T_FLT *pParam);
```

5.11.5.2 Arguments

Table 5-35. 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.

5.11.5.3 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);
}
```

5.12 Function GDFLIB_FilterMA

This function implements an exponential moving average filter.

5.12.1 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_Eq2**

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_Eq3**

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 an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.12.2 Re-entrancy

The function is re-entrant.

5.12.3 Function GDFLIB_FilterMA_F32

5.12.3.1 Declaration

```
tFrac32 GDFLIB_FilterMA_F32(tFrac32 f32In, GDFLIB_FILTER_MA_T_F32 *pParam);
```

5.12.3.2 Arguments

Table 5-36. 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.

5.12.3.3 Return

The function returns a 32-bit value in format Q1.31, representing the filtered value of the input signal in step (k).

5.12.3.4 Variant Specifics

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:

$$N = 2^{u16\text{NSamples}} \quad 0 \le u16\text{NSamples} \le 31$$

Equation **GDFLIB_FilterMA_Eq4**

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 ($0 < u16NSamples < 31$). Incorrect setting of this parameter will yield meaningless results.

5.12.3.5 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
  // ##############################################################

  // output should be 0x1000000 = FRAC32(0.0078125)
  f32Output = GDFLIB_FilterMA (f32Input,&f32trMyMA);
}
```

5.12.4 Function GDFLIB_FilterMA_F16

5.12.4.1 Declaration

```
tFrac16 GDFLIB_FilterMA_F16(tFrac16 f16In, GDFLIB_FILTER_MA_T_F16 *pParam);
```

5.12.4.2 Arguments

Table 5-37. 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.

5.12.4.3 Return

The function returns a 16-bit value in format Q1.15, representing the filtered value of the input signal in step (k).

5.12.4.4 Variant Specifics

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:

$$N = 2^{u16\mathrm{NSamples}} \quad 0 \le u16\mathrm{NSamples} \le 31$$

Equation **GDFLIB_FilterMA_Eq4**

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 (0 < u16NSamples < 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.

5.12.4.5 Code Example

```
#include "gdflib.h"

tFrac16 f16Input;
tFrac16 f16Output;

GDFLIB_FILTER_MA_T_F16 f16trMyMA = GDFLIB_FILTER_MA_DEFAULT_F16;

void main(void)
{
  // 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);
  }
```

5.12.5 Function GDFLIB_FilterMA_FLT

5.12.5.1 Declaration

```
tFloat GDFLIB_FilterMA_FLT(tFloat fltIn, GDFLIB_FILTER_MA_T_FLT *pParam);
```

5.12.5.2 Arguments

Table 5-38. 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.

5.12.5.3 Return

The function returns a single precision floating point value, representing the filtered value of the input signal in step (k).

5.12.5.4 Variant Specifics

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_Eq4**

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

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.

5.12.5.5 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
  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);
}
```

5.13 Function GFLIB_Acos

This function implements an approximation of arccosine function.

5.13.1 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.

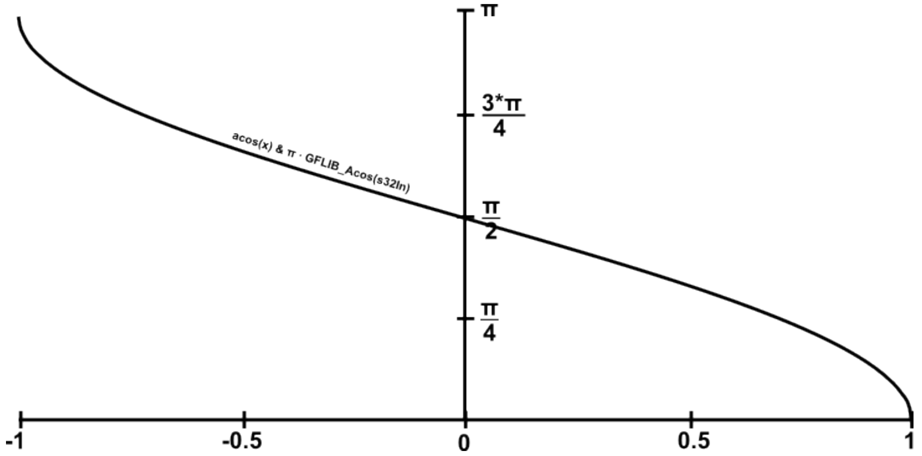

Figure 5-10. Course of the function GFLIB_Acos

Note

The input pointer must contain a valid address otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.13.2 Re-entrancy

The function is re-entrant.

5.13.3 Function GFLIB_Acos_F32

5.13.3.1 Declaration

```
tFrac32 GFLIB_Acos_F32(tFrac32 f32In, const GFLIB_ACOS_T_F32 *const pParam);
```

5.13.3.2 Arguments

Table 5-39. 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.

5.13.3.3 Return

The function returns arccos(f32In)/ π as a fixed point 32-bit number, normalized between [0,1).

5.13.3.4 Variant Specifics

The computational algorithm uses the symmetry of the arccos(x) function around the point (0, π/2), which allows 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:

$$y_{[-1,0)} = \frac{\pi}{2} + y_{[0,1)}$$

Equation **GFLIB_Acos_Eq1**

where:

- y[-1, 0) is the arccos(x) function value in the interval [-1, 0)
- y[0, 1) is the arccos(x) function value in the interval [0, 1)

Additionally, because the arccos(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]:

$$\arccos\left(\sqrt{1-x}\right) = \frac{\pi}{2} - \arccos\left(\sqrt{x}\right)$$

Equation **GFLIB_Acos_Eq2**

In this way, the computation of the arccos(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 a polynomial approximation as follows:

$$f32\mathrm{Dump} = a_0 \bullet f32\mathrm{In}^4 + a_1 \bullet f32\mathrm{In}^3 + a_2 \bullet f32\mathrm{In}^2 + a_3 \bullet f32\mathrm{In} + a_4$$

Equation **GFLIB_Acos_Eq3**

$$\arccos(f32In) = \begin{cases} -f32Dump & \text{if} \quad -1 \le f32In \le 0 \\ f32Dump & \text{if} \quad 0 \le f32In < 1 \end{cases}$$

Equation **GFLIB_Acos_Eq4**

The division of the [0,1) interval into three sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in Table 5-40.

Table 5-40. Integer polynomial coefficients for each interval

Interval	a_0	a_1	a_2	a_3	a_4
<0, 1/2)	91918582	66340080	9729967	682829947	12751
<1/2, 0.997)	-52453538	-36708911	-15136243	-964576326	1073630175
<0.997, 1)	-52453538	-36708911	-15136243	-966167437	1073739175

The implementation of the GFLIB_Acos_F32 is almost the same as in the function GFLIB_Asin_F32. However, the output of the GFLIB_Acos_F32 is corrected as follows:

$$s32Dump = FRAC32(0.5) - f32Dump$$

Equation **GFLIB_Acos_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.

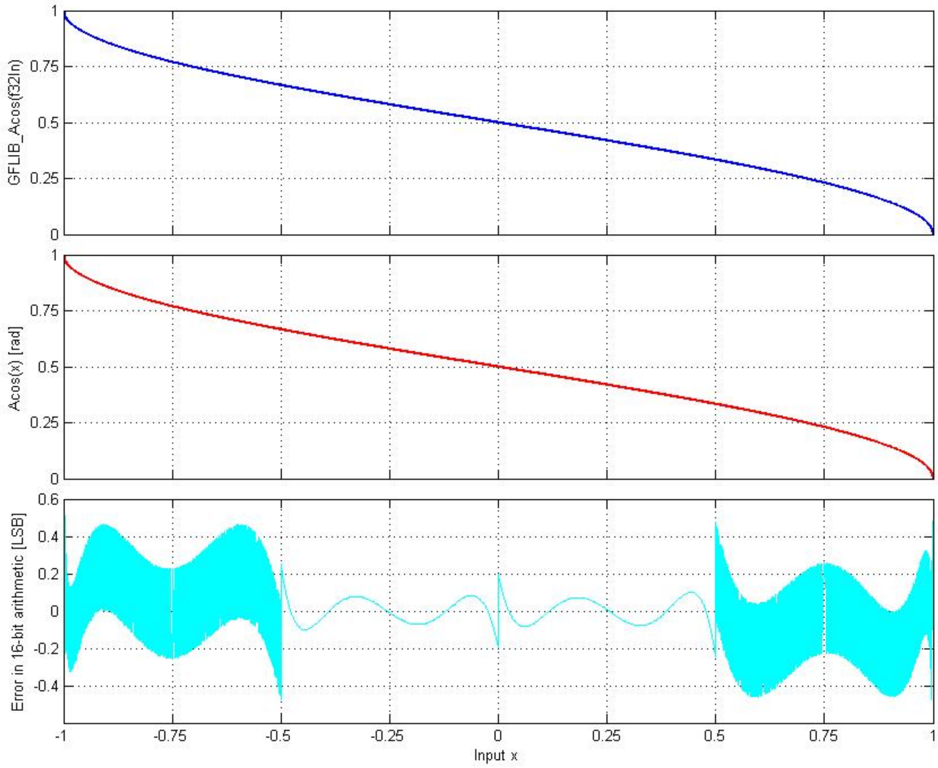

Figure 5-11. acos(x) vs. GFLIB_Acos(s32In)

Note

The function may raise floating-point exceptions (floating-point underflow, inexact, invalid operation).

The output angle is normalized into the range [0,1). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Acos_F32(f32In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ACOS_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Acos(f32In, &pParam, F32), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be

replaced with GFLIB_ACOS_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Acos(f32In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_ACOS_DEFAULT_F32 approximation coefficients are used.

5.13.3.5 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);
}
```

5.13.4 Function GFLIB_Acos_F16

5.13.4.1 Declaration

```
tFrac16 GFLIB_Acos_F16(tFrac16 f16In, const GFLIB_ACOS_T_F16 *const pParam);
```

5.13.4.2 Arguments

Table 5-41. 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.

5.13.4.3 Return

The function returns arccos(f16In)/ π as a fixed point 16-bit number, normalized between [0,1).

5.13.4.4 Variant Specifics

The computational algorithm uses the symmetry of the arccos(x) function around the point (0, π/2), which allows 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:

$$y_{[-1,0)} = \frac{\pi}{2} + y_{[0,1)}$$

Equation **GFLIB_Acos_Eq1**

where:

- y[-1, 0) is the arccos(x) function value in the interval [-1, 0)
- y[0, 1) is the arccos(x) function value in the interval [0, 1)

Additionally, because the arccos(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]:

$$\arccos\left(\sqrt{1-x}\right) = \frac{\pi}{2} - \arccos\left(\sqrt{x}\right)$$

Equation **GFLIB_Acos_Eq2**

In this way, the computation of the arccos(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 a polynomial approximation as follows:

$$f16\text{Dump} = a_0 \bullet f16\text{In}^4 + a_1 \bullet f16\text{In}^3 + a_2 \bullet f16\text{In}^2 + a_3 \bullet f16\text{In} + a_4$$

Equation **GFLIB_Acos_Eq3**

$$\arccos\left(f16\text{In}\right) = \begin{cases} -f16\text{Dump} & \text{if} \quad -1 \le f16\text{In} \le 0 \\ f16\text{Dump} & \text{if} \quad 0 \le f16\text{In} < 1 \end{cases}$$

Equation **GFLIB_Acos_Eq4**

The division of the [0,1) interval into two sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in Table 5-42.

Table 5-42. Integer polynomial coefficients for each interval

Interval	a_0	a_1	a_2	a_3	a_4
<0, 1/2)	1403	1012	148	10419	1
<1/2, 1)	-800	-560	-231	-14718	16384

The implementation of the GFLIB_Acos_F16 is almost the same as in the function GFLIB_Asin_F16. However, the output of the GFLIB_Acos_F16 is corrected as follows:

$$s16\text{Dump} = \text{FRAC16}\left(0.5\right) - f16\text{Dump}$$

Equation **GFLIB_Acos_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.

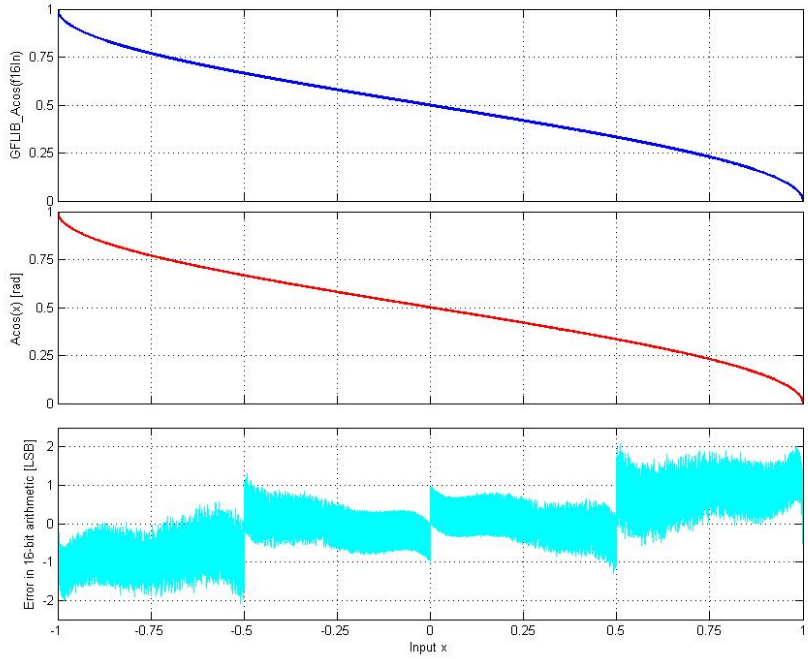

Figure 5-12. acos(x) vs. GFLIB_Acos(f16In)

Note

The function may raise floating-point exceptions (floating-point underflow, inexact, invalid operation).

The output angle is normalized into the range [0,1). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Acos_F16(f16In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ACOS_DEFAULT_F16 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Acos(f16In, &pParam, F16), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ACOS_DEFAULT_F16 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Acos(f16In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default

GFLIB_ACOS_DEFAULT_F16 approximation coefficients are used.

5.13.4.5 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);
}
```

5.13.5 Function GFLIB_Acos_FLT

5.13.5.1 Declaration

```
tFloat GFLIB_Acos_FLT(tFloat fltIn, const GFLIB_ACOS_T_FLT *const pParam);
```

5.13.5.2 Arguments

Table 5-43. 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. The function alias GFLIB_Acos uses the default coefficients.

5.13.5.3 Return

The function returns arccos(fltIn) as a single precision floating point number.

5.13.5.4 Variant Specifics

The computation algorithm for arccos(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 calculate the function values in the interval [-1, 0) use the simple formula:

$$y_{[-1,0)} = \frac{\pi}{2} + y_{[0,1)}$$

Equation **GFLIB_Acos_Eq1**

where:

- y[-1, 0) is the arccos(x) function value in the interval [-1, 0)
- y[0, 1) is the arccos(x) function value in the interval [0, 1)

For the interval [0, 1], the algorithm uses a polynomial approximation as follows:

$$\arcsin(\text{fltIn}) = \frac{\pi}{2} - \sqrt{1 - \text{fltIn}} \bullet \left(a_4 + a_3 \bullet \text{fltIn} + a_2 \bullet \text{fltIn}^2 + a_1 \bullet \text{fltIn}^3 + a_0 \bullet \text{fltIn}^4\right)$$

Equation **GFLIB_Acos_Eq2**

To calculate the values of the arcsin(x) function in interval [-1, 0), the symmetry of the arcsin(x) function is used and the value is determined by the following formula:

$$\arcsin(\text{fltIn})_{[-1,0)} = -\arcsin(\text{fltIn})_{(0,1]}$$

Equation **GFLIB_Acos_Eq3**

The arccos(x) values are then calculated by the simple formula:

$$\arccos(x) = \frac{\pi}{2} - \arcsin(x)$$

Equation **GFLIB_Acos_Eq4**

Polynomial approximation coefficients used for GFLIB_Acos_FLT calculation are noted in Table 5-44.

Table 5-44. Approximation polynomial coefficients

a_0	a_1	a_2	a_3	a_4
7.6983864e-03	-3.3877850e-02	8.3511069e-02	-2.1389899e-01	1.5707811e+00

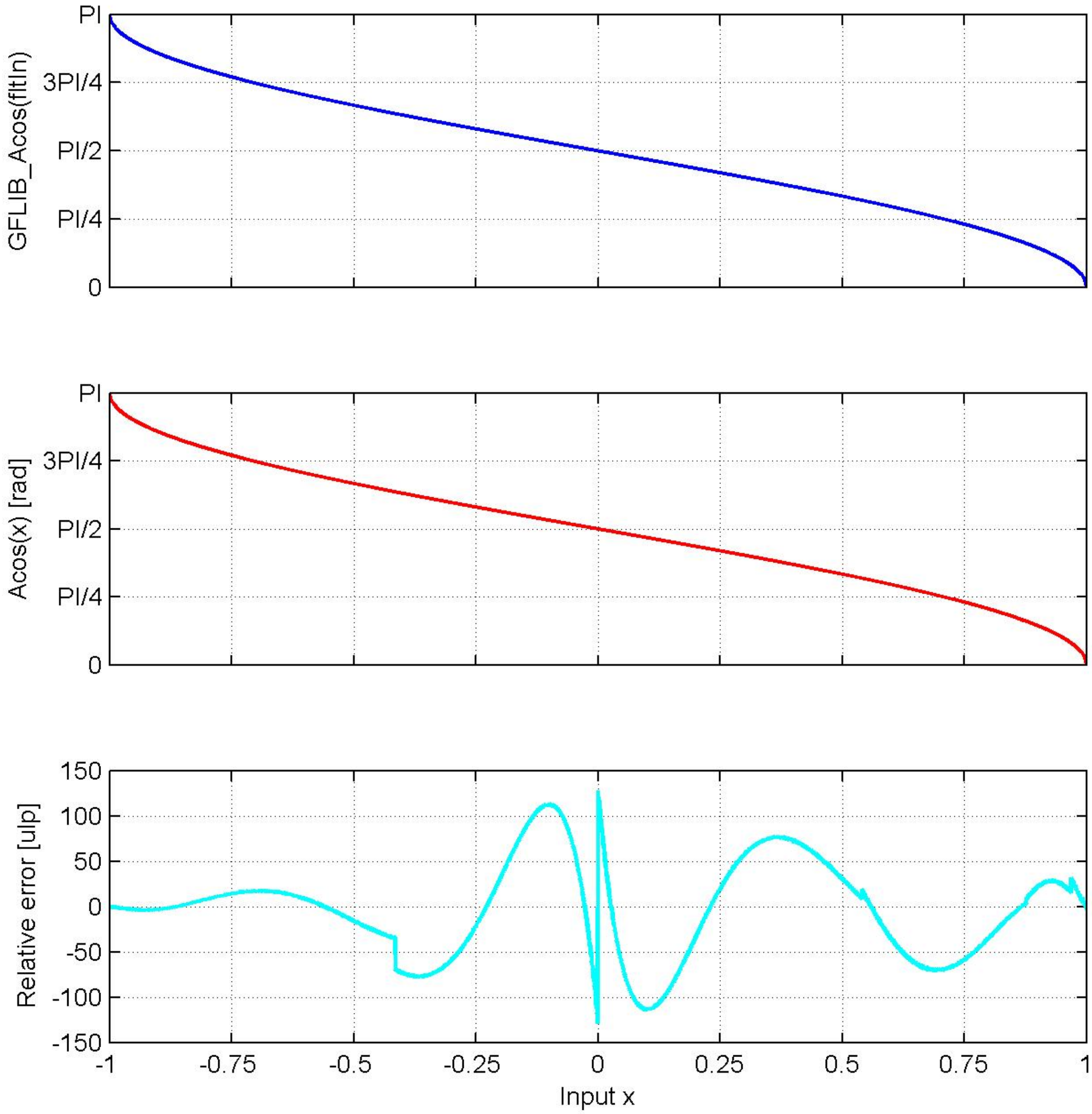

Figure 5-13. acos(x) vs. GFLIB_Acos(s32In)

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation, zero divide).

The input value is assumed to be in the interval [-1, 1]. The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Acos_FLT(fltIn, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ACOS_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Acos(fltIn, &pParam, FLT), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ACOS_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Acos(fltIn, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_ACOS_DEFAULT_FLT approximation coefficients are used.

5.13.5.5 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);
}
```

5.14 Function GFLIB_Asin

This function implements polynomial approximation of arcsine function.

5.14.1 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.

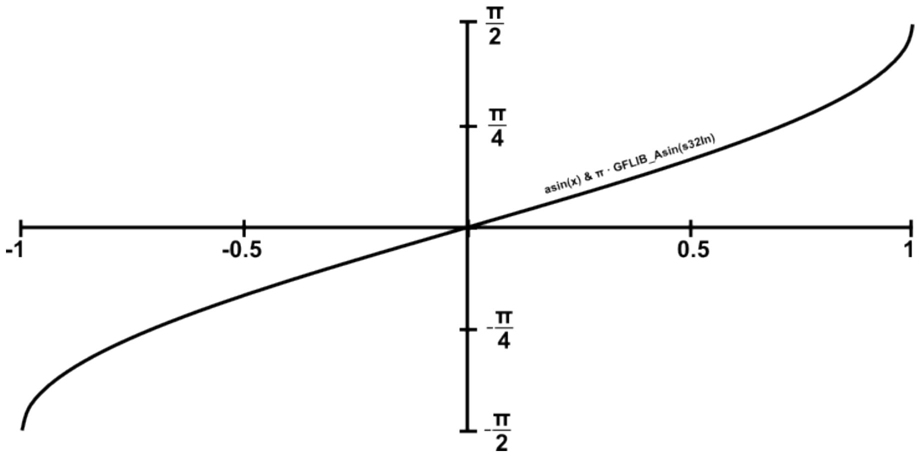

Figure 5-14. Course of the function GFLIB_Asin

Note

The input pointer must contain a valid address otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.14.2 Re-entrancy

The function is re-entrant.

5.14.3 Function GFLIB_Asin_F32

5.14.3.1 Declaration

```
tFrac32 GFLIB_Asin_F32(tFrac32 f32In, const GFLIB_ASIN_T_F32 *const pParam);
```

5.14.3.2 Arguments

Table 5-45. 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.

5.14.3.3 Return

The function returns arcsin(f32In)/ π as a fixed point 32-bit number, normalized between [-1,1).

5.14.3.4 Variant Specifics

The computational algorithm uses the symmetry of the arcsin(x) function around the point (0, π/2), 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:

$$y_{[-1,0)} = -y_{[0,1)}$$

Equation **GFLIB_Asin_Eq1**

where:

- y[-1, 0) is the arcsin(x) function value in the interval [-1, 0)
- y[1, 0) is the arcsin(x) function value in the interval [1, 0)

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] :

$$\arcsin\left(\sqrt{1-x}\right) = \frac{\pi}{2} - \arcsin\left(\sqrt{x}\right)$$

Equation **GFLIB_Asin_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:

$$f32\mathrm{Dump} = a_0 \bullet f32\mathrm{In}^4 + a_1 \bullet f32\mathrm{In}^3 + a_2 \bullet f32\mathrm{In}^2 + a_3 \bullet f32\mathrm{In} + a_4$$

Equation **GFLIB_Asin_Eq3**

$$\arcsin\left(f32\mathrm{In}\right) = \begin{cases} -f32\mathrm{Dump} & \text{if} \quad -1 \le f32\mathrm{In} \le 0 \\ f32\mathrm{Dump} & \text{if} \quad 0 \le f32\mathrm{In} < 1 \end{cases}$$

Equation **GFLIB_Asin_Eq4**

The division of the [0,1) interval into two sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in Table 5-46.

Table 5-46. Integer polynomial coefficients for each interval

Interval	a_0	a_1	a_2	a_3	a_4
<0 , 1/2)	91918582	66340080	9729967	682829947	12751
<1/2, 0.997)	-52453538	-36708911	-15136243	-964576326	1073630175
<0.997, 1)	-52453538	-36708911	-15136243	-966167437	1073739175

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,:),'-');
    coef = round(p * (2^31))
```

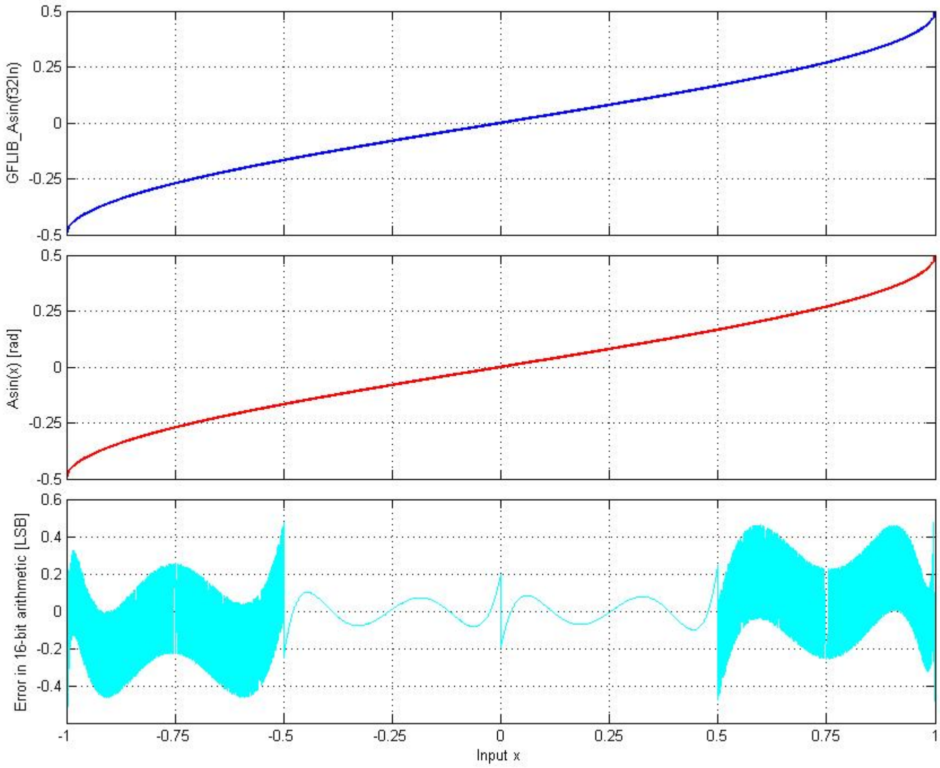

Figure 5-15. asin(x) vs. GFLIB_Asin(f32In)

Note

The function may raise floating-point exceptions (floating-point underflow, inexact, invalid operation).

The output angle is normalized into the range [-0.5,0.5). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Asin_F32(f32In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ASIN_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Asin(f32In, &pParam, F32), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ASIN_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Asin(f32In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_ASIN_DEFAULT_F32 approximation coefficients are used.

5.14.3.5 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);
}
```

5.14.4 Function GFLIB_Asin_F16

5.14.4.1 Declaration

```
tFrac16 GFLIB_Asin_F16(tFrac16 f16In, const GFLIB_ASIN_T_F16 *const pParam);
```

5.14.4.2 Arguments

Table 5-47. 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.

5.14.4.3 Return

The function returns arcsin(f16In)/ π as a fixed point 16-bit number, normalized between [-1,1).

5.14.4.4 Variant Specifics

The computational algorithm uses the symmetry of the arcsin(x) function around the point (0, π/2), 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:

$$y_{[-1,0)} = -y_{[0,1)}$$

Equation **GFLIB_Asin_Eq1**

where:

- y[-1, 0) is the arcsin(x) function value in the interval [-1, 0)
- y[1, 0) is the arcsin(x) function value in the interval [1, 0)

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]:

$$\arcsin\left(\sqrt{1-x}\right) = \frac{\pi}{2} - \arcsin\left(\sqrt{x}\right)$$

Equation **GFLIB_Asin_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:

$$f16\mathrm{Dump} = a_0 \bullet f16\mathrm{In}^4 + a_1 \bullet f16\mathrm{In}^3 + a_2 \bullet f16\mathrm{In}^2 + a_3 \bullet f16\mathrm{In} + a_4$$

Equation **GFLIB_Asin_Eq3**

$$\arcsin\left(f16\mathrm{In}\right) = \begin{cases} -f16\mathrm{Dump} & \text{if} \quad -1 \le f16\mathrm{In} \le 0 \\ f16\mathrm{Dump} & \text{if} \quad 0 \le f16\mathrm{In} < 1 \end{cases}$$

Equation **GFLIB_Asin_Eq4**

The division of the [0,1) interval into two sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in Table 5-48.

Table 5-48. Integer polynomial coefficients for each interval

Interval	a_0	a_1	a_2	a_3	a_4
<0 , 1/2)	1403	1012	148	10419	1
<1/2, 1)	-800	-560	-231	-14718	16384

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,:),'-');
coef = round(p * (2^31))
```

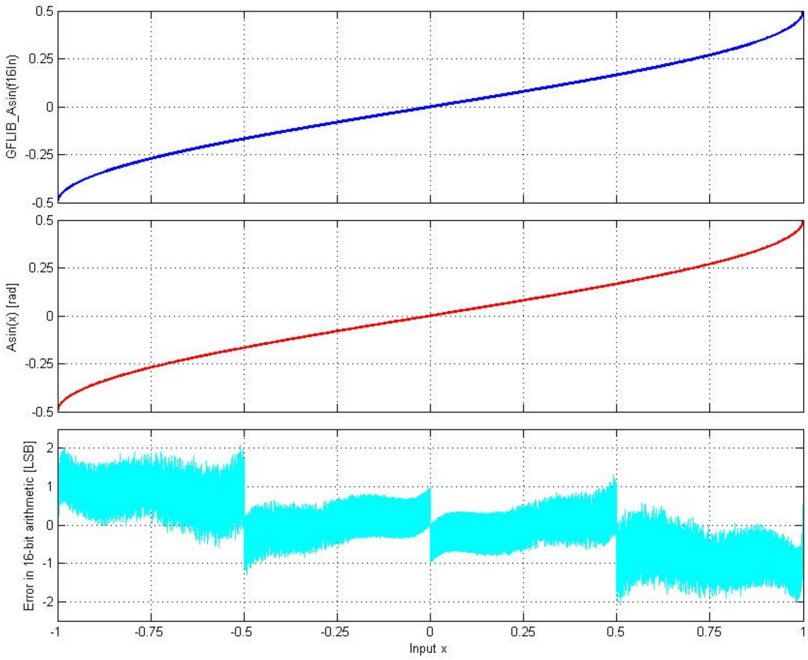

Figure 5-16. asin(x) vs. GFLIB_Asin(f16In)

Note

The function may raise floating-point exceptions (floating-point underflow, inexact, invalid operation).

The output angle is normalized into the range [-0.5,0.5). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Asin_F16(f16In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ASIN_DEFAULT_F16 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Asin(f16In, &pParam, F16), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ASIN_DEFAULT_F16 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Asin(f16In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_ASIN_DEFAULT_F16 approximation coefficients are used.

5.14.4.5 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);
}
```

5.14.5 Function GFLIB_Asin_FLT

5.14.5.1 Declaration

```
tFloat GFLIB_Asin_FLT(tFloat fltIn, const GFLIB_ASIN_T_FLT *const pParam);
```

5.14.5.2 Arguments

Table 5-49. 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.

5.14.5.3 Return

The function returns arcsin(fltIn) as a single precision floating point number.

5.14.5.4 Variant Specifics

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:

$$y_{[-1,0)} = -y_{[0,1)}$$

Equation **GFLIB_Asin_Eq1**

where:

- y[-1, 0) is the arcsin(x) function value in the interval [-1, 0)
- y[0, 1] is the arcsin(x) function value in the interval [0, 1]

For the interval [0, 0.41), the algorithm uses a Minimax approximation as follows:

$$\arcsin(\mathrm{fltIn}) = a_0 \bullet \mathrm{fltIn} + a_1 \bullet \mathrm{fltIn}^3 + a_2 \bullet \mathrm{fltIn}^5$$

Equation **GFLIB_Asin_Eq2**

For the interval [0.41, 1), the algorithm uses a polynomial approximation as follows.

$$\arcsin(\mathrm{fltIn}) = \frac{\pi}{2} - \sqrt{1 - \mathrm{fltIn}} \bullet \left(a_3 + a_2 \bullet \mathrm{fltIn} + a_1 \bullet \mathrm{fltIn}^2 + a_0 \bullet \mathrm{fltIn}^3\right)$$

Equation **GFLIB_Asin_Eq3**

To calculate the values of the arcsin(x) function in interval [-1, 0), the symmetry of the arcsin(x) function is used and the value is determined by the following formula:

$$\arcsin(\mathrm{fltIn})_{[-1,0)} = -\arcsin(\mathrm{fltIn})_{[0,1]}$$

Equation **GFLIB_Asin_Eq4**

Minimax approximation coefficients used for GFLIB_Asin_FLT calculation in interval [0, 0.41), are noted in Table 5-50.

Table 5-50. Approximation polynomial coefficients

a_0	a_1	a_2
1.0000083e+00	1.6580229e-01	8.8028289e-02

In Table 5-51 there are the approximation coefficients used for GFLIB_Asin_FLT calculation in interval [0.41, 1).

Table 5-51. Approximation polynomial coefficients

a_0	a_1	a_2	a_3
-1.2426286e-02	6.1917335e-02	-2.0464350e-01	1.5693614e+00

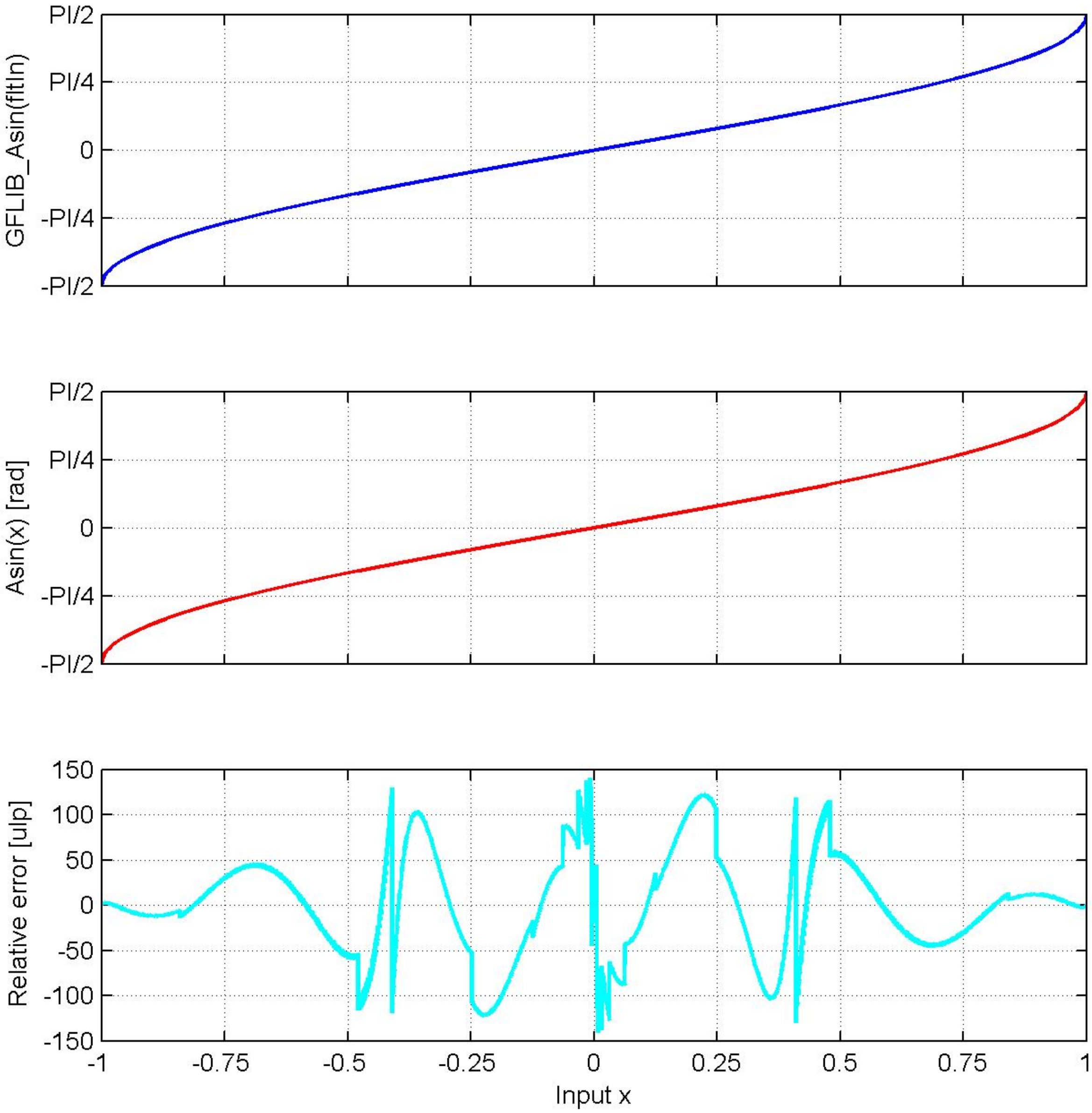

Figure 5-17. asin(x) vs. GFLIB_Asin(f32In)

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation, zero divide).

The input value is assumed to be in the interval [-1, 1] .

The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Asin_FLT(fltIn, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ASIN_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Asin(fltIn, &pParam, FLT), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ASIN_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Asin(fltIn, &pParam), where &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_ASIN_DEFAULT_FLT approximation coefficients are used.

5.14.5.5 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);
}
```

5.15 Function GFLIB_Atan

This function implements minimax polynomial approximation of arctangent function.

5.15.1 Description

The GFLIB_Atan_F32 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 5-18.

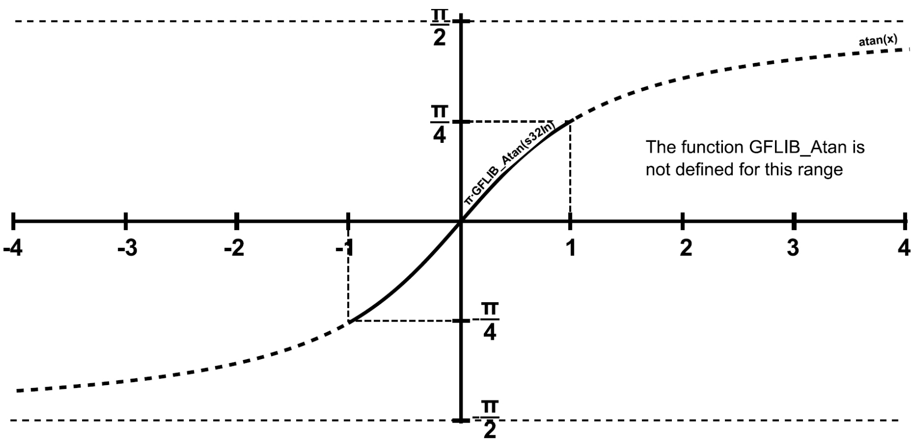

Figure 5-18. Course of the function GFLIB_Atan

Note

The input pointer must contain a valid address otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.15.2 Re-entrancy

The function is re-entrant.

5.15.3 Function GFLIB_Atan_F32

5.15.3.1 Declaration

```
tFrac32 GFLIB_Atan_F32(tFrac32 f32In, const GFLIB_ATAN_T_F32 *const pParam);
```

5.15.3.2 Arguments

Table 5-52. 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.

5.15.3.3 Return

The function returns the atan of the input argument as a fixed point 32-bit number that contains the angle in radians between [- π/4, π/4), normalized between [-0.25,0.25).

5.15.3.4 Variant Specifics

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 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:

$$f32\mathrm{Dump} = a_1 f32\mathrm{In}^2 + a_2 f32\mathrm{In} + a_3$$

Equation **GFLIB_Atan_Eq1**

$$\arctan\left(f32\mathrm{In}\right) = \begin{cases} f32\mathrm{Dump} & \text{if} \quad 0 \le f32\mathrm{In} < 1 \\ -f32\mathrm{Dump} & \text{if} \quad -1 \le f32\mathrm{In} < 0 \end{cases}$$

Equation **GFLIB_Atan_Eq2**

The division of the [0, 1) interval into eight sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in Table 5-53.

Table 5-53. Integer minimax polynomial coefficients for each interval

Interval	a_1	a_2	a_3
<0, 1/8)	-164794	42515925	42667172
<1/8, 2/8)	-465182	41238272	126697014
<2/8, 3/8)	-690034	38899574	207041074
<3/8, 4/8)	-820713	35848645	281909001
<4/8, 5/8)	-865105	32453241	350251355
<5/8, 6/8)	-845462	29016149	411702516
<6/8, 7/8)	-786689	25743137	466407809
<7/8, 1)	-708969	22748418	514828039

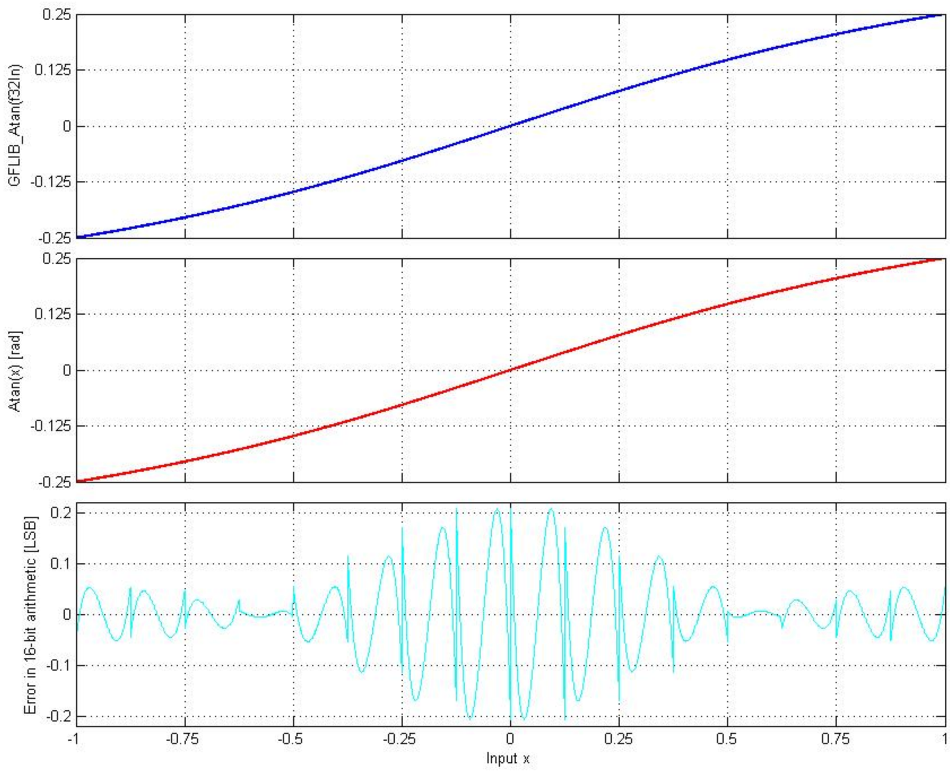

Figure 5-19. atan(x) vs. GFLIB_Atan(f32In)

Figure 5-19 depicts a floating point *arctangent* function generated from Matlab and the approximated value of the *arctangent* function obtained from GFLIB_Atan_F32, plus their difference. The course of calculation accuracy as a function of the input value can be observed from this figure. The achieved accuracy with consideration to the 3rd order piece-wise minimax polynomial approximation and described fixed point scaling is less than 0.5LSB on the upper 16 bits of the 32-bit result.

Note

The output angle is normalized into the range [-0.25, 0.25). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Atan_F32(f32In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ATAN_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Atan(f32In, &pParam, F32), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ATAN_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Atan(f32In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_ATAN_DEFAULT_F32 approximation coefficients are used.

5.15.3.5 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);
    }
```

5.15.4 Function GFLIB_Atan_F16

5.15.4.1 Declaration

```
tFrac16 GFLIB_Atan_F16(tFrac16 f16In, const GFLIB_ATAN_T_F16 *const pParam);
```

5.15.4.2 Arguments

Table 5-54. 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.

5.15.4.3 Return

The function returns the atan of the input argument as a fixed point 16-bit number that contains the angle in radians between [- π/4, π/4), normalized between [-0.25, 0.25).

5.15.4.4 Variant Specifics

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:

$$f16\text{Dump} = a_1 f16\text{In}^2 + a_2 f16\text{In} + a_3$$

Equation **GFLIB_Atan_Eq1**

$$\arctan\left(f16\text{In}\right) = \begin{cases} f16\text{Dump} & \text{if} \quad 0 \le f16\text{In} < 1 \\ -f16\text{Dump} & \text{if} \quad -1 \le f16\text{In} < 0 \end{cases}$$

Equation **GFLIB_Atan_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 5-55.

Table 5-55. Integer polynomial coefficients for each interval

Interval	a_1	a_2	a_3
<0, 1/8)	-3	649	652
<1/8, 2/8)	-7	630	1934
<2/8, 3/8)	-11	594	3160
<3/8, 4/8)	-13	547	4302
<4/8, 5/8)	-13	495	5345
<5/8, 6/8)	-13	443	6283
<6/8, 7/8)	-12	393	7117
<7/8, 1)	-11	347	7856

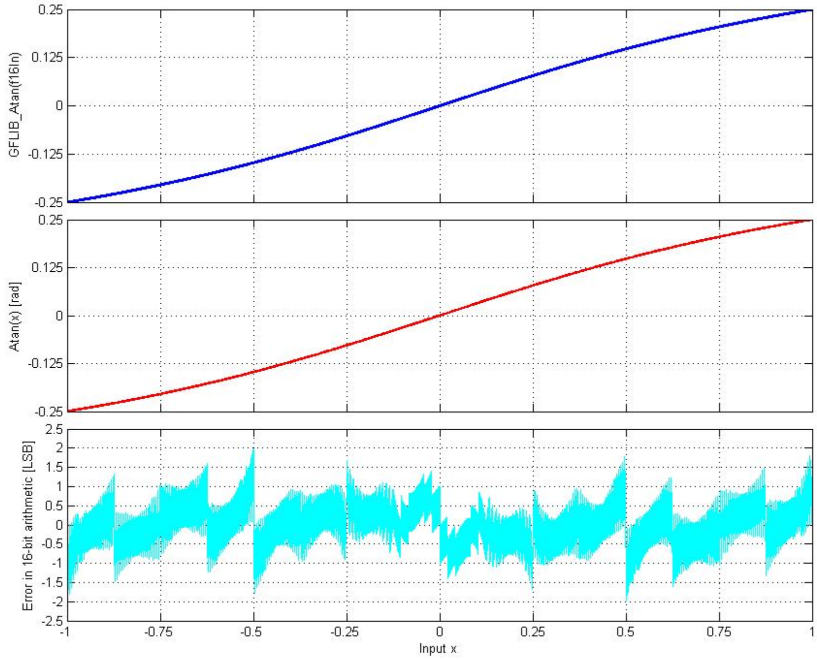

Figure 5-20. atan(x) vs. GFLIB_Atan(f16In)

Note

The output angle is normalized into the range [-0.25,0.25). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Atan_F16(f16In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ATAN_DEFAULT_F16 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Atan(f16In, &pParam, F16), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ATAN_DEFAULT_F16 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Atan(f16In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_ATAN_DEFAULT_F16 approximation coefficients are used.

5.15.4.5 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);
}
```

5.15.5 Function GFLIB_Atan_FLT

5.15.5.1 Declaration

```
tFloat GFLIB_Atan_FLT(tFloat fltIn, const GFLIB_ATAN_T_FLT *const pParam);
```

5.15.5.2 Arguments

Table 5-56. 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}).
const GFLIB_ATAN_T_FLT *const	pParam	**input**	Pointer to an array of rational polynomial coefficients.

5.15.5.3 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).

5.15.5.4 Variant Specifics

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

$$\mathrm{fltOut} = \frac{\mathrm{fltIn} \bullet \left(a_1 + \mathrm{fltIn}^2\left(a_2 + a_3 \bullet \mathrm{fltIn}^2\right)\right)}{a_4 + \mathrm{fltIn}^2\left(a_5 + \mathrm{fltIn}^2\left(a_6 + \mathrm{fltIn}^2\right)\right)}$$

Equation **GFLIB_Atan_Eq1**

The rational polynomial coefficients are noted in Table 5-57.

Table 5-57. Rational polynomial approximation coefficients

a_1	a_2	a_3	a_4	a_5	a_6
48.70107004404890	49.53263287722540	9.4060435423162400	65.76632705089560	21.58787016702020	48.70107004404990

Input values in the interval (tan(π/12), 1] are transformed into the base interval [0, tan(π/12)] as follows:

$$\mathrm{fltIn} = \frac{\mathrm{fltIn} - \tan\left(\frac{\pi}{6}\right)}{1 + \mathrm{fltIn} \bullet \tan\left(\frac{\pi}{6}\right)} \quad \text{if} \quad |\mathrm{fltIn}| > \tan\left(\frac{\pi}{12}\right)$$

Equation **GFLIB_Atan_Eq2**

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.

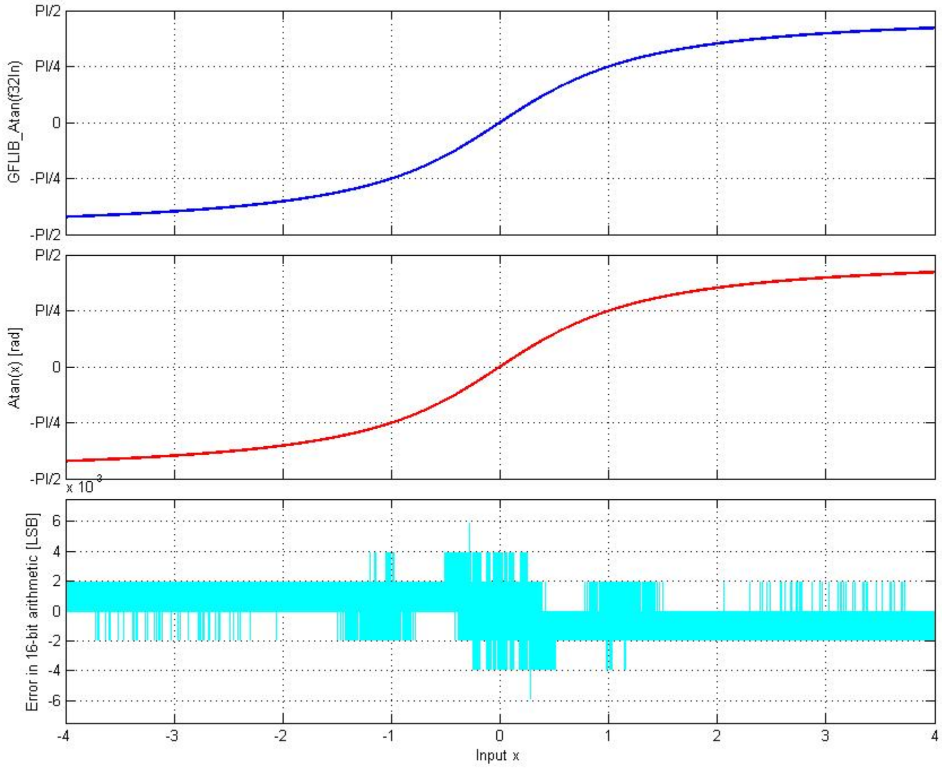

Figure 5-21. atan(x) vs. GFLIB_Atan_FLT(fltIn)

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation, zero divide).

The input value is assumed to be in the interval (-2^{128}, 2^{128}). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Atan_FLT(fltIn, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ATAN_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Atan(fltIn, &pParam, FLT), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be

replaced with GFLIB_ATAN_DEFAULT_FLT symbol. The &pParam parameter is mandatory.

- With preselected default implementation (i.e. GFLIB_Atan(fltIn, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_ATAN_DEFAULT_FLT approximation coefficients are used.

5.15.5.5 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);

}
```

5.16 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.

5.16.1 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).

5.16.2 Re-entrancy

The function is re-entrant.

5.16.3 Function GFLIB_AtanYX_F32

5.16.3.1 Declaration

```
tFrac32 GFLIB_AtanYX_F32(tFrac32 f32InY, tFrac32 f32InX);
```

5.16.3.2 Arguments

Table 5-58. 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).

5.16.3.3 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.

5.16.3.4 Variant Specifics

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).

5.16.3.5 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
    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);
}
```

5.16.4 Function GFLIB_AtanYX_F16

5.16.4.1 Declaration

```
tFrac16 GFLIB_AtanYX_F16(tFrac16 f16InY, tFrac16 f16InX);
```

5.16.4.2 Arguments

Table 5-59. 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).

5.16.4.3 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.

5.16.4.4 Variant Specifics

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.

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).

5.16.4.5 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);
}
```

5.16.5 Function GFLIB_AtanYX_FLT

5.16.5.1 Declaration

```
tFloat GFLIB_AtanYX_FLT(tFloat fltInY, tFloat fltInX);
```

5.16.5.2 Arguments

Table 5-60. 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).

5.16.5.3 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.

5.16.5.4 Variant Specifics

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 $[-\pi, \pi]$. 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 (floating-point overflow, underflow, inexact, invalid operation, zero divide).

The function calls the GFLIB_Atan_FLT function. The computed value is within the range of $[-\pi, \pi]$.

5.16.5.5 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);
}
```

5.17 Function GFLIB_AtanYXShifted

This function calculates the angle of two sine waves shifted in phase to each other. This function calculates the angle of two sine waves shifted in phase to each other.

5.17.1 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
- $\Delta\theta$ 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 an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.17.2 Re-entrancy

The function is re-entrant.

5.17.3 Function GFLIB_AtanYXShifted_F32

5.17.3.1 Declaration

```
tFrac32 GFLIB_AtanYXShifted_F32(tFrac32 f32InY, tFrac32 f32InX, const
GFLIB_ATANYXSHIFTED_T_F32 *pParam);
```

5.17.3.2 Arguments

Table 5-61. 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, output**	The parameters for the function.

5.17.3.3 Return

The function returns the angle of two sine waves shifted in phase to each other.

5.17.3.4 Variant Specifics

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 structure GFLIB_ATANYXSHIFTED_T_F32, need to be computed by means of the provided Matlab function (see below).

If $\Delta\theta = \pi/2$ or $\Delta\theta = -\pi/2$, 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 $\Delta\theta$ and θ_{Offset}, the $\Delta\theta$ 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:

$$b = \frac{S}{2\cos\left(\frac{\Delta\theta}{2}\right)}\left(y + x\right)$$

$$a = \frac{S}{2\sin\left(\frac{\Delta\theta}{2}\right)}\left(x - y\right)$$

$$\theta = \mathrm{AtanYX}\left(b, a\right) - \left(\frac{\Delta\theta}{2} - \theta_{offset}\right)$$

Equation **GFLIB_AtanYXShifted_Eq2**

where:

- x, y are respectively, the f32InX, and f32InY
- θ is the angle to be computed by the function, see the previous equation
- $\Delta\theta$ 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 $\theta + \theta_{Offset}$

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:

$$\tan\left(\theta + \Delta\theta\right) = \frac{(y+x)\cos\frac{\Delta\theta}{2}}{(x-y)\sin\frac{\Delta\theta}{2}}$$

Equation **GFLIB_AtanYXShifted_Eq3**

For the purposes of fractional arithmetic, the algorithm is implemented such that additional values are used as shown in the equation below:

$$\frac{S}{2\cos\left(\frac{\Delta\theta}{2}\right)} = C_y = K_y 2^{N_y}$$

$$\frac{S}{2\sin\left(\frac{\Delta\theta}{2}\right)} = C_x = K_x 2^{N_x}$$

$$\theta_{\mathrm{adj}} = \frac{\Delta\theta}{2} - \theta_{\mathrm{offset}}$$

Equation **GFLIB_AtanYXShifted_Eq4**

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;
        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 $\Delta\theta$ 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.

5.17.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32InY;
tFrac32 f32InX;
tFrac32 f32Ang;
GFLIB_ATANYXSHIFTED_T_F32 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.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);
    }
```

5.17.4 Function GFLIB_AtanYXShifted_F16

5.17.4.1 Declaration

```
tFrac16 GFLIB_AtanYXShifted_F16(tFrac16 f16InY, tFrac16 f16InX, const
GFLIB_ATANYXSHIFTED_T_F16 *pParam);
```

5.17.4.2 Arguments

Table 5-62. 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, output**	The parameters for the function.

5.17.4.3 Return

The function returns the angle of two sine waves shifted in phase to each other.

5.17.4.4 Variant Specifics

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 $\Delta\theta = \pi/2$ or $\Delta\theta = -\pi/2$, 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 $\Delta\theta$ and θ_{Offset}, the $\Delta\theta$ 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:

$$b = \frac{S}{2\cos\left(\frac{\Delta\theta}{2}\right)}\left(y + x\right)$$

$$a = \frac{S}{2\sin\left(\frac{\Delta\theta}{2}\right)}\left(x - y\right)$$

$$\theta = \mathrm{AtanYX}\left(b, a\right) - \left(\frac{\Delta\theta}{2} - \theta_{\mathrm{offset}}\right)$$

Equation **GFLIB_AtanYXShifted_Eq2**

where:

- x, y are respectively, the f16InX, and f16InY
- θ is the angle to be computed by the function, see the previous equation
- $\Delta\theta$ 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 $\theta + \theta_{Offset}$

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:

$$\tan\left(\theta + \Delta\theta\right) = \frac{\left(y+x\right)\cos\frac{\Delta\theta}{2}}{\left(x-y\right)\sin\frac{\Delta\theta}{2}}$$

Equation **GFLIB_AtanYXShifted_Eq3**

For the purposes of fractional arithmetic, the algorithm is implemented such that additional values are used as shown in the equation below:

$$\frac{S}{2\cos\left(\frac{\Delta\theta}{2}\right)} = C_y = K_y 2^{N_y}$$

$$\frac{S}{2\sin\left(\frac{\Delta\theta}{2}\right)} = C_x = K_x 2^{N_x}$$

$$\theta_{\mathrm{adj}} = \frac{\Delta\theta}{2} - \theta_{\mathrm{offset}}$$

Equation **GFLIB_AtanYXShifted_Eq4**

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;
        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 $\Delta\theta$ 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.

5.17.4.5 Code Example

```
#include "gflib.h"

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);
    }
```

5.17.5 Function GFLIB_AtanYXShifted_FLT

5.17.5.1 Declaration

```
tFloat GFLIB_AtanYXShifted_FLT(tFloat fltInY, tFloat fltInX, const GFLIB_ATANYXSHIFTED_T_FLT
*pParam);
```

5.17.5.2 Arguments

Table 5-63. GFLIB_AtanYXShifted_FLT arguments

Type	Name	Direction	Description
tFloat	fltInY	**input**	The value of the first signal, assumed to be sin(θ).
tFloat	fltInX	**input**	The value of the second signal, assumed to be sin(θ + Δθ).
const GFLIB_ATANYXSHIFTED_T_FLT *	pParam	**input, output**	The parameters for the function.

5.17.5.3 Return

The function returns the angle of two sine waves shifted in phase to each other.

5.17.5.4 Variant Specifics

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 $\Delta\theta = \pi/2$ or $\Delta\theta = -\pi/2$, 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 $\Delta\theta$ and θ_{Offset}, the $\Delta\theta$ 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:

$$b = \frac{1}{2\cos\left(\frac{\Delta\theta}{2}\right)}\left(y + x\right)$$

$$a = \frac{1}{2\sin\left(\frac{\Delta\theta}{2}\right)}\left(x - y\right)$$

$$\theta = \mathrm{AtanYX}\left(b, a\right) - \left(\frac{\Delta\theta}{2} - \theta_{\mathrm{offset}}\right)$$

Equation **GFLIB_AtanYXShifted_Eq2**

where:

- x, y are respectively, the fltInX, and fltInY
- θ is the angle to be computed by the function, see the previous equation
- $\Delta\theta$ 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 $\theta + \theta_{Offset}$

The algorithm can be easily justified by proving the trigonometric identity:

$$\tan\left(\theta + \Delta\theta\right) = \frac{(y+x)\cos\frac{\Delta\theta}{2}}{(x-y)\sin\frac{\Delta\theta}{2}}$$

Equation **GFLIB_AtanYXShifted_Eq3**

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:

$$\frac{1}{2\cos\left(\frac{\Delta\theta}{2}\right)} = K_y$$

$$\frac{1}{2\sin\left(\frac{\Delta\theta}{2}\right)} = K_x$$

$$\theta_{\mathrm{adj}} = \frac{\Delta\theta}{2} - \theta_{\mathrm{offset}}$$

Equation **GFLIB_AtanYXShifted_Eq4**

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;

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 (floating-point overflow, underflow, inexact, invalid operation, zero divide).

5.17.5.5 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);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // Output angle should be 0.69228911 rad
    fltAng = GFLIB_AtanYXShifted (fltInY, fltInX, &Param);
}
```

5.18 Function GFLIB_ControllerPIp

This function calculates a parallel form of the Proportional-Integral controller, without integral anti-windup.

5.18.1 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:

$$u(t) = e(t) \bullet K_P + K_I \int_0^t e(t) \mathrm{dt}$$

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 transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = K_P + K_I \frac{1}{s}$$

Equation **GFLIB_ControllerPIp_Eq2**

The proportional part of equation GFLIB_ControllerPIp_Eq2 is transformed into the discrete time domain simply as:

$$u_P(k) = K_P \bullet e(k)$$

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:

$$u_I(k) = u_I(k-1) + e(k) \bullet \frac{K_I T_s}{2} + e(k-1) \bullet \frac{K_I T_s}{2}$$

Equation **GFLIB_ControllerPIp_Eq4**

where T_s [sec] is the sampling time.

Note

The input pointer must contain a valid address otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.18.2 Re-entrancy

The function is re-entrant.

5.18.3 Function GFLIB_ControllerPIp_F32

5.18.3.1 Declaration

```
tFrac32 GFLIB_ControllerPIp_F32(tFrac32 f32InErr, GFLIB_CONTROLLER_PI_P_T_F32 *const pParam);
```

5.18.3.2 Arguments

Table 5-64. 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.

5.18.3.3 Return

The function returns a 32-bit value in format 1.31, representing the signal to be applied to the controlled system so that the input error is forced to zero.

5.18.3.4 Variant Specifics

In order to implement the discrete equation of the controller on the fixed point arithmetic platform, the maximal values (scales) of the input and output signals have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values:

- 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:

$$e_f(k) = \frac{e(k)}{E^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIp_Eq5**

$$u_f(k) = \frac{u(k)}{U^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIp_Eq6**

Applying such scaling (normalization) on the proportional term of equation GFLIB_ControllerPIp_Eq3 results in:

$$u_{\mathrm{Pf}}(k) = e_f(k) \bullet K_{P\_\mathrm{SC}} \quad \text{where} \quad K_{P\_\mathrm{SC}} = K_P \frac{E^{\mathrm{MAX}}}{U^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIp_Eq7**

where K_{P_sc} is the proportional gain parameter considering input/output scaling.

Analogically, scaling the integral term of equation GFLIB_ControllerPIp_Eq4 results in:

$$u_{\mathrm{If}}(k) = u_{\mathrm{If}}(k-1) + K_{I\_\mathrm{SC}} \bullet e_f(k) + K_{I\_\mathrm{SC}} \bullet e_f(k-1) \quad \text{where} \quad K_{I\_\mathrm{SC}} = \frac{K_I T_S}{2} \bullet \frac{E^{\mathrm{MAX}}}{U^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIp_Eq8**

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:

$$u_f(k) = e_f(k) \bullet K_{P\_SC} + u_{If}(k-1) + K_{I\_SC} \bullet e_f(k) + K_{I\_SC} \bullet e_f(k-1)$$

Equation **GFLIB_ControllerPIp_Eq9**

The problem is however, that either of the gain parameters K_{P_sc}, K_{I_sc} can be out of the [-1, 1) range, hence cannot be directly interpreted as fractional values. To overcome this, it is necessary to scale these gain parameters using the shift values as follows:

$$f32\text{PropGain} = K_{P\_sc} \bullet 2^{-s16\text{PropGainShift}}$$

Equation **GFLIB_ControllerPIp_Eq10**

and

$$f32\text{IntegGain} = K_{I\_sc} \bullet 2^{-s16\text{IntegGainShift}}$$

Equation **GFLIB_ControllerPIp_Eq11**

where

- f32PropGain - is the scaled value of proportional gain [-1, 1)
- s16PropGainShift - is the scaling shift for proportional gain [-31,31]
- f32IntegGain - is the scaled value of integral gain [-1, 1)
- s16IntegGainShift - is the scaling shift for integral gain [-31,31]

Note

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PI_P_DEFAULT_F32 macro. As the GFLIB_ControllerPIp also contains the integration part, the output result is encumbered by cumulative error. To enumerate the computation error in one calculation cycle, the internal integration accumulator is not used for testing purposes and is replaced by output from the previous calculation step of the reference model.

5.18.3.5 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)
{
    // input error = 0.25
    f32InErr  = FRAC32 (0.25);

    // controller parameters
    trMyPI.f32PropGain          = FRAC32 (0.01);
    trMyPI.f32IntegGain         = FRAC32 (0.02);
    trMyPI.s16PropGainShift     = 1;
    trMyPI.s16IntegGainShift    = 1;
    trMyPI.f32IntegPartK_1      = 0;

    // output should be 0x01EB851E
    f32Output = GFLIB_ControllerPIp_F32 (f32InErr, &trMyPI);

    // clearing of internal states
    trMyPI.f32IntegPartK_1 = (tFrac32)0;
    trMyPI.f32InK_1 = (tFrac32)0;

    // output should be 0x01EB851E
    f32Output = GFLIB_ControllerPIp (f32InErr, &trMyPI, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // clearing of internal states
    trMyPI.f32IntegPartK_1 = (tFrac32)0;
    trMyPI.f32InK_1 = (tFrac32)0;

    // output should be 0x01EB851E
    f32Output = GFLIB_ControllerPIp (f32InErr, &trMyPI);
}
```

5.18.4 Function GFLIB_ControllerPIp_F16

5.18.4.1 Declaration

```
tFrac16 GFLIB_ControllerPIp_F16(tFrac16 f16InErr, GFLIB_CONTROLLER_PI_P_T_F16 *const pParam);
```

5.18.4.2 Arguments

Table 5-65. 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.

5.18.4.3 Return

The function returns a 16-bit value in format 1.15, representing the signal to be applied to the controlled system so that the input error is forced to zero.

5.18.4.4 Variant Specifics

In order to implement the discrete equation of the controller on the fixed point arithmetic platform, the maximal values (scales) of the input and output signals have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values:

- 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:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

Equation **GFLIB_ControllerPIp_Eq5**

$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation **GFLIB_ControllerPIp_Eq6**

Applying such scaling (normalization) on the proportional term of equation GFLIB_ControllerPIp_Eq3 results in:

$$u_{Pf}(k) = e_f(k) \bullet K_{P\_SC} \quad \text{where} \quad K_{P\_SC} = K_P \frac{E^{MAX}}{U^{MAX}}$$

Equation **GFLIB_ControllerPIp_Eq7**

where K $_{P\_sc}$ is the proportional gain parameter considering input/output scaling.

Analogically, scaling the integral term of equation GFLIB_ControllerPIp_Eq4 results in:

$$u_{If}(k) = u_{If}(k-1) + K_{I\_SC} \bullet e_f(k) + K_{I\_SC} \bullet e_f(k-1) \quad \text{where} \quad K_{I\_SC} = \frac{K_I T_S}{2} \bullet \frac{E^{MAX}}{U^{MAX}}$$

Equation **GFLIB_ControllerPIp_Eq8**

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:

$$u_f(k) = e_f(k) \bullet K_{P\_SC} + u_{If}(k-1) + K_{I\_SC} \bullet e_f(k) + K_{I\_SC} \bullet e_f(k-1)$$

Equation **GFLIB_ControllerPIp_Eq9**

The problem is however, that either of the gain parameters K $_{P\_sc}$, K $_{I\_sc}$ can be out of the [-1, 1) range, hence cannot be directly interpreted as fractional values. To overcome this, it is necessary to scale these gain parameters using the shift values as follows:

$$f16\text{PropGain} = K_{P\_sc} \bullet 2^{-s16\text{PropGainShift}}$$

Equation **GFLIB_ControllerPIp_Eq10**

and

$$f16\text{IntegGain} = K_{I\_sc} \bullet 2^{-s16\text{IntegGainShift}}$$

Equation **GFLIB_ControllerPIp_Eq11**

where

- f16PropGain - is the scaled value of proportional gain [-1, 1)
- s16PropGainShift - is the scaling shift for proportional gain [-15, 15)
- f16IntegGain - is the scaled value of integral gain [-1, 1)
- s16IntegGainShift - is the scaling shift for integral gain [-15, 15)

Note

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PI_P_DEFAULT_F16 macro. As the GFLIB_ControllerPIp also contains the integration part, the output result is encumbered by cumulative error. To enumerate the computation error in one calculation cycle, the internal integration accumulator is not used for testing purposes and is replaced by output from the previous calculation step of the reference model.

5.18.4.5 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)
{
    // input error = 0.25
    f16InErr  = FRAC16 (0.25);

    // controller parameters
    trMyPI.f16PropGain          = FRAC16 (0.01);
    trMyPI.f16IntegGain         = FRAC16 (0.02);
    trMyPI.s16PropGainShift     = 1;
    trMyPI.s16IntegGainShift    = 1;
    trMyPI.f32IntegPartK_1      = 0;

    // output should be 0x01EB
    f16Output = GFLIB_ControllerPIp_F16 (f16InErr, &trMyPI);

    // clearing of internal states
    trMyPI.f32IntegPartK_1 = (tFrac32)0;
    trMyPI.f16InK_1 = (tFrac16)0;

    // output should be 0x01EB
    f16Output = GFLIB_ControllerPIp (f16InErr, &trMyPI, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // clearing of internal states
    trMyPI.f32IntegPartK_1 = (tFrac32)0;
    trMyPI.f16InK_1 = (tFrac16)0;

    // output should be 0x01EB
    f16Output = GFLIB_ControllerPIp (f16InErr, &trMyPI);
}
```

5.18.5 Function GFLIB_ControllerPIp_FLT

5.18.5.1 Declaration

```
tFloat GFLIB_ControllerPIp_FLT(tFloat fltInErr, GFLIB_CONTROLLER_PI_P_T_FLT *const pParam);
```

5.18.5.2 Arguments

Table 5-66. 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.

5.18.5.3 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.

5.18.5.4 Variant Specifics

The sum of the scaled proportional and integral terms gives a complete equation of the controller:

$$u_f(k) \;= e_f(k) \bullet K_{P\_SC} + u_{If}(k-1) + K_{I\_SC} \bullet e_f(k) + K_{I\_SC} \bullet e_f(k-1)$$

Equation **GFLIB_ControllerPIp_Eq9**

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation, zero divide).

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PI_P_DEFAULT_FLT macro. As the GFLIB_ControllerPIp also contains the integration part, the output result is affected by a cumulative error. To enumerate the computation error in one calculation cycle, the internal integration accumulator is not used for testing purposes and is replaced by output from the previous calculation step of the reference model.

5.18.5.5 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)
{
    // input error = 0.25
    fltInErr  = (tFloat)(0.25);

    // controller parameters
    trMyPI.fltPropGain        = (tFloat)(0.04);
    trMyPI.fltIntegGain       = (tFloat)(0.02);
    trMyPI.fltIntegPartK_1    = 0;

    // output should be 1.5e-2
    fltOutput = GFLIB_ControllerPIp_FLT (fltInErr, &trMyPI);

    // clearing of internal states
    trMyPI.fltIntegPartK_1 = (tFloat)0;
    trMyPI.fltInK_1 = (tFloat)0;

    // output should be 1.5e-2
    fltOutput = GFLIB_ControllerPIp (fltInErr, &trMyPI, FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // clearing of internal states
    trMyPI.fltIntegPartK_1 = (tFloat)0;
    trMyPI.fltInK_1 = (tFloat)0;

    // output should be 1.5e-2
    fltOutput = GFLIB_ControllerPIp (fltInErr, &trMyPI);
}
```

5.19 Function GFLIB_ControllerPIpAW

The function calculates the parallel form of the Proportional-Integral (PI) controller with implemented integral anti-windup functionality.

5.19.1 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:

$$u(t) = e(t) \bullet K_P + K_I \int_0^t e(t) \mathrm{dt}$$

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 transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = K_P + K_I \frac{1}{s}$$

Equation **GFLIB_ControllerPIpAW_Eq2**

The proportional part of equation GFLIB_ControllerPIpAW_Eq2 is transformed into the discrete time domain simply as:

$$u_P(k) = K_P \bullet e(k)$$

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:

$$u_I(k) = u_I(k-1) + e(k) \bullet \frac{K_I T_s}{2} + e(k-1)\frac{K_I T_s}{2}$$

Equation **GFLIB_ControllerPIpAW_Eq4**

where T_s [sec] is the sampling time.

Note

The input pointer must contain a valid address otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.19.2 Re-entrancy

The function is re-entrant.

5.19.3 Function GFLIB_ControllerPIpAW_F32

5.19.3.1 Declaration

```
tFrac32 GFLIB_ControllerPIpAW_F32(tFrac32 f32InErr, GFLIB_CONTROLLER_PIAW_P_T_F32 *const
pParam);
```

5.19.3.2 Arguments

Table 5-67. 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.

5.19.3.3 Return

The function returns a 32-bit value in format 1.31, representing the signal to be applied to the controlled system so that the input error is forced to zero.

5.19.3.4 Variant Specifics

In order to implement the discrete equation of the controller on the fixed point arithmetic platform, the maximal values (scales) of input and output signals have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values:

- 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:

$$e_f(k) = \frac{e(k)}{E^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIpAW_Eq5**

$$u_f(k) = \frac{u(k)}{U^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIpAW_Eq6**

Applying such scaling (normalization) on the proportional term of equation GFLIB_ControllerPIpAW_Eq3 results in:

$$u_{\mathrm{Pf}}(k) = e_f(k) \bullet K_{P\_\mathrm{SC}} \quad \text{where} \quad K_{P\_\mathrm{SC}} = K_P \frac{E^{\mathrm{MAX}}}{U^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIpAW_Eq7**

where K_{P_sc} is the proportional gain parameter considering input/output scaling.

Analogically, scaling the integral term of equation GFLIB_ControllerPIpAW_Eq4 results in:

$$u_{\mathrm{If}}(k) = u_{\mathrm{If}}(k-1) + K_{I\_\mathrm{SC}} \bullet e_f(k) + K_{I\_\mathrm{SC}} \bullet e_f(k-1) \quad \text{where} \quad K_{I\_\mathrm{SC}} = \frac{K_I T_S}{2} \bullet \frac{E^{\mathrm{MAX}}}{U^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIpAW_Eq8**

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. The problem is however, that either of the gain parameters K $_{P\_sc}$, K $_{I\_sc}$ can be out of the [-1, 1) range, hence can not be directly interpreted as fractional values. To overcome this, it is necessary to scale these gain parameters using the shift values as follows:

$$f32\text{PropGain} = K_{P\_sc} \bullet 2^{-s16\text{PropGainShift}}$$

Equation **GFLIB_ControllerPIpAW_Eq9**

$$f32\text{IntegGain} = K_{I\_sc} \bullet 2^{-s16\text{IntegGainShift}}$$

Equation **GFLIB_ControllerPIpAW_Eq10**

where

- f16PropGain - is the scaled value of proportional gain [-1, 1)
- s16PropGainShift - is the scaling shift for proportional gain [-31, 31)
- f16IntegGain - is the scaled value of integral gain [-1, 1)
- 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:

$$u(k) = e_f \bullet K_{P\_sc} + u_{\text{If}}(k-1) + K_{I\_sc} \bullet e_f(k) + K_{I\_sc} \bullet e_f(k-1)$$

Equation **GFLIB_ControllerPIpAW_Eq11**

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.

$$u(k) = \begin{cases} f32\text{UpperLimit} & \Rightarrow \quad u_f(k) \ge f32\text{UpperLimit} \\ u_f(k) & \Rightarrow \quad f32\text{LowerLimit} < u_f(k) < f32\text{UpperLimit} \\ f32\text{LowerLimit} & \Rightarrow \quad u_f(k) \le f32\text{LowerLimit} \end{cases}$$

Equation **GFLIB_ControllerPIpAW_Eq12**

The bounds are described by a limitation element equation GFLIB_ControllerPIpAW_Eq12. 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.

Note

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32 macro. As the GFLIB_ControllerPIp also contains the integration part, the output result is encumbered by cumulative error. To enumerate the computation error in one calculation cycle, the internal integration accumulator is not used for testing purposes and is replaced by output from the previous calculation step of the reference model.

5.19.3.5 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)
{
    // input error = 0.25
    f32InErr  = FRAC32 (0.25);

    // controller parameters
    trMyPI.f32PropGain         = FRAC32 (0.01);
    trMyPI.f32IntegGain        = FRAC32 (0.02);
    trMyPI.s16PropGainShift    = 1;
    trMyPI.s16IntegGainShift   = 1;
    trMyPI.f32IntegPartK_1     = FRAC32 (0);
    trMyPI.f32UpperLimit       = FRAC32 (1.0);
    trMyPI.f32LowerLimit       = FRAC32 (-1.0);

    // output should be 0x01EB851E
    f32Output = GFLIB_ControllerPIpAW_F32 (f32InErr, &trMyPI);

    // clearing of internal states
    trMyPI.f32IntegPartK_1 = (tFrac32)0;
    trMyPI.f32InK_1 = (tFrac32)0;

    // output should be 0x01EB851E
```

```
        f32Output = GFLIB_ControllerPIpAW (f32InErr, &trMyPI, F32);

        // ###############################################################
        // Available only if 32-bit fractional implementation selected
        // as default
        // ###############################################################

        // clearing of internal states
        trMyPI.f32IntegPartK_1 = (tFrac32)0;
        trMyPI.f32InK_1 = (tFrac32)0;

        // output should be 0x01EB851E
        f32Output = GFLIB_ControllerPIpAW (f32InErr, &trMyPI);
    }
```

5.19.4 Function GFLIB_ControllerPIpAW_F16

5.19.4.1 Declaration

```
tFrac16 GFLIB_ControllerPIpAW_F16(tFrac16 f16InErr, GFLIB_CONTROLLER_PIAW_P_T_F16 *const pParam);
```

5.19.4.2 Arguments

Table 5-68. 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.

5.19.4.3 Return

The function returns a 16-bit value in format 1.15, representing the signal to be applied to the controlled system so that the input error is forced to zero.

5.19.4.4 Variant Specifics

In order to implement the discrete equation of the controller on the fixed point arithmetic platform, the maximal values (scales) of input and output signals have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values:

- 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:

$$e_f(k) = \frac{e(k)}{E^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIpAW_Eq5**

$$u_f(k) = \frac{u(k)}{U^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIpAW_Eq6**

Applying such scaling (normalization) on the proportional term of equation GFLIB_ControllerPIpAW_Eq3 results in:

$$u_{\mathrm{Pf}}(k) = e_f(k) \bullet K_{P\_\mathrm{SC}} \quad \text{where} \quad K_{P\_\mathrm{SC}} = K_P \frac{E^{\mathrm{MAX}}}{U^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIpAW_Eq7**

where K_{P_sc} is the proportional gain parameter considering input/output scaling.

Analogically, scaling the integral term of equation GFLIB_ControllerPIpAW_Eq4 results in:

$$u_{\mathrm{If}}(k) = u_{\mathrm{If}}(k-1) + K_{I\_\mathrm{SC}} \bullet e_f(k) + K_{I\_\mathrm{SC}} \bullet e_f(k-1) \quad \text{where} \quad K_{I\_\mathrm{SC}} = \frac{K_I T_S}{2} \bullet \frac{E^{\mathrm{MAX}}}{U^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIpAW_Eq8**

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. The problem is however, that either of the gain parameters K_{P_sc}, K_{I_sc} can be out of the [-1, 1) range, hence can not be directly interpreted as fractional values. To overcome this, it is necessary to scale these gain parameters using the shift values as follows:

$$f16\text{PropGain} = K_{P\_sc} \bullet 2^{-s16\text{PropGainShift}}$$

Equation **GFLIB_ControllerPIpAW_Eq9**

$$f16\text{IntegGain} = K_{I\_sc} \bullet 2^{-s16\text{IntegGainShift}}$$

Equation **GFLIB_ControllerPIpAW_Eq10**

where

- f16PropGain - is the scaled value of proportional gain [-1, 1)
- s16PropGainShift - is the scaling shift for proportional gain [-31, 31)
- f16IntegGain - is the scaled value of integral gain [-1, 1)
- 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:

$$u(k) = e_f \bullet K_{P\_sc} + u_{\text{If}}(k-1) + K_{I\_sc} \bullet e_f(k) + K_{I\_sc} \bullet e_f(k-1)$$

Equation **GFLIB_ControllerPIpAW_Eq11**

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.

$$u(k) = \begin{cases} f16\text{UpperLimit} & \Rightarrow \quad u_f(k) \geq f16\text{UpperLimit} \\ u_f(k) & \Rightarrow \quad f16\text{LowerLimit} < u_f(k) < f16\text{UpperLimit} \\ f16\text{LowerLimit} & \Rightarrow \quad u_f(k) \leq f16\text{LowerLimit} \end{cases}$$

Equation **GFLIB_ControllerPIpAW_Eq12**

The bounds are described by a limitation element equation GFLIB_ControllerPIpAW_Eq12. 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.

Note

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16 macro. To effectively reach the target precision the internal calculation is done in 32-bit fractional arithmetic and internal accumulator is 32-bit wide. As the GFLIB_ControllerPIp also contains the integration part, the output result is encumbered by cumulative error. To enumerate the computation error in one calculation cycle, the internal integration accumulator is not used for testing purposes and is replaced by output from the previous calculation step of the reference model.

5.19.4.5 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)
{
    // input error = 0.25
    f16InErr  = FRAC16 (0.25);

    // controller parameters
    trMyPI.f16PropGain        = FRAC16 (0.01);
    trMyPI.f16IntegGain       = FRAC16 (0.02);
    trMyPI.s16PropGainShift   = 1;
    trMyPI.s16IntegGainShift  = 1;
    trMyPI.f32IntegPartK_1    = FRAC32 (0);
    trMyPI.f16UpperLimit      = FRAC16 (1.0);
    trMyPI.f16LowerLimit      = FRAC16 (-1.0);

    // output should be 0x01EB
    f16Output = GFLIB_ControllerPIpAW_F16 (f16InErr, &trMyPI);

    // clearing of internal states
    trMyPI.f32IntegPartK_1 = (tFrac32)0;
    trMyPI.f16InK_1 = (tFrac16)0;

    // output should be 0x01EB
```

```
        f16Output = GFLIB_ControllerPIpAW (f16InErr, &trMyPI, F16);

        // ##############################################################
        // Available only if 16-bit fractional implementation selected
        // as default
        // ##############################################################

        // clearing of internal states
        trMyPI.f32IntegPartK_1 = (tFrac32)0;
        trMyPI.f16InK_1 = (tFrac16)0;

        // output should be 0x01EB
        f16Output = GFLIB_ControllerPIpAW (f16InErr, &trMyPI);
    }
```

5.19.5 Function GFLIB_ControllerPIpAW_FLT

5.19.5.1 Declaration

```
tFloat GFLIB_ControllerPIpAW_FLT(tFloat fltInErr, GFLIB_CONTROLLER_PIAW_P_T_FLT *const
pParam);
```

5.19.5.2 Arguments

Table 5-69. 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.
GFLIB_CONTROLLER_PIAW_P_T_FLT *const	pParam	**input, output**	Pointer to the controller parameters structure.

5.19.5.3 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.

5.19.5.4 Variant Specifics

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.

$$u(k) = \begin{cases} \text{fltUpperLimit} & \Rightarrow & u_f(k) \geq \text{fltUpperLimit} \\ u_f(k) & \Rightarrow & \text{fltLowerLimit} < u_f(k) < \text{fltUpperLimit} \\ \text{fltLowerLimit} & \Rightarrow & u_f(k) \leq \text{fltLowerLimit} \end{cases}$$

Equation **GFLIB_ControllerPIpAW_Eq12**

The sum of the scaled proportional and integral terms gives a complete equation of the controller:

$$u(k) = e_f(k) \bullet K_P + u_I(k-1) + e(k)\frac{K_I T_s}{2} + e(k-1)\frac{K_I T_s}{2}$$

Equation **GFLIB_ControllerPIpAW_Eq6**

The bounds are described by a limitation element equation GFLIB_ControllerPIpAW_Eq6. 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 clipped to its bounds and the wind-up occurrence of the accumulator portion is avoided by saturating the actual sum.

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT macro. As the GFLIB_ControllerPIp also contains the integration part, the output result is encumbered by cumulative error. To enumerate the computation error in one calculation cycle, the internal integration accumulator is not used for testing purposes and is replaced by output from the previous calculation step of the reference model.

5.19.5.5 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)
{
    // input error = 0.25
    fltInErr  = (tFloat)(0.25);

    // controller parameters
    trMyPI.fltPropGain         = (tFloat)(0.04);
    trMyPI.fltIntegGain        = (tFloat)(0.02);
    trMyPI.fltIntegPartK_1     = 0;
    trMyPI.fltUpperLimit       = (tFloat)(1.0);
    trMyPI.fltLowerLimit       = (tFloat)(-1.0);

    // output should be 1.5e-2
    fltOutput = GFLIB_ControllerPIpAW_FLT (fltInErr, &trMyPI);

    // clearing of internal states
    trMyPI.fltIntegPartK_1 = (tFloat)0;
    trMyPI.fltInK_1 = (tFloat)0;

    // output should be 1.5e-2
    fltOutput = GFLIB_ControllerPIpAW (fltInErr, &trMyPI, FLT);

    // ##############################################################
    // Available only if single precision floating point implementation
    // selected as default
    // ##############################################################

    // clearing of internal states
    trMyPI.fltIntegPartK_1 = (tFloat)0;
    trMyPI.fltInK_1 = (tFloat)0;

    // output should be 1.5e-2
    fltOutput = GFLIB_ControllerPIpAW (fltInErr, &trMyPI);
}
```

5.20 Function GFLIB_ControllerPIr

This function calculates a standard recurrent form of the Proportional-Integral controller, without integral anti-windup.

5.20.1 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:

$$u(t) = e(t) \bullet K_P + K_I \int_0^t e(t) \mathrm{dt}$$

Equation **GFLIB_ControllerPIr_Eq1**

The transfer function for this kind of PI controller, in a continuous time domain, is described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{s \bullet K_P + K_I}{s}$$

Equation **GFLIB_ControllerPIr_Eq2**

Transforming equation GFLIB_ControllerPIr_Eq2 into a discrete time domain leads to the following equation:

$$u(k) = u(k-1) + e(k) \bullet \mathrm{CC1} + e(k-1) \bullet \mathrm{CC2}$$

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 5-70.

Table 5-70. Calculation of coefficients CC1 and CC2 using various discretization methods

	Trapezoidal	Backward Rect.	Forward Rect.
CC1=	$K_p + K_I T_S/2$	$K_p + K_I T_S$	K_p
CC2=	$-K_p + K_I T_S/2$	$-K_p$	$-K_p + K_I T_S$

Note

The input pointer must contain a valid address otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.20.2 Re-entrancy

The function is re-entrant.

5.20.3 Function GFLIB_ControllerPIr_F32

5.20.3.1 Declaration

```
tFrac32 GFLIB_ControllerPIr_F32(tFrac32 f32InErr, GFLIB_CONTROLLER_PI_R_T_F32 *const pParam);
```

5.20.3.2 Arguments

Table 5-71. 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.

5.20.3.3 Return

The function returns a 32-bit value in fractional format 1.31, representing the signal to be applied to the controlled system so that the input error is forced to zero.

5.20.3.4 Variant Specifics

In order to implement the discrete equation of the controller GFLIB_ControllerPIr_Eq3 on the fixed point arithmetic platform, the maximal values (scales) of the input and output signals

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values [-1, 1).

Then the fractional representation [-1, 1) of both input and output signals is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

Equation **GFLIB_ControllerPIr_Eq4**

$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation **GFLIB_ControllerPIr_Eq5**

The resulting controller discrete time domain equation in fixed point fractional representation is therefore given as:

$$u_f(k) \bullet U^{\mathrm{MAX}} = u_f(k-1) \bullet U^{\mathrm{MAX}} + e_f(k) \bullet E^{\mathrm{MAX}} \bullet \mathrm{CC1} + e_f(k-1) \bullet E^{\mathrm{MAX}} \bullet \mathrm{CC2}$$

Equation **GFLIB_ControllerPIr_Eq6**

which can be rearranged into the following form:

$$u_f(k) = u_f(k-1) + e_f(k) \bullet \mathrm{CC1}_f + e_f(k-1) \bullet \mathrm{CC2}_f$$

Equation **GFLIB_ControllerPIr_Eq7**

where

$$\mathrm{CC1}_f = \mathrm{CC1}\frac{E^{\mathrm{MAX}}}{U^{\mathrm{MAX}}} \quad \mathrm{CC2}_f = \mathrm{CC2}\frac{E^{\mathrm{MAX}}}{U^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIr_Eq8**

are the controller coefficients adapted according to the input and output scale values. In order to implement both coefficients as fractional numbers, both $CC1_f$ and $CC2_f$ must reside in the fractional range [-1, 1). However, depending on values CC1, CC2, E^{MAX}, U^{MAX}, the calculation of $CC1_f$ and $CC2_f$ may result in values outside this fractional range. Therefore, a scaling of $CC1_f$, $CC2_f$ is introduced as follows:

$$f32\mathrm{CC1sc} = \mathrm{CC1}_f \bullet 2^{-u16\mathrm{NShift}}$$

Equation **GFLIB_ControllerPIr_Eq9**

$$f32\mathrm{CC2sc} = \mathrm{CC2}_f \bullet 2^{-u16\mathrm{NShift}}$$

Equation **GFLIB_ControllerPIr_Eq10**

The introduced scaling shift u16NShift is chosen such that both coefficients f32CC1sc, f32CC2sc reside in the range [-1, 1). To simplify the implementation, this scaling shift is chosen to be a power of 2, so the final scaling is a simple shift operation. Moreover, the scaling shift cannot be a negative number, so the operation of scaling is always to scale numbers with an absolute value larger than 1 down to fit in the range [-1, 1).

$$u16Nshift = \max\left(\mathrm{ceil}\left(\frac{\log\left(\mathrm{abs}\left(CC1_f\right)\right)}{\log(2)}\right), \mathrm{ceil}\left(\frac{\log\left(\mathrm{abs}\left(CC2_f\right)\right)}{\log(2)}\right)\right)$$

Equation **GFLIB_ControllerPIr_Eq11**

The final, scaled, fractional equation of a recurrent PI controller on a 32-bit fixed point platform is therefore implemented as follows:

$$u_f(k) \bullet \left(2^{-u16NShift}\right) = u_f(k-1) \bullet \left(2^{-u16NShift}\right) + e_f(k) \bullet f32CC1sc + e_f(k-1) \bullet f32CC2sc$$

Equation **GFLIB_ControllerPIr_Eq12**

where:

- u $_f$(k) - fractional representation [-1, 1) of the controller output
- e $_f$(k) - fractional representation [-1, 1) of the controller input (error)
- f32CC1sc - fractional representation [-1, 1) of the 1st controller coefficient
- f32CC2sc - fractional representation [-1, 1) of the 2nd controller coefficient
- u16NShift - in range [0,31] - is chosen such that both coefficients f32CC1sc and f32CC2sc are in the range [-1, 1)

Note

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PI_R_DEFAULT_F32 macro.

5.20.3.5 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)
{
    // input error = 0.25
    f32InErr  = FRAC32 (0.25);

    // controller parameters
    trMyPI.f32CC1sc   = FRAC32 (0.01);
    trMyPI.f32CC2sc   = FRAC32 (0.02);
    trMyPI.u16NShift = 1;

    // output should be 0x00A3D70A
    f32Output = GFLIB_ControllerPIr_F32 (f32InErr,&trMyPI);

    // clearing of internal states
```

```
        trMyPI.f32Acc = (tFrac32)0;
        trMyPI.f32InErrK1 = (tFrac32)0;

        // output should be 0x00A3D70A
        f32Output = GFLIB_ControllerPIr (f32InErr,&trMyPI, F32);

        // ##############################################################
        // Available only if 32-bit fractional implementation selected
        // as default
        // ##############################################################

        // clearing of internal states
        trMyPI.f32Acc = (tFrac32)0;
        trMyPI.f32InErrK1 = (tFrac32)0;

        // output should be 0x00A3D70A
        f32Output = GFLIB_ControllerPIr (f32InErr,&trMyPI);
    }
```

5.20.4 Function GFLIB_ControllerPIr_F16

5.20.4.1 Declaration

```
tFrac16 GFLIB_ControllerPIr_F16(tFrac16 f16InErr, GFLIB_CONTROLLER_PI_R_T_F16 *const pParam);
```

5.20.4.2 Arguments

Table 5-72. 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.

5.20.4.3 Return

The function returns a 16-bit value in fractional format 1.15, representing the signal to be applied to the controlled system so that the input error is forced to zero.

5.20.4.4 Variant Specifics

In order to implement the discrete equation of the controller GFLIB_ControllerPIr_Eq3 on the fixed point arithmetic platform, the maximal values (scales) of the input and output signals

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values [-1, 1).

Then the fractional representation [-1, 1) of both input and output signals is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

Equation **GFLIB_ControllerPIr_Eq4**

$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation **GFLIB_ControllerPIr_Eq5**

The resulting controller discrete time domain equation in fixed point fractional representation is therefore given as:

$$u_f(k) \bullet U^{MAX} = u_f(k-1) \bullet U^{MAX} + e_f(k) \bullet E^{MAX} \bullet CC1 + e_f(k-1) \bullet E^{MAX} \bullet CC2$$

Equation **GFLIB_ControllerPIr_Eq6**

which can be rearranged into the following form:

$$u_f(k) = u_f(k-1) + e_f(k) \bullet CC1_f + e_f(k-1) \bullet CC2_f$$

Equation **GFLIB_ControllerPIr_Eq7**

where

$$CC1_f = CC1\frac{E^{MAX}}{U^{MAX}} \quad CC2_f = CC2\frac{E^{MAX}}{U^{MAX}}$$

Equation **GFLIB_ControllerPIr_Eq8**

are the controller coefficients adapted according to the input and output scale values. In order to implement both coefficients as fractional numbers, both $CC1_f$ and $CC2_f$ must reside in the fractional range [-1, 1). However, depending on values CC1, CC2, E^{MAX}, U^{MAX}, the calculation of $CC1_f$ and $CC2_f$ may result in values outside this fractional range. Therefore, a scaling of $CC1_f$, $CC2_f$ is introduced as follows:

$$f16CC1sc = CC1_f \bullet 2^{-u16NShift}$$

Equation **GFLIB_ControllerPIr_Eq9**

$$f16CC2sc = CC2_f \bullet 2^{-u16NShift}$$

Equation **GFLIB_ControllerPIr_Eq10**

The introduced scaling shift u16NShift is chosen such that both coefficients f16CC1sc, f16CC2sc reside in the range [-1, 1). To simplify the implementation, this scaling shift is chosen to be a power of 2, so the final scaling is a simple shift operation. Moreover, the scaling shift cannot be a negative number, so the operation of scaling is always to scale numbers with an absolute value larger than 1 down to fit in the range [-1, 1).

$$u16Nshift = \max\left(\mathrm{ceil}\left(\frac{\log\left(\mathrm{abs}\left(CC1_f\right)\right)}{\log(2)}\right), \mathrm{ceil}\left(\frac{\log\left(\mathrm{abs}\left(CC2_f\right)\right)}{\log(2)}\right)\right)$$

Equation **GFLIB_ControllerPIr_Eq11**

The final, scaled, fractional equation of a recurrent PI controller on a 16-bit fixed point platform is therefore implemented as follows:

$$u_f(k) \bullet \left(2^{-u16NShift}\right) = u_f(k-1) \bullet \left(2^{-u16NShift}\right) + e_f(k) \bullet f16CC1sc + e_f(k-1) \bullet f16CC2sc$$

Equation **GFLIB_ControllerPIr_Eq12**

where:

- $u_f(k)$ - fractional representation [-1, 1) of the controller output
- $e_f(k)$ - fractional representation [-1, 1) of the controller input (error)
- f16CC1sc - fractional representation [-1, 1) of the 1st controller coefficient
- f16CC2sc - fractional representation [-1, 1) of the 2nd controller coefficient
- u16NShift - in range [0,15] - is chosen such that both coefficients f16CC1sc and f16CC2sc are in the range [-1, 1)

Note

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PI_R_DEFAULT_F16 macro.

5.20.4.5 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)
{
    // input error = 0.25
    f16InErr  = FRAC16 (0.25);

    // controller parameters
    trMyPI.f16CC1sc   = FRAC16 (0.01);
    trMyPI.f16CC2sc   = FRAC16 (0.02);
    trMyPI.u16NShift = 1;

    // output should be 0x00A3
    f16Output = GFLIB_ControllerPIr_F16 (f16InErr,&trMyPI);

    // clearing of internal states
    trMyPI.f32Acc = (tFrac32)0;
    trMyPI.f16InErrK1 = (tFrac16)0;

    // output should be 0x00A3
    f16Output = GFLIB_ControllerPIr (f16InErr,&trMyPI, F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // clearing of internal states
    trMyPI.f32Acc = (tFrac32)0;
    trMyPI.f16InErrK1 = (tFrac16)0;

    // output should be 0x00A3
    f16Output = GFLIB_ControllerPIr (f16InErr,&trMyPI);
}
```

5.20.5 Function GFLIB_ControllerPIr_FLT

5.20.5.1 Declaration

```
tFloat GFLIB_ControllerPIr_FLT(tFloat fltInErr, GFLIB_CONTROLLER_PI_R_T_FLT *const pParam);
```

5.20.5.2 Arguments

Table 5-73. 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.

5.20.5.3 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.

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PI_R_DEFAULT_FLT macro. As the PI controller also contains the integration part, the output result is encumbered by cumulative error. To enumerate the computation error in one calculation cycle, the internal accumulator is not used for testing purposes and is replaced by output from the previous calculation step of the reference model.

5.20.5.4 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)
{
    // input error = 0.25
    fltInErr  = (tFloat)(0.25);

    // controller parameters
    trMyPI.fltCC1sc          = (tFloat)(0.01);
    trMyPI.fltCC2sc          = (tFloat)(0.01);
```

```
    // output should be 2.5e-3
    fltOutput = GFLIB_ControllerPIr_FLT (fltInErr,&trMyPI);

    // clearing of internal states
    trMyPI.fltAcc = (tFloat)0;
    trMyPI.fltInErrK1 = (tFloat)0;

    // output should be 2.5e-3
    fltOutput = GFLIB_ControllerPIr (fltInErr,&trMyPI, FLT);

    // ##############################################################
    // Available only if single precision floating point implementation
    // selected as default
    // ##############################################################

    // clearing of internal states
    trMyPI.fltAcc = (tFloat)0;
    trMyPI.fltInErrK1 = (tFloat)0;

    // output should be 2.5e-3
    fltOutput = GFLIB_ControllerPIr (fltInErr,&trMyPI);
}
```

5.21 Function GFLIB_ControllerPIrAW

This function calculates a standard recurrent form of the Proportional-Integral controller, with integral anti-windup.

5.21.1 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:

$$u(t) = e(t) \bullet K_P + K_I \int_0^t e(t) \mathrm{dt}$$

Equation **GFLIB_ControllerPIrAW_Eq1**

The transfer function for this kind of PI controller, in a continuous time domain is described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{s \bullet K_P + K_I}{s}$$

Equation **GFLIB_ControllerPIrAW_Eq2**

Transforming equation GFLIB_ControllerPIrAW_Eq2 into a discrete time domain leads to the following equation:

$$u(k) = u(k-1) + e(k) \bullet CC1 + e(k-1) \bullet CC2$$

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 5-74.

Table 5-74. Calculation of coefficients CC1 and CC2 using various discretization methods

	Trapezoidal	Backward Rect.	Forward Rect.
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 an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.21.2 Re-entrancy

The function is re-entrant.

5.21.3 Function GFLIB_ControllerPIrAW_F32

5.21.3.1 Declaration

```
tFrac32 GFLIB_ControllerPIrAW_F32(tFrac32 f32InErr, GFLIB_CONTROLLER_PIAW_R_T_F32 *const pParam);
```

5.21.3.2 Arguments

Table 5-75. 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).

Table continues on the next page...

Table 5-75. GFLIB_ControllerPIrAW_F32 arguments (continued)

Type	Name	Direction	Description
GFLIB_CONTROLLER_PIAW_R_T_F32 *const	pParam	**input, output**	Pointer to the controller parameters structure.

5.21.3.3 Return

The function returns a 32-bit value in fractional format 1.31, representing the signal to be applied to the controlled system so that the input error is forced to zero.

5.21.3.4 Variant Specifics

In order to implement the discrete equation of the controller eq3_GFLIB_ControllerPIrAW_F32 on the fixed point arithmetic platform, the maximal values (scales) of input and output signals

- E ^{MAX} - maximal value of the controller input error signal
- U ^{MAX} - maximal value of the controller output signal

have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values [-1, 1).

Then the fractional representation [-1, 1) of both input and output signals is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIrAW_Eq4**

$$u_f(k) = \frac{e(k)}{U^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIrAW_Eq5**

The resulting controller discrete time domain equation in fixed point fractional representation is therefore given as:

$$u_f(k) \bullet U^{\mathrm{MAX}} = u_f(k-1) \bullet U^{\mathrm{MAX}} + e_f \bullet E^{\mathrm{MAX}} \bullet \mathrm{CC1} + e_f(k-1) \bullet E^{\mathrm{MAX}} \bullet \mathrm{CC2}$$

Equation **GFLIB_ControllerPIrAW_Eq6**

which can be rearranged into the following form:

$$u_f(k) = u_f(k-1) + e_f(k)\mathrm{CC1}_f + e_f(k-1) \bullet \mathrm{CC2}_f$$

Equation **GFLIB_ControllerPIrAW_Eq7**

where

$$\mathrm{CC1}_f = \mathrm{CC1} \bullet \frac{E^{\mathrm{MAX}}}{U^{\mathrm{MAX}}} \quad \mathrm{CC2}_f = \mathrm{CC2} \bullet \frac{E^{\mathrm{MAX}}}{U^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIrAW_Eq8**

are the controller coefficients adapted according to the input and output scale values. In order to implement both coefficients as fractional numbers, both $CC1_f$ and $CC2_f$ must reside in the fractional range [-1, 1). However, depending on values CC1, CC2, E^{MAX}, U^{MAX}, the calculation of $CC1_f$ and $CC2_f$ may result in values outside this fractional range. Therefore, a scaling of $CC1_f$, $CC2_f$ is introduced as follows:

$$f32\mathrm{CC1}_{\mathrm{SC}} = \mathrm{CC1}_f \bullet 2^{-u16\mathrm{NShift}}$$

Equation **GFLIB_ControllerPIrAW_Eq9**

$$f32\mathrm{CC2}_{\mathrm{SC}} = \mathrm{CC2}_f \bullet 2^{-u16\mathrm{NShift}}$$

Equation **GFLIB_ControllerPIrAW_Eq10**

The introduced scaling shift u16NShift is chosen such that both coefficients f32CC1sc,f32CC2sc reside in the range [-1, 1). To simplify the implementation, this scaling shift is chosen to be a power of 2, so the final scaling is a simple shift operation. Moreover, the scaling shift cannot be a negative number, so the operation of scaling is always to scale numbers with an absolute value larger than 1 down to fit in the range [-1, 1).

$$u16\mathrm{NShift} = \max\left(\mathrm{ceil}\left(\frac{\log(\mathrm{abs}(\mathrm{CC1}_f))}{\log(2)}\right), \quad \mathrm{ceil}\frac{\log(\mathrm{abs}(\mathrm{CC2}_f))}{\log(2)}\right)$$

Equation **GFLIB_ControllerPIrAW_Eq11**

The final, scaled, fractional equation of the recurrent PI controller on a 32-bit fixed point platform is therefore implemented as follows:

$$u_f(k) \bullet \left(2^{-u16NShift}\right) = u_f(k-1) \bullet \left(2^{-u16NShift}\right) + e_f(k) \bullet f32CC1_{SC} + e_f(k-1) \bullet f32CC2_{SC}$$

Equation **GFLIB_ControllerPIrAW_Eq12**

where:

- u $_f$(k) - fractional representation [-1, 1) of the controller output
- e $_f$(k) - fractional representation [-1, 1) of the controller input (error)
- f32CC1sc - fractional representation [-1, 1) of the 1st controller coefficient
- f32CC2sc - fractional representation [-1, 1) of the 2nd controller coefficient
- u16NShift - in range [0,31] - is chosen such that both coefficients f32CC1sc and f32CC2sc are in the range [-1, 1)

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.

$$u_f\left(k\right) = \begin{cases} f32UpperLimit & \Rightarrow \quad u_f(k) \geq f32UpperLimit \\ u_f(k) & \Rightarrow \quad f32LowerLimit < u_f(k) < f32UpperLimit \\ f32LowerLimit & \Rightarrow \quad u_f(k) \leq f32LowerLimit \end{cases}$$

Equation **GFLIB_ControllerPIrAW_Eq13**

The bounds are described by a limitation element equation GFLIB_ControllerPIrAW_Eq13. 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 clipped to its bounds and the wind-up occurrence of the accumulator is avoided by saturating the output sum.

Note

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32 macro. As the GFLIB_ControllerPIrAW_F32 also contains the internal accumulator, the output result is encumbered by cumulative error. To enumerate the computation error in one calculation cycle, the internal accumulator is not used for testing purposes and is replaced by output from the previous calculation step of the reference model. The anti-windup mechanism is implemented based on the output limitation.

5.21.3.5 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)
{
    // input error = 0.25
    f32InErr  = FRAC32 (0.25);

    // 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);

    // output should be 0x00A3D70A
    f32Output = GFLIB_ControllerPIrAW_F32 (f32InErr, &trMyPI);

    // clearing of internal states
    trMyPI.f32Acc = (tFrac32)0;
    trMyPI.f32InErrK1 = (tFrac32)0;

    // output should be 0x00A3D70A
    f32Output = GFLIB_ControllerPIrAW (f32InErr, &trMyPI, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // clearing of internal states
    trMyPI.f32Acc = (tFrac32)0;
    trMyPI.f32InErrK1 = (tFrac32)0;

    // output should be 0x00A3D70A
    f32Output = GFLIB_ControllerPIrAW (f32InErr, &trMyPI);
}
```

5.21.4 Function GFLIB_ControllerPIrAW_F16

5.21.4.1 Declaration

```
tFrac16 GFLIB_ControllerPIrAW_F16(tFrac16 f16InErr, GFLIB_CONTROLLER_PIAW_R_T_F16 *const
pParam);
```

5.21.4.2 Arguments

Table 5-76. 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.

5.21.4.3 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.

5.21.4.4 Variant Specifics

In order to implement the discrete equation of the controller eq3_GFLIB_ControllerPIrAW_F16 on the fixed point arithmetic platform, the maximal values (scales) of input and output signals

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values [-1, 1).

Then the fractional representation [-1, 1) of both input and output signals is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

Equation **GFLIB_ControllerPIrAW_Eq4**

$$u_f(k) = \frac{e(k)}{U^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIrAW_Eq5**

The resulting controller discrete time domain equation in fixed point fractional representation is therefore given as:

$$u_f(k) \bullet U^{\mathrm{MAX}} = u_f(k-1) \bullet U^{\mathrm{MAX}} + e_f \bullet E^{\mathrm{MAX}} \bullet \mathrm{CC1} + e_f(k-1) \bullet E^{\mathrm{MAX}} \bullet \mathrm{CC2}$$

Equation **GFLIB_ControllerPIrAW_Eq6**

which can be rearranged into the following form:

$$u_f(k) = u_f(k-1) + e_f(k)\mathrm{CC1}_f + e_f(k-1) \bullet \mathrm{CC2}_f$$

Equation **GFLIB_ControllerPIrAW_Eq7**

where

$$\mathrm{CC1}_f = \mathrm{CC1} \bullet \frac{E^{\mathrm{MAX}}}{U^{\mathrm{MAX}}} \quad \mathrm{CC2}_f = \mathrm{CC2} \bullet \frac{E^{\mathrm{MAX}}}{U^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIrAW_Eq8**

are the controller coefficients adapted according to the input and output scale values. In order to implement both coefficients as fractional numbers, both $CC1_f$ and $CC2_f$ must reside in the fractional range [-1, 1). However, depending on values CC1, CC2, E^{MAX}, U^{MAX}, the calculation of $CC1_f$ and $CC2_f$ may result in values outside this fractional range. Therefore, a scaling of $CC1_f$, $CC2_f$ is introduced as follows:

$$f16\mathrm{CC1}_{\mathrm{SC}} = \mathrm{CC1}_f \bullet 2^{-u16\mathrm{NShift}}$$

Equation **GFLIB_ControllerPIrAW_Eq9**

$$f16\mathrm{CC2}_{\mathrm{SC}} = \mathrm{CC2}_f \bullet 2^{-u16\mathrm{NShift}}$$

Equation **GFLIB_ControllerPIrAW_Eq10**

The introduced scaling shift u16NShift is chosen such that both coefficients f16CC1sc,f16CC2sc reside in the range [-1, 1). To simplify the implementation, this scaling shift is chosen to be a power of 2, so the final scaling is a simple shift operation. Moreover, the scaling shift cannot be a negative number, so the operation of scaling is always to scale numbers with an absolute value larger than 1 down to fit in the range [-1, 1).

$$u16\text{NShift} = \max\left(\text{ceil}\left(\frac{\log\left(\text{abs}\left(\text{CC1}_f\right)\right)}{\log(2)}\right), \quad \text{ceil}\frac{\log\left(\text{abs}\left(\text{CC2}_f\right)\right)}{\log(2)}\right)$$

Equation **GFLIB_ControllerPIrAW_Eq11**

The final, scaled, fractional equation of the recurrent PI controller on a 16-bit fixed point platform is therefore implemented as follows:

$$u_f(k) \bullet \left(2^{-u16\text{NShift}}\right) = u_f(k-1) \bullet \left(2^{-u16\text{NShift}}\right) + e_f(k) \bullet f16\text{CC1}_{\text{SC}} + e_f(k-1) \bullet f16\text{CC2}_{\text{SC}}$$

Equation **GFLIB_ControllerPIrAW_Eq12**

where:

- u $_f$(k) - fractional representation [-1, 1) of the controller output
- e $_f$(k) - fractional representation [-1, 1) of the controller input (error)
- f16CC1sc - fractional representation [-1, 1) of the 1st controller coefficient
- f16CC2sc - fractional representation [-1, 1) of the 2nd controller coefficient
- u16NShift - in range [0,15] - is chosen such that both coefficients f16CC1sc and f16CC2sc are in the range [-1, 1)

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.

$$u_f\left(k\right) = \begin{cases} f16\text{UpperLimit} & \Rightarrow \quad u_f(k) \geq f16\text{UpperLimit} \\ u_f(k) & \Rightarrow \quad f16\text{LowerLimit} < u_f(k) < f16\text{UpperLimit} \\ f16\text{LowerLimit} & \Rightarrow \quad u_f(k) \leq f16\text{LowerLimit} \end{cases}$$

Equation **GFLIB_ControllerPIrAW_Eq13**

The bounds are described by a limitation element equation GFLIB_ControllerPIrAW_Eq13. 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 clipped to its bounds and the wind-up occurrence of the accumulator is avoided by saturating the output sum.

Note

Due to effectivity reasons this function is implemented using intrinsic functions and inline assembly and is therefore not ANSI-C compliant. Note that some compilers do not support the enhanced features.

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16 macro. As the GFLIB_ControllerPIrAW_F16 also contains the internal accumulator, the output result is encumbered by cumulative error. To enumerate the computation error in one calculation cycle, the internal accumulator is not used for testing purposes and is replaced by output from the previous calculation step of the reference model. To eliminate the calculation error the GFLIB_ControllerPIrAW_F16 uses the 32-bit wide internal accumulator. The anti-windup mechanism is implemented based on the output limitation.

5.21.4.5 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)
{
    // input error = 0.25
    f16InErr  = FRAC16 (0.25);

    // 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);

    // output should be 0x00A3
    f16Output = GFLIB_ControllerPIrAW_F16 (f16InErr, &trMyPI);

    // clearing of internal states
    trMyPI.f32Acc = (tFrac32)0;
    trMyPI.f16InErrK1 = (tFrac16)0;
```

```
        // output should be 0x00A3
        f16Output = GFLIB_ControllerPIrAW (f16InErr, &trMyPI, F16);

        // ##############################################################
        // Available only if 16-bit fractional implementation selected
        // as default
        // ##############################################################

        // clearing of internal states
        trMyPI.f32Acc = (tFrac32)0;
        trMyPI.f16InErrK1 = (tFrac16)0;

        // output should be 0x00A3
        f16Output = GFLIB_ControllerPIrAW (f16InErr, &trMyPI);
    }
```

5.21.5 Function GFLIB_ControllerPIrAW_FLT

5.21.5.1 Declaration

```
tFloat GFLIB_ControllerPIrAW_FLT(tFloat fltInErr, GFLIB_CONTROLLER_PIAW_R_T_FLT *const
pParam);
```

5.21.5.2 Arguments

Table 5-77. GFLIB_ControllerPIrAW_FLT arguments

Type	Name	Direction	Description
tFloat	fltInErr	**input**	Input error signal to the controller in single precision floating point data format.
GFLIB_CONTROLLER_PIAW_R_T_FLT *const	pParam	**input, output**	Pointer to the controller parameters structure.

5.21.5.3 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.

5.21.5.4 Variant Specifics

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.

$$u_f(k) = \begin{cases} \text{fltUpperLimit} & \Rightarrow \quad u_f(k) \geq \text{fltUpperLimit} \\ u_f(k) & \Rightarrow \quad \text{fltLowerLimit} < u_f(k) < \text{fltUpperLimit} \\ \text{fltLowerLimit} & \Rightarrow \quad u_f(k) \leq \text{fltLowerLimit} \end{cases}$$

Equation **GFLIB_ControllerPIrAW_Eq13**

The bounds are described by a limitation element equation GFLIB_ControllerPIrAW_Eq4. 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 clipped to its bounds and the wind-up occurrence of the accumulator is avoided by saturating the output sum.

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT macro. As the PI controller also contains the integration part, the output result is encumbered by cumulative error. To enumerate the computation error in one calculation cycle, the internal accumulator is not used for testing purposes and is replaced by output from the previous calculation step of the reference model. The anti-windup mechanism is implemented based on the output limitation.

5.21.5.5 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)
{
    // input error = 0.25
    fltInErr  = (tFloat)(0.25);

    // controller parameters
    trMyPI.fltCC1sc            = (tFloat)(0.01);
    trMyPI.fltCC2sc            = (tFloat)(0.01);
    trMyPI.fltUpperLimit       = (tFloat)(1.0);
    trMyPI.fltLowerLimit       = (tFloat)(-1.0);

    // output should be 2.5e-3
    fltOutput = GFLIB_ControllerPIrAW_FLT (fltInErr, &trMyPI);

    // clearing of internal states
    trMyPI.fltAcc = (tFloat)0;
    trMyPI.fltInErrK1 = (tFloat)0;

    // output should be 2.5e-3
    fltOutput = GFLIB_ControllerPIrAW (fltInErr, &trMyPI, FLT);

    // ##############################################################
    // Available only if single precision floating point implementation
    // selected as default
    // ##############################################################

    // clearing of internal states
    trMyPI.fltAcc = (tFloat)0;
    trMyPI.fltInErrK1 = (tFloat)0;

    // output should be 2.5e-3
    fltOutput = GFLIB_ControllerPIrAW (fltInErr, &trMyPI);
}
```

5.22 Function GFLIB_Cos

5.22.1 Description

The function GFLIB_Cos calculates the trigonometric cosine function using a polynomial approximation of the sine function

$$\cos(x) = \sin\left(\frac{\pi}{2} + x\right)$$

Equation **GFLIB_Cos_Eq1**

Note

The input pointer must contain a valid address otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.22.2 Re-entrancy

The function is re-entrant.

5.22.3 Function GFLIB_Cos_F32

5.22.3.1 Declaration

```
tFrac32 GFLIB_Cos_F32(tFrac32 f32In, const GFLIB_COS_T_F32 *const pParam);
```

5.22.3.2 Arguments

Table 5-78. GFLIB_Cos_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument is a 32-bit number that contains an angle in radians between [- π, π) normalized between [-1, 1).
const GFLIB_COS_T_F32 *const	pParam	**input**	Pointer to an array of Taylor coefficients.

5.22.3.3 Return

The function returns the cos of the input argument as a fixed point 32-bit number, normalized between [-1, 1).

5.22.3.4 Variant Specifics

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.

Figure 5-22 depicts a floating point *cosine* function generated from Matlab and the approximated value of the *cosine* function obtained from GFLIB_Cos_F32, plus their difference.

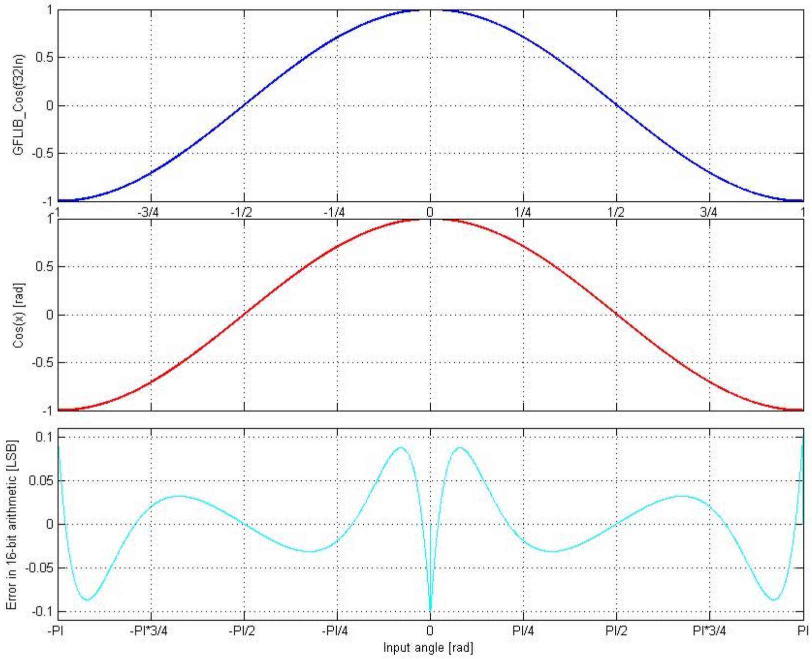

Figure 5-22. cos(x) vs. GFLIB_Cos(f32In)

Note

The function call is slightly different from common approach in the library set. The function can be called in three different ways:

- With implementation postfix (i.e. GFLIB_Cos_F32(f32In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_COS_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Cos(f32In, &pParam, F32), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_COS_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Cos(f32In, &pParam), where &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_COS_DEFAULT_F32 approximation coefficients are used.

5.22.3.5 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);
}
```

5.22.4 Function GFLIB_Cos_F16

5.22.4.1 Declaration

```
tFrac16 GFLIB_Cos_F16(tFrac16 f16In, const GFLIB_COS_T_F16 *const pParam);
```

5.22.4.2 Arguments

Table 5-79. GFLIB_Cos_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument is a 16-bit number that contains an angle in radians between [- π, π) normalized between [-1, 1).
const GFLIB_COS_T_F16 *const	pParam	**input**	Pointer to an array of Taylor coefficients.

5.22.4.3 Return

The function returns the cos of the input argument as a fixed point 16-bit number, normalized between [-1, 1).

5.22.4.4 Variant Specifics

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.

Figure 5-23 depicts a floating point *cosine* function generated from Matlab and the approximated value of the *cosine* function obtained from GFLIB_Cos_F16, plus their difference.

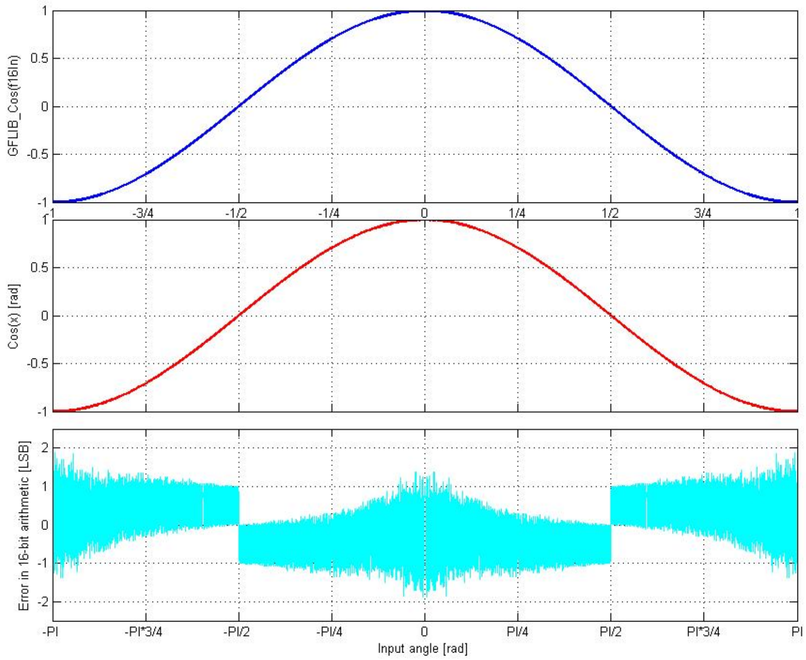

Figure 5-23. cos(x) vs. GFLIB_Cos(f16In)

Note

The function call is slightly different from common approach in the library set. The function can be called in three different ways:

- With implementation postfix (i.e. GFLIB_Cos_F16(f16In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_COS_DEFAULT_F16 symbol. The &pParam parameter is mandatory.

- With additional implementation parameter (i.e. GFLIB_Cos(f16In, &pParam, F16), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_COS_DEFAULT_F16 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Cos(f16In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_COS_DEFAULT_F16 approximation coefficients are used.

5.22.4.5 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);
}
```

5.22.5 Function GFLIB_Cos_FLT

5.22.5.1 Declaration

```
tFloat GFLIB_Cos_FLT(tFloat fltIn, const GFLIB_COS_T_FLT *const pParam);
```

5.22.5.2 Arguments

Table 5-80. 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 between [- π, π].
const GFLIB_COS_T_FLT *const	pParam	**input**	Pointer to an array of approximation coefficients.

5.22.5.3 Return

The function returns the cosine of the input argument as a single precision floating point number.

5.22.5.4 Variant Specifics

The function uses a 7th order minimax polynomial approximation; the default polynomial coefficients are provided in the GFLIB_COS_DEFAULT_FLT structure.

Figure 5-24 depicts a floating point *cosine* function generated from Matlab and the approximated value of the *cosine* function obtained from GFLIB_Cos_FLT, plus their difference.

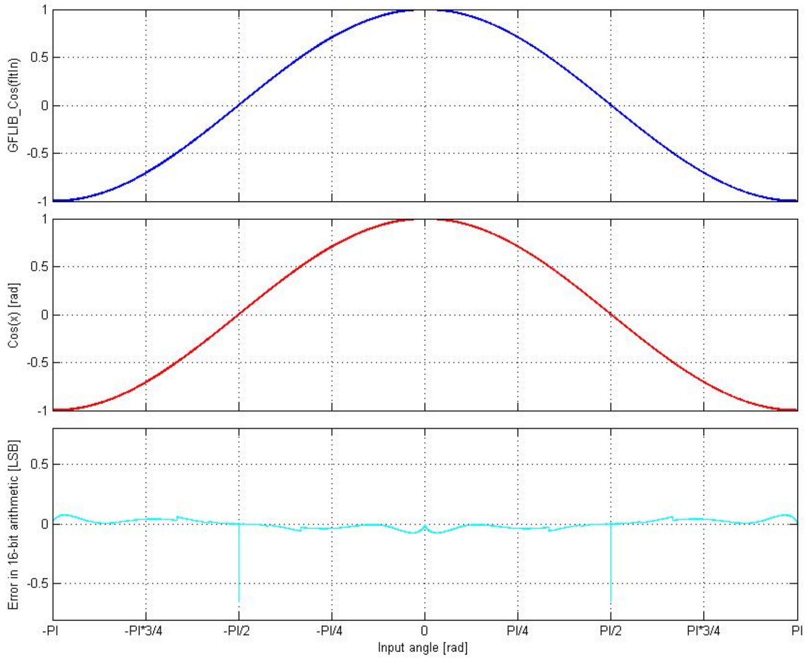

Figure 5-24. cos(x) vs. GFLIB_Cos(fltIn)

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

The function call is slightly different from common approach in the library set. The function can be called in three different ways:

- With implementation postfix (i.e. GFLIB_Cos_FLT(fltIn, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_COS_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Cos(fltIn, &pParam, FLT), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_COS_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Cos(fltIn, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default

GFLIB_COS_DEFAULT_FLT approximation coefficients are used.

5.22.5.5 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);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output should be 0.70710678
  fltOutput = GFLIB_Cos (fltAngle);
}
```

5.23 Function GFLIB_Hyst

5.23.1 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).

$$OutState(k) = \begin{cases} OutValOn, & In \geq HystOn \\ OutValOff, & In \leq HystOff \\ OutState(k-1), & otherwise \end{cases}$$

Equation **GFLIB_Hyst_Eq1**

A graphical description of GFLIB_Hyst functionality is shown in Figure 5-25.

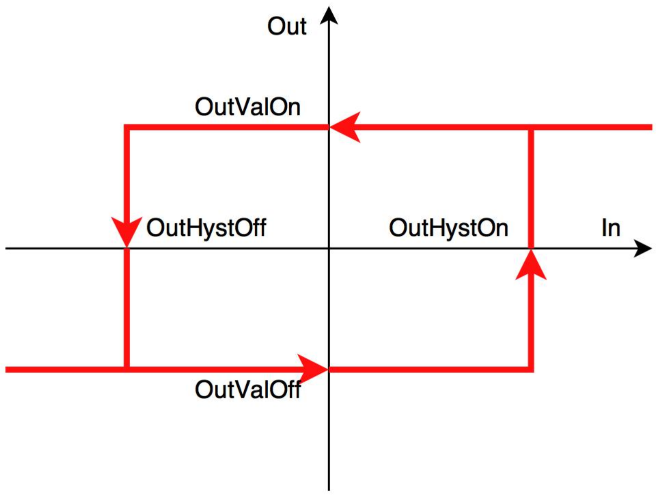

Figure 5-25. Hysteresis function

Note

The input pointer must contain a valid address otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.23.2 Re-entrancy

The function is re-entrant.

5.23.3 Function GFLIB_Hyst_F32

5.23.3.1 Declaration

```
tFrac32 GFLIB_Hyst_F32(tFrac32 f32In, GFLIB_HYST_T_F32 *const pParam);
```

5.23.3.2 Arguments

Table 5-81. 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).

5.23.3.3 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.

5.23.3.4 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);

    // 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);
}
```

5.23.4 Function GFLIB_Hyst_F16

5.23.4.1 Declaration

```
tFrac16 GFLIB_Hyst_F16(tFrac16 f16In, GFLIB_HYST_T_F16 *const pParam);
```

5.23.4.2 Arguments

Table 5-82. 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).

5.23.4.3 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.

5.23.4.4 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;
GFLIB_HYST_T_F16 f16trMyHyst = GFLIB_HYST_DEFAULT_F16;

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);
}
```

5.23.5 Function GFLIB_Hyst_FLT

5.23.5.1 Declaration

```
tFloat GFLIB_Hyst_FLT(tFloat fltIn, GFLIB_HYST_T_FLT *const pParam);
```

5.23.5.2 Arguments

Table 5-83. 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.

5.23.5.3 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 (floating-point invalid operation).

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.

5.23.5.4 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);
    }
```

5.24 Function GFLIB_IntegratorTR

The function calculates a discrete implementation of the integrator (sum), discretized using a trapezoidal (Bilinear) transformation with overflow on range boundaries.

5.24.1 Description

The function GFLIB_IntegratorTR implements a discrete integrator using trapezoidal (Bilinear) transformation.

The continuous time domain representation of the integrator is defined as:

$$u(t) = \int_0^t e(t)\,\mathrm{dt}$$

Equation **GFLIB_IntegratorTR_Eq1**

The transfer function for this integrator, in a continuous time domain, is described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{1}{s}$$

Equation **GFLIB_IntegratorTR_Eq2**

Transforming equation GFLIB_IntegratorTR_Eq2 into a digital time domain using Bilinear transformation, leads to the following transfer function:

$$\mathbb{Z}\ \{\mathrm{H(s)}\}= \frac{U(z)}{E(z)}\ \frac{T_s+T_s z^{-1}}{2-2z^{-1}}$$

Equation **GFLIB_IntegratorTR_Eq3**

where T_s is the sampling period of the system. The discrete implementation of the digital transfer function GFLIB_IntegratorTR_Eq3 is as follows:

$$u(k) = u(k-1) + e(k)\frac{T_s}{2} + e(k-1)\frac{T_s}{2}$$

Equation **GFLIB_IntegratorTR_Eq4**

Note

The input pointer must contain a valid address otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.24.2 Re-entrancy

The function is re-entrant.

5.24.3 Function GFLIB_IntegratorTR_F32

5.24.3.1 Declaration

```
tFrac32 GFLIB_IntegratorTR_F32(tFrac32 f32In, GFLIB_INTEGRATOR_TR_T_F32 *const pParam);
```

5.24.3.2 Arguments

Table 5-84. 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.

5.24.3.3 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.

5.24.3.4 Variant Specifics

Considering fractional math implementation, the integrator input and output maximal values (scales) must be known. Then the discrete implementation is given as follows:

$$u(k) = u(k-1) + e(k) \bullet \frac{T_s}{2} \bullet \frac{E_{\mathrm{MAX}}}{U_{\mathrm{MAX}}} + e(k-1) \bullet \frac{T_s}{2} \bullet \frac{E_{\mathrm{MAX}}}{U_{\mathrm{MAX}}}$$

Equation **GFLIB_IntegratorTR_Eq5**

where E_{MAX} is the input scale and U_{MAX} is the output scale. Then integrator constant C1 is defined as:

$$C1_f = \frac{T_s}{2} \bullet \frac{E_{\mathrm{MAX}}}{U_{\mathrm{MAX}}}$$

Equation **GFLIB_IntegratorTR_Eq6**

In order to implement the discrete form integrator as in GFLIB_IntegratorTR_Eq5 on a fixed point platform, the value of $C1_f$ coefficient must reside in the fractional range [-1,1). Therefore, scaling must be introduced as follows:

$$f32C1 = C1_f \bullet 2^{-u16\mathrm{NShift}}$$

Equation **GFLIB_IntegratorTR_Eq7**

The introduced scaling is chosen such that coefficient f32C1 fits into fractional range [-1,1). To simplify the implementation, this scaling is chosen to be a power of 2, so the final scaling is a simple shift operation using the u16NShift variable. Hence, the shift is calculated as:

$$u16\mathrm{NShift} = \mathrm{ceil}\left(\frac{\log(C1_f)}{\log(2)}\right)$$

Equation **GFLIB_IntegratorTR_Eq8**

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_Eq5 overflows.

5.24.3.5 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);
    trMyIntegrator.u16NShift = (tU16)0;

    // input value = 0.5
    f32In = FRAC32 (0.5);

    // output should be 0x0051EB85
    f32Out = GFLIB_IntegratorTR_F32 (f32In, &trMyIntegrator);

    // clearing of the internal states
    trMyIntegrator.f32State = (tFrac32)0;
    trMyIntegrator.f32InK1  = (tFrac32)0;

    // output should be 0x0051EB85
    f32Out = GFLIB_IntegratorTR (f32In, &trMyIntegrator, F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // clearing of the internal states
    trMyIntegrator.f32State = (tFrac32)0;
    trMyIntegrator.f32InK1  = (tFrac32)0;

    // output should be 0x0051EB85
    f32Out = GFLIB_IntegratorTR (f32In, &trMyIntegrator);
}
```

5.24.4 Function GFLIB_IntegratorTR_F16

5.24.4.1 Declaration

```
tFrac16 GFLIB_IntegratorTR_F16(tFrac16 f16In, GFLIB_INTEGRATOR_TR_T_F16 *const pParam);
```

5.24.4.2 Arguments

Table 5-85. 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.

5.24.4.3 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.

5.24.4.4 Variant Specifics

Considering fractional math implementation, the integrator input and output maximal values (scales) must be known. Then the discrete implementation is given as follows:

$$u(k) = u(k-1) + e(k) \bullet \frac{T_s}{2} \bullet \frac{E_{\mathrm{MAX}}}{U_{\mathrm{MAX}}} + e(k-1) \bullet \frac{T_s}{2} \bullet \frac{E_{\mathrm{MAX}}}{U_{\mathrm{MAX}}}$$

Equation **GFLIB_IntegratorTR_Eq5**

where E $_{MAX}$ is the input scale and U $_{MAX}$ is the output scale. Then integrator constant C1 is defined as:

$$C1_f = \frac{T_s}{2} \bullet \frac{E_{\mathrm{MAX}}}{U_{\mathrm{MAX}}}$$

Equation **GFLIB_IntegratorTR_Eq6**

In order to implement the discrete form integrator as in GFLIB_IntegratorTR_Eq5 on a fixed point platform, the value of C1 $_f$ coefficient must reside in the fractional range [-1,1). Therefore, scaling must be introduced as follows:

$$f16C1 = C1_f \bullet 2^{-u16\mathrm{NShift}}$$

Equation **GFLIB_IntegratorTR_Eq7**

The introduced scaling is chosen such that coefficient f16C1 fits into fractional range [-1,1). To simplify the implementation, this scaling is chosen to be a power of 2, so the final scaling is a simple shift operation using the u16NShift variable. Hence, the shift is calculated as:

$$u16\mathrm{NShift} = \mathrm{ceil}\left(\frac{\log(C1_f)}{\log(2)}\right)$$

Equation **GFLIB_IntegratorTR_Eq8**

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_Eq5 overflows.

5.24.4.5 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);
    trMyIntegrator.u16NShift = (tU16)0;

    // input value = 0.5
    f16In = FRAC16 (0.5);

    // output should be 0x0051
    f16Out = GFLIB_IntegratorTR_F16 (f16In, &trMyIntegrator);

    // clearing of the internal states
    trMyIntegrator.f32State = (tFrac32)0;
    trMyIntegrator.f16InK1  = (tFrac16)0;

    // output should be 0x0051
    f16Out = GFLIB_IntegratorTR (f16In, &trMyIntegrator, F16);
```

```
        // ###############################################################
        // Available only if 16-bit fractional implementation selected
        // as default
        // ###############################################################

        // clearing of the internal states
        trMyIntegrator.f32State = (tFrac32)0;
        trMyIntegrator.f16InK1  = (tFrac16)0;

        // output should be 0x0051
        f16Out = GFLIB_IntegratorTR (f16In, &trMyIntegrator);
    }
```

5.24.5 Function GFLIB_IntegratorTR_FLT

5.24.5.1 Declaration

```
tFloat GFLIB_IntegratorTR_FLT(tFloat fltIn, GFLIB_INTEGRATOR_TR_T_FLT *const pParam);
```

5.24.5.2 Arguments

Table 5-86. 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.

5.24.5.3 Return

The function returns a single precision floating point value, which represents the actual integrated value of the input signal.

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

All parameters and states used by the function can be reset during declaration using the GFLIB_INTEGRATOR_TR_DEFAULT_FLT macro.

5.24.5.4 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);

    // input value = 0.5
    fltIn = (tFloat)0.5;

    // output should be 2.5e-3
    fltOut = GFLIB_IntegratorTR_FLT (fltIn, &trMyIntegrator);

    // clearing of the internal states
    trMyIntegrator.fltState = (tFloat)0;
    trMyIntegrator.fltInK1  = (tFloat)0;

    // output should be 2.5e-3
    fltOut = GFLIB_IntegratorTR (fltIn, &trMyIntegrator, FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // clearing of the internal states
    trMyIntegrator.fltState = (tFloat)0;
    trMyIntegrator.fltInK1  = (tFloat)0;

    // output should be 2.5e-3
    fltOut = GFLIB_IntegratorTR (fltIn, &trMyIntegrator);
}
```

5.25 Function GFLIB_Limit

This function tests whether the input value is within the upper and lower limits.

5.25.1 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 an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.25.2 Re-entrancy

The function is re-entrant.

5.25.3 Function GFLIB_Limit_F32

5.25.3.1 Declaration

```
tFrac32 GFLIB_Limit_F32(tFrac32 f32In, const GFLIB_LIMIT_T_F32 *const pParam);
```

5.25.3.2 Arguments

Table 5-87. 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.

5.25.3.3 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.

5.25.3.4 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);
}
```

5.25.4 Function GFLIB_Limit_F16

5.25.4.1 Declaration

```
tFrac16 GFLIB_Limit_F16(tFrac16 f16In, const GFLIB_LIMIT_T_F16 *const pParam);
```

5.25.4.2 Arguments

Table 5-88. 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.

5.25.4.3 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.

5.25.4.4 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);
}
```

5.25.5 Function GFLIB_Limit_FLT

5.25.5.1 Declaration

```
tFloat GFLIB_Limit_FLT(tFloat fltIn, const GFLIB_LIMIT_T_FLT *const pParam);
```

5.25.5.2 Arguments

Table 5-89. 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.

5.25.5.3 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 (floating-point invalid operation).

The function assumes that the upper limit fltUpperLimit is greater than the lower limit fltLowerLimit. Otherwise, the function returns an undefined value.

5.25.5.4 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);
}
```

5.26 Function GFLIB_Log10_FLT

This function calculates a base-10 logarithm of an absolute value.

5.26.1 Declaration

```
tFloat GFLIB_Log10_FLT(tFloat fltIn, const GFLIB_LOG10_T_FLT *const pParam);
```

5.26.2 Arguments

Table 5-90. 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.

5.26.3 Return

The function returns log10(abs(fltIn)) as a single precision floating point number.

5.26.4 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.

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact).

5.26.5 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);
}
```

5.27 Function GFLIB_LowerLimit

This function tests whether the input value is above the lower limit.

5.27.1 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 an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.27.2 Re-entrancy

The function is re-entrant.

5.27.3 Function GFLIB_LowerLimit_F32

5.27.3.1 Declaration

```
tFrac32 GFLIB_LowerLimit_F32(tFrac32 f32In, const GFLIB_LOWERLIMIT_T_F32 *const pParam);
```

5.27.3.2 Arguments

Table 5-91. 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.

5.27.3.3 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.

5.27.3.4 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)
    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);
}
```

5.27.4 Function GFLIB_LowerLimit_F16

5.27.4.1 Declaration

```
tFrac16 GFLIB_LowerLimit_F16(tFrac16 f16In, const GFLIB_LOWERLIMIT_T_F16 *const pParam);
```

5.27.4.2 Arguments

Table 5-92. 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.

5.27.4.3 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.

5.27.4.4 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
        // ##############################################################

        // output should be 0x6000 ~ FRAC16(0.75)
        f16Out = GFLIB_LowerLimit (f16In,&f16trMyLowerLimit);
    }
```

5.27.5 Function GFLIB_LowerLimit_FLT

5.27.5.1 Declaration

```
tFloat GFLIB_LowerLimit_FLT(tFloat fltIn, const GFLIB_LOWERLIMIT_T_FLT *const pParam);
```

5.27.5.2 Arguments

Table 5-93. 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.

5.27.5.3 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 (floating-point invalid operation).

5.27.5.4 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
    // ##############################################################

    // output should be 0.75
    fltOut = GFLIB_LowerLimit (fltIn,&flttrMyLowerLimit);
}
```

5.28 Function GFLIB_Lut1D

This function implements the one-dimensional look-up table.

5.28.1 Description

The GFLIB_Lut1D function performs one dimensional linear interpolation over a table of data. The data is assumed to represent a one dimensional function sampled at equidistant points. The following interpolation formula is used:

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}\left(x - x_1\right)$$

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
- the x is the input value provided to the function in the In argument

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 an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.28.2 Re-entrancy

The function is re-entrant.

5.28.3 Function GFLIB_Lut1D_F32

5.28.3.1 Declaration

```
tFrac32 GFLIB_Lut1D_F32(tFrac32 f32In, const GFLIB_LUT1D_T_F32 *const pParam);
```

5.28.3.2 Arguments

Table 5-94. 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.

5.28.3.3 Return

The interpolated value from the look-up table with 16-bit accuracy.

5.28.3.4 Variant Specifics

The interpolating intervals are defined in the table provided by 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 5-95. GFLIB_Lut1D example table

ordinate (y)	interpolating index	abscissa (x)
-0.5	-1	$-1*(2^{-1})$
pf32Table{rarrow} 0.0	0	$0*(2^{-1})$
0.25	1	$1*(2^{-1})$
0.5	N/A	$2*(2^{-1})$

The Table 5-95 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:

- the values of the interpolated function are 32 bits long
- the length of each interpolating interval is equal to 2^{-n}, where n is an integer in the range of 1, 2, ... 29
- the provided abscissa for interpolation is 32 bits long

The algorithm performs the following steps:

1. Compute the index representing the interval, in which the linear interpolation will be performed:

$$I = x > > s_{\text{Interval}}$$

Equation **GFLIB_Lut1D_Eq2**

where I is the interval index and the s_{Interval} the shift amount provided in the parameters structure as the member s32ShamIntvl. The operator >> represents the binary arithmetic right shift.

2. Compute the abscissa offset within an interpolating interval:

$$\Delta\text{x} = x < < s_{\text{Offset}} \, \& \, 0x\text{7ffffff}$$

Equation **GFLIB_Lut1D_Eq3**

where $\Delta\{x\}$ is the abscissa offset within an interval and the s_{Offset} is the shift amount provided in the parameters structure. The operators << and & represent, respectively, the binary left shift and the bitwise logical conjunction. It should be noted that the computation represents the extraction of some least significant bits of the x with the sign bit cleared.

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.

$$y_1 = (32-\text{bit data at address pTable}) + 4 \bullet I$$
$$y_2 = (32-\text{bit data at address pTable}) + 4 \bullet (I+1)$$
$$y = y_1 + (y_2 - y_1) \bullet \Delta x$$

Equation **GFLIB_Lut1D_Eq4**

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 address provided in the parameters structure pParam->f32Table. It should be noted that due to assumption of equidistant data points, division by the interval length is avoided.

It should be noted that the computations are performed with a 16-bit accuracy. In particular, the 16 least significant bits are ignored in all multiplications.

The shift amounts shall be provided in the parameters structure (pParam->u32ShamOffset). The address of the table with the data, the pTable, shall be defined by the parameter structure member pParam->pf32Table.

The shift amounts, the $s_{Interval}$ and s_{Offset}, can be computed with the following formulas:

$$s_{\text{Interval}} = 31 - |n|$$
$$s_{\text{Offset}} = |n|$$

Equation **GFLIB_Lut1D_Eq5**

where n is the integer defining the length of the interpolating interval in the range of -1, -2, ... -29.

The computation of the abscissa offset and the interval index can be viewed also in the following way. The input abscissa value can be divided into two parts. The first n most significant bits of the 32-bit word, after the sign bit, compose the interval index, in which interpolation is performed. The rest of the bits form the abscissa offset within the interpolating interval. This simple way to calculate the interpolating interval index and the abscissa offset is the consequence of assuming that all interpolating interval lengths equal 2^{-n}.

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, which is located behind the pParam->pf32Table 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.

5.28.3.5 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)};

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);

        // available only if 32-bit fractional implementation selected as default
        // output should be 0x66666666 ~ FRAC32(0.8)
        f32Out = GFLIB_Lut1D (f32In,&trf32MyLut1D);
    }
```

5.28.4 Function GFLIB_Lut1D_F16

5.28.4.1 Declaration

```
tFrac16 GFLIB_Lut1D_F16(tFrac16 f16In, const GFLIB_LUT1D_T_F16 *const pParam);
```

5.28.4.2 Arguments

Table 5-96. 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.

5.28.4.3 Return

The interpolated value from the look-up table.

5.28.4.4 Variant Specifics

The interpolating intervals are defined in the table provided by 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 5-97. GFLIB_Lut1D example table

ordinate (y)	interpolating index	abscissa (x)
-0.5	-1	-1*(2 $^{-1}$)
pf16Table{rarrow} 0.0	0	0*(2 $^{-1}$)
0.25	1	1*(2 $^{-1}$)
0.5	N/A	2*(2 $^{-1}$)

The Table 5-97 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.

A special algorithm is used to make the computation efficient, however, under some additional assumptions, as provided below:

- the values of the interpolated function are 16 bits long
- 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 16 bits long

The algorithm performs the following steps:

1. Compute the index representing the interval, in which the linear interpolation will be performed:

$$I = x >> s_{\mathrm{Interval}}$$

Equation **GFLIB_Lut1D_Eq2**

   where I is the interval index and the s_{Interval} the shift amount provided in the parameters structure as the member s16ShamIntvl. The operator >> represents the binary arithmetic right shift.
2. Compute the abscissa offset within an interpolating interval:

$$\Delta \mathrm{x} = x << s_{\mathrm{Offset}} \;\&\; 0x7\mathrm{fff}$$

Equation **GFLIB_Lut1D_Eq3**

   where $\Delta\{x\}$ is the abscissa offset within an interval and the s_{Offset} is the shift amount provided in the parameters structure. The operators << and & represent, respectively, the binary left shift and the bitwise logical conjunction. It should be noted that the computation represents the extraction of some least significant bits of the x with the sign bit cleared.
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.

$$y_1 = \left(16-\mathrm{bit\ \ data\ \ at\ \ address\ \ pTable}\right) + 4 \bullet I$$
$$y_2 = \left(16-\mathrm{bit\ \ data\ \ at\ \ address\ \ pTable}\right) + 4 \bullet \left(I+1\right)$$
$$y = y_1 + \left(y_2 - y_1\right) \bullet \Delta \mathrm{x}$$

Equation **GFLIB_Lut1D_Eq4**

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 address provided in the parameters structure pParam->f16Table. It should be noted that due to assumption of equidistant data points, division by the interval length is avoided.

It should be noted that the computations are performed with a 16-bit accuracy. In particular, the 16 least significant bits are ignored in all multiplications.

The shift amounts shall be provided in the parameters structure (pParam->u16ShamOffset). The address of the table with the data, the pTable, shall be defined by the parameter structure member pParam->pf16Table.

The shift amounts, the s_{Interval} and s_{Offset}, can be computed with the following formulas:

$$s_{\text{Interval}} = 15 - |n|$$
$$s_{\text{Offset}} = |n|$$

Equation **GFLIB_Lut1D_Eq5**

where n is the integer defining the length of the interpolating interval in the range of -1, -2, ... -15.

The computation of the abscissa offset and the interval index can be viewed also in the following way. The input abscissa value can be divided into two parts. The first n most significant bits of the 16-bit halfword, after the sign bit, compose the interval index, in which interpolation is performed. The rest of the bits form the abscissa offset within the interpolating interval. This simple way to calculate the interpolating interval index and the abscissa offset is the consequence of assuming that all interpolating interval lengths equal 2^{-n}.

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.

5.28.4.5 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;
    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);
}
```

5.28.5 Function GFLIB_Lut1D_FLT

5.28.5.1 Declaration

```
tFloat GFLIB_Lut1D_FLT(tFloat fltIn, const GFLIB_LUT1D_T_FLT *const pParam);
```

5.28.5.2 Arguments

Table 5-98. 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.

5.28.5.3 Return

The interpolated value from the look-up table.

5.28.5.4 Variant Specifics

The interpolating intervals are defined in the table provided by the pfltTable 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. 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:

- the values of the interpolated function are 32 bits long
- the length of each interpolating interval is equal to 2^{-n}, where n is an integer in the range of 1, 2, ... 29
- the abscissa provided for interpolation is 32 bits long

The algorithm performs the following steps:

1. Count the equidistant interpolating interval, in which the linear interpolation will be performed:

$$\mathrm{fSegments} = (\mathrm{tFloat})(1 << s32\mathrm{ShamOffset} + 1)$$

Equation **GFLIB_Lut1D_Eq2**

where fSegments is the interval length and the pParam->u32ShamOffset is the shift amount provided in the parameters structure as the member u32ShamOffset. Operator << represents the binary arithmetic left shift.

2. Compute the position of the interpolated values in the table:

$$s32\mathrm{Intvl} = (\mathrm{tS32})((\mathrm{fsegments}) \bullet (\mathrm{fltIn}))$$

Equation **GFLIB_Lut1D_Eq3**

where s32Intvl is the position of the first ordinate value at the beginning of the interpolating interval, the fSegments is the equidistant interpolating interval and the fltIn is the input value provided to the function as an argument. The casting to tS32 data type is a final operation of the equation which cuts the fractional part of the value and then indicates the position of the first ordinate value in the table.

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.

$$y_1 = \left(32 - \text{bit} \quad \text{data} \quad \text{at} \quad \text{address} \quad \text{pTable}\right) + s32\text{Intvl}$$

$$y_2 = \left(32 - \text{bit} \quad \text{data} \quad \text{at} \quad \text{address} \quad \text{pTable}\right) + \left(s32\text{Intvl} + 1\right)$$

$$\Delta\text{x} = \left(\text{tFloat}\right)\left(x \bullet 2^n\right) - \left(\text{tU32}\right)\left(x \bullet 2^n\right)$$

$$y = y_1 + \left(y_2 - y_1\right) \bullet \Delta\text{x}$$

Equation **GFLIB_Lut1D_Eq4**

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 address provided in the parameters structure pParam->fltTable and the x is the input value provided to the function as an fltIn argument. It should be noted that due to assumption of equidistant data points, division by the interval length is avoided.

The address of the table with the data, the pTable, shall be defined by the parameter structure member pParam->pfltTable.

The offset amount can be provided by the following formula:

$$s32\text{ShamOffset} = |n|$$

Equation **GFLIB_Lut1D_Eq5**

where n is the integer defining the length of the interpolating interval in the range of 1, 2, ... 29.

This simple way to calculate the interpolating abscissa offset is a consequence of assuming that all interpolating interval lengths equal 2^{-n}.

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->pfltTable. However, if it is negative, then the ordinate values are read from the memory, which is located behind the pParam->pfltTable pointer.

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

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.

5.28.5.5 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;

                    // 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);
                }
```

5.29 Function GFLIB_Lut2D

This function implements the two-dimensional look-up table.

5.29.1 Description

The GFLIB_Lut2D function performs two dimensional linear interpolation over a 2D table of data.

The following interpolation formulas are used:

$$f[x, y_1] = f[x_1, y_1] + \frac{f[x_2, y_1] - f[x_1, y_1]}{x_2 - x_1} \bullet (x - x_1)$$

Equation **GFLIB_Lut2D_Eq1**

$$f[x, y_2] = f[x_1, y_2] + \frac{f[x_2, y_2] - f[x_1, y_2]}{x_2 - x_1} \bullet (x - x_1)$$

Equation **GFLIB_Lut2D_Eq2**

$$f[x, y] = f[x, y_1] + \frac{f[x, y_2] - f[x, y_1]}{y_2 - y_1} \bullet (y - y_1)$$

Equation **GFLIB_Lut2D_Eq3**

where:

- f[x,y] is the interpolated value
- f[x,y_1] and f[x,y_2] are the ordinate values at, respectively, the beginning and the end of the final interpolating interval
- x_1, x_2, y_1 and y_2 are the area values, respectively, the interpolated area
- the x, y are the input values provided to the function in the In1 and In2 arguments

Note

The input pointer must contain a valid address and the range of the input values must fall within the range of the interpolating data table otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.29.2 Re-entrancy

The function is re-entrant.

5.29.3 Function GFLIB_Lut2D_F32

5.29.3.1 Declaration

```
tFrac32 GFLIB_Lut2D_F32(tFrac32 f32In1, tFrac32 f32In2, const GFLIB_LUT2D_T_F32 *const
pParam);
```

5.29.3.2 Arguments

Table 5-99. GFLIB_Lut2D_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In1	**input**	First input variable for which 2D interpolation is performed.
tFrac32	f32In2	**input**	Second input variable for which 2D interpolation is performed.

Table continues on the next page...

Table 5-99. GFLIB_Lut2D_F32 arguments (continued)

Type	Name	Direction	Description
const GFLIB_LUT2D_T_F32 *const	pParam	**input**	Pointer to the parameters structure with parameters of the two dimensional look-up table function.

5.29.3.3 Return

The interpolated value from the look-up table with 16-bit accuracy.

5.29.3.4 Variant Specifics

The interpolating intervals are defined in the table provided by the pf32Table member of the parameters structure. The table contains ordinate values consecutively over the whole 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 5-100. GFLIB_Lut2D example table

ordinate (f[x,y])	interpolating index	abscissa (x)	abscissa (y)
-0.5	-1	$-1*(2^{-1})$	$-1*(2^{-1})$
pf32Table{rarrow} 0.0	0	$0*(2^{-1})$	$0*(2^{-1})$
0.5	1	$1*(2^{-1})$	$1*(2^{-1})$
1.0	N/A	$2*(2^{-1})$	$2*(2^{-1})$

The Table 5-100 contains 4 interpolating points (note four rows). 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 noticed that the pf32Table pointer does not have to point to the start of the memory area with ordinate values. Therefore the interpolating index can be positive or negative or, even, does not have to have zero in its range.

Special algorithm is used to make the computation efficient, however under some additional assumptions as provided below:

- the values of the interpolated function are 32-bit long

- the length of each interpolating interval is equal to 2^{-n}, where n is an integer in the range of 1, 2, ... 29
- the provided abscissa for interpolation is 32-bit long

The algorithm performs the following steps:

1. Compute the index representing the interval, in which the bilinear interpolation will be performed in each axis:

$$I_x = x >> s_{\mathrm{IntegralX}}$$

Equation **GFLIB_Lut2D_Eq4**

$$I_y = y >> s_{\mathrm{IntegralY}}$$

Equation **GFLIB_Lut2D_Eq5**

   where I_x and I_y is the interval index and the $s_{IntervalX}$ and $s_{IntervalY}$ are the shift amounts provided in the parameters structure as a member u32ShamOffset1 or u32ShamOffset2. The operator >> represents the binary arithmetic right shift.
2. Compute the abscissas offset for both axis within an interpolating intervals:

$$\Delta\mathrm{x} = x << s_{\mathrm{Offset1}} \,\&\, 0x7\mathrm{FFFFFFF}$$

Equation **GFLIB_Lut2D_Eq6**

$$\Delta\mathrm{y} = y << s_{\mathrm{Offset2}} \,\&\, 0x7\mathrm{FFFFFFF}$$

Equation **GFLIB_Lut2D_Eq7**

   where $\Delta\{x\}$ and $\Delta\{y\}$ are the abscissas offset within an Xinterval and Yinterval. The $s_{Offset1}$ and $s_{Offset2}$ are the shift amounts provided in the parameters structure. The operators << and & represent the binary left shift and the bitwise logical conjunction. It should be noted that the computation represents extraction of some least significant bits of the x and y with the sign bit cleared.
3. Compute the interpolated value by the linear interpolation in X-axis by y_1 value between the ordinates x_1 and x_2 readed from the table at the start and the end of the computed interval in X-axis way. The computed abscissa offset is used for the linear interpolation:

$$f\left[x_1, y_1\right] = 32 - \mathrm{bit\ \ data\ \ at\ \ address\ \ pTable\ \ for}\ \ x1, y1$$

Equation **GFLIB_Lut2D_Eq8**

$$f\left[x_2, y_1\right] = 32 - \text{bit} \quad \text{data} \quad \text{at} \quad \text{address} \quad \text{pTable} \quad \text{for} \quad x2, y1$$

Equation **GFLIB_Lut2D_Eq9**

$$f\left[x, y_1\right] = f\left[x_1, y_1\right] + \frac{f\left[x_2, y_1\right] - f\left[x_1, y_1\right]}{\Delta \mathrm{x}} \bullet \left(x - x_1\right)$$

Equation **GFLIB_Lut2D_Eq10**

where f[x,y 1] is interpolated value, f[x 1,y 1] and f[x 2,y 1] are, the ordinate at the start of the interpolating interval and the ordinate at the end of the interpolating interval. The pTable} is the address provided in the parameters structure pParam->f32Table.

4. The same computation as shown above in X-axis by y 2 value. Interpolation formulas are the same as paragraph 3:

$$f\left[x_1, y_2\right] = 32 - \text{bit} \quad \text{data} \quad \text{at} \quad \text{address} \quad \text{pTable} \quad \text{for} \quad x1, y2$$

Equation **GFLIB_Lut2D_Eq11**

$$f\left[x_2, y_2\right] = 32 - \text{bit} \quad \text{data} \quad \text{at} \quad \text{address} \quad \text{pTable} \quad \text{for} \quad x2, y2$$

Equation **GFLIB_Lut2D_Eq12**

$$f\left[x, y_2\right] = f\left[x_1, y_2\right] + \frac{f\left[x_2, y_2\right] - f\left[x_1, y_2\right]}{\Delta \mathrm{x}} \bullet \left(x - x_1\right)$$

Equation **GFLIB_Lut2D_Eq13**

5. Final linear interpolation in Y-axis way Interpolation formulas are the same as paragraph 3 or 4:

$$f\left[x, y_1\right] = \text{result} \quad \text{coputed} \quad \text{in} \quad \text{paragraph} \quad 3$$

Equation **GFLIB_Lut2D_Eq14**

$$f\left[x, y_2\right] = \text{result} \quad \text{coputed} \quad \text{in} \quad \text{paragraph} \quad 4$$

Equation **GFLIB_Lut2D_Eq15**

$$f\left[x, y\right] = f\left[x, y_1\right] + \frac{f\left[x, y_2\right] - f\left[x, y_1\right]}{\Delta y} \bullet \left(y - y_1\right)$$

Equation **GFLIB_Lut2D_Eq16**

It should be noted that due to assumption of the equidistant data points, the division by the interval length is avoided.

It should be noted that the computations are performed with the 16-bit accuracy. In particular the 16 least significant bits are ignored in all multiplications.

The shift amounts shall be provided in the parameters structure (pParam->u32ShamOffset1 and pParam->u32ShamOffset2). The address of the table with the data, the pTable}, shall be defined by the parameter structure member pParam->pf32Table.

The shift amounts, the $s_{\text{Interval1,2}}$ and $s_{\text{Offset1,2}}$, can be computed with the following formulas:

$$s_{\text{Interval1,2}} = 31 - |n|$$
$$s_{\text{Offset1,2}} = |n|$$

Equation **GFLIB_Lut2D_Eq17**

where n is the integer defining the length of the interpolating interval in the range of -1, -2, ... -29.

The computation of the abscissa offset and the interval index can be viewed also in the following way. The input abscissa value can be divided into two parts. The first n most significant bits of the 32-bit word, after the sign bit, compose the interval index, in which interpolation is performed. The rest of the bits form the abscissa offset within the interpolating interval. This simple way to calculate the interpolating interval index and the abscissa offset is the consequence of assumption of all interpolating interval lengths equal 2^{-n}.

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, which is located behind the pParam->pf32Table pointer.

Note

The function performs the bilinear interpolation with 16-bit accuracy.

CAUTION

The function does not check whether the input values are within a 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.

5.29.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32Out;
GFLIB_LUT2D_T_F32 tr32tMyLut2D = GFLIB_LUT2D_DEFAULT_F32;
tFrac32 pf32Table2D[81] = {FRAC32 (0.9), FRAC32 (0.01), FRAC32 (0.02), FRAC32
(0.03), FRAC32 (0.04), FRAC32 (0.05), FRAC32 (0.06), FRAC32 (0.07), FRAC32 (0.08),
                                  FRAC32 (0.1), FRAC32 (0.11), FRAC32 (0.12), FRAC32
(0.13), FRAC32 (0.14), FRAC32 (0.15), FRAC32 (0.16), FRAC32 (0.17), FRAC32 (0.18),
                                  FRAC32 (0.2), FRAC32 (0.21), FRAC32 (0.22), FRAC32
(0.23), FRAC32 (0.24), FRAC32 (0.25), FRAC32 (0.26), FRAC32 (0.27), FRAC32 (0.28),
                                  FRAC32 (0.3), FRAC32 (0.31), FRAC32 (0.32), FRAC32
(0.33), FRAC32 (0.34), FRAC32 (0.35), FRAC32 (0.36), FRAC32 (0.37), FRAC32 (0.38),
                                  FRAC32 (0.4), FRAC32 (0.41), FRAC32 (0.42), FRAC32
(0.43), FRAC32 (0.44), FRAC32 (0.45), FRAC32 (0.46), FRAC32 (0.47), FRAC32 (0.48),
                                  FRAC32 (0.5), FRAC32 (0.51), FRAC32 (0.52), FRAC32
(0.53), FRAC32 (0.54), FRAC32 (0.55), FRAC32 (0.56), FRAC32 (0.57), FRAC32 (0.58),
                                  FRAC32 (0.6), FRAC32 (0.61), FRAC32 (0.62), FRAC32
(0.63), FRAC32 (0.64), FRAC32 (0.65), FRAC32 (0.66), FRAC32 (0.67), FRAC32 (0.68),
                                  FRAC32 (0.7), FRAC32 (0.71), FRAC32 (0.72), FRAC32
(0.73), FRAC32 (0.74), FRAC32 (0.75), FRAC32 (0.76), FRAC32 (0.77), FRAC32 (0.78),
                                  FRAC32 (0.8), FRAC32 (0.81), FRAC32 (0.82), FRAC32
(0.83), FRAC32 (0.84), FRAC32 (0.85), FRAC32 (0.86), FRAC32 (0.87), FRAC32 (0.88)};

void main(void)
{
    // ######################################################################
    // Pointer is located in the middle of the interpolating data table.
    // ######################################################################
```

```
    // setting parameters for Lut2D function
    tr32tMyLut2D.u32ShamOffset1 = (tU32)3;
    tr32tMyLut2D.u32ShamOffset2 = (tU32)3;
    tr32tMyLut2D.pf32Table = &(pf32Table2D[40]);

    // input vector
    f32In1 = FRAC32 (-0.5);
    f32In2 = FRAC32 (-0.5);

    // output should be 0x73333333 ~ FRAC32(0.9)
    f32Out = GFLIB_Lut2D_F32 (f32In1,f32In2,&tr32tMyLut2D);

    // output should be 0x73333333 ~ FRAC32(0.9)
    f32Out = GFLIB_Lut2D (f32In1,f32In2,&tr32tMyLut2D,F32);

    // available only if 32-bit fractional implementation selected as default
    // output should be 0x73333333 ~ FRAC32(0.9)
    f32Out = GFLIB_Lut2D (f32In1,f32In2,&tr32tMyLut2D);

    // ######################################################################
    // Pointer is located at the beginning of the interpolating data table.
    // ######################################################################
    // setting parameters for Lut2D function
    tr32tMyLut2D.u32ShamOffset1 = (tU32)3;
    tr32tMyLut2D.u32ShamOffset2 = (tU32)3;
    tr32tMyLut2D.pf32Table = &(pf32Table2D[0]);

    // input vector
    f32In1 = FRAC32 (0);
    f32In2 = FRAC32 (0);

    // output should be 0x73333333 ~ FRAC32(0.9)
    f32Out = GFLIB_Lut2D_F32 (f32In1,f32In2,&tr32tMyLut2D);

    // output should be 0x73333333 ~ FRAC32(0.9)
    f32Out = GFLIB_Lut2D (f32In1,f32In2,&tr32tMyLut2D,F32);

    // available only if 32-bit fractional implementation selected as default
    // output should be 0x73333333 ~ FRAC32(0.9)
    f32Out = GFLIB_Lut2D (f32In1,f32In2,&tr32tMyLut2D);

    // ######################################################################
    // Pointer is located at the end of the interpolating data table.
    // ######################################################################
    // setting parameters for Lut2D function
    tr32tMyLut2D.u32ShamOffset1 = (tU32)3;
    tr32tMyLut2D.u32ShamOffset2 = (tU32)3;
    tr32tMyLut2D.pf32Table = &(pf32Table2D[80]);

    // input vector
    f32In1 = FRAC32 (-1);
    f32In2 = FRAC32 (-1);

    // output should be 0x73333333 ~ FRAC32(0.9)
    f32Out = GFLIB_Lut2D_F32 (f32In1,f32In2,&tr32tMyLut2D);

    // output should be 0x73333333 ~ FRAC32(0.9)
    f32Out = GFLIB_Lut2D (f32In1,f32In2,&tr32tMyLut2D,F32);

    // available only if 32-bit fractional implementation selected as default
    // output should be 0x73333333 ~ FRAC32(0.9)
    f32Out = GFLIB_Lut2D (f32In1,f32In2,&tr32tMyLut2D);
}
```

5.29.4 Function GFLIB_Lut2D_F16

5.29.4.1 Declaration

```
tFrac16 GFLIB_Lut2D_F16(tFrac16 f16In1, tFrac16 f16In2, const GFLIB_LUT2D_T_F16 *const
pParam);
```

5.29.4.2 Arguments

Table 5-101. GFLIB_Lut2D_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In1	**input**	First input variable for which 2D interpolation is performed.
tFrac16	f16In2	**input**	Second input variable for which 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.

5.29.4.3 Return

The interpolated value from the look-up table.

5.29.4.4 Variant Specifics

The interpolating intervals are defined in the table provided by the pf16Table member of the parameters structure. The table contains ordinate values consecutively over the whole 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 5-102. GFLIB_Lut2D example table

ordinate (f[x,y])	interpolating index	abscissa (x)	abscissa (y)
-0.5	-1	-1*(2 $^{-1}$)	-1*(2 $^{-1}$)
pf16Table{rarrow} 0.0	0	0*(2 $^{-1}$)	0*(2 $^{-1}$)
0.5	1	1*(2 $^{-1}$)	1*(2 $^{-1}$)
1.0	N/A	2*(2 $^{-1}$)	2*(2 $^{-1}$)

The Table 5-102 contains 4 interpolating points (note four rows). 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 noticed that the pf16Table pointer does not have to point to the start of the memory area with ordinate values. Therefore the interpolating index can be positive or negative or, even, does not have to have zero in its range.

Special algorithm is used to make the computation efficient, however under some additional assumptions as provided below:

- the values of the interpolated function are 16-bit long
- 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 16-bit long

The algorithm performs the following steps:

1. Compute the index representing the interval, in which the bilinear interpolation will be performed in each axis:

$$I_x = x >> s_{\mathrm{IntegralX}}$$

Equation **GFLIB_Lut2D_Eq4**

$$I_y = y >> s_{\mathrm{IntegralY}}$$

Equation **GFLIB_Lut2D_Eq5**

where I_x and I_y is the interval index and the $s_{\mathrm{IntervalX}}$ and $s_{\mathrm{IntervalY}}$ are the shift amounts provided in the parameters structure as a member u16ShamOffset1 or u16ShamOffset2. The operator >> represents the binary arithmetic right shift.

2. Compute the abscissas offset for both axis within an interpolating intervals:

$$\Delta\mathrm{x} = x << s_{\mathrm{Offset1}} \,\&\, 0x7\mathrm{FFF}$$

Equation **GFLIB_Lut2D_Eq6**

$$\Delta\mathrm{y} = y << s_{\mathrm{Offset2}} \,\&\, 0x7\mathrm{FFF}$$

Equation **GFLIB_Lut2D_Eq7**

where Δ{x} and Δ{y} are the abscissas offset within an Xinterval and Yinterval. The s $_{Offset1}$ and s $_{Offset2}$ are the shift amounts provided in the parameters structure. The operators << and & represent the binary left shift and the bitwise logical conjunction. It should be noted that the computation represents extraction of some least significant bits of the x and y with the sign bit cleared.

3. Compute the interpolated value by the linear interpolation in X-axis by y $_1$ value between the ordinates x_1 and x $_2$ readed from the table at the start and the end of the computed interval in X-axis way. The computed abscissa offset is used for the linear interpolation:

$$f\left[x_1, y_1\right] = 16 - \text{bit data at address pTable for } x1, y1$$

Equation **GFLIB_Lut2D_Eq8**

$$f\left[x_2, y_1\right] = 16 - \text{bit data at address pTable for } x2, y1$$

Equation **GFLIB_Lut2D_Eq9**

$$f\left[x, y_1\right] = f\left[x_1, y_1\right] + \frac{f\left[x_2, y_1\right] - f\left[x_1, y_1\right]}{\Delta x} \bullet \left(x - x_1\right)$$

Equation **GFLIB_Lut2D_Eq10**

where f[x,y $_1$] is interpolated value, f[x $_1$,y $_1$] and f[x $_2$,y $_1$] are, the ordinate at the start of the interpolating interval and the ordinate at the end of the interpolating interval. The pTable} is the address provided in the parameters structure pParam->f16Table.

4. The same computation as shown above in X-axis by y $_2$ value. Interpolation formulas are the same as paragraph 3:

$$f\left[x_1, y_2\right] = 16 - \text{bit data at address pTable for } x1, y2$$

Equation **GFLIB_Lut2D_Eq11**

$$f\left[x_2, y_2\right] = 16 - \text{bit data at address pTable for } x2, y2$$

Equation **GFLIB_Lut2D_Eq12**

$$f\left[x, y_2\right] = f\left[x_1, y_2\right] + \frac{f\left[x_2, y_2\right] - f\left[x_1, y_2\right]}{\Delta x} \bullet \left(x - x_1\right)$$

Equation **GFLIB_Lut2D_Eq13**

5. Final linear interpolation in Y-axis way Interpolation formulas are the same as paragraph 3 or 4:

$$f\left[x, y_1\right] = \text{result coputed in paragraph 3}$$

Equation **GFLIB_Lut2D_Eq14**

$$f\left[x, y_2\right] = \text{result coputed in paragraph 4}$$

Equation **GFLIB_Lut2D_Eq15**

$$f\left[x, y\right] = f\left[x, y_1\right] + \frac{f\left[x, y_2\right] - f\left[x, y_1\right]}{\Delta y} \bullet \left(y - y_1\right)$$

Equation **GFLIB_Lut2D_Eq16**

It should be noted that due to assumption of the equidistant data points, the division by the interval length is avoided.

It should be noted that the computations are performed with the 16-bit accuracy. In particular the 16 least significant bits are ignored in all multiplications.

The shift amounts shall be provided in the parameters structure (pParam->u16ShamOffset1 and pParam->u16ShamOffset2). The address of the table with the data, the pTable}, shall be defined by the parameter structure member pParam->pf16Table.

The shift amounts, the s $_{\text{Interval1,2}}$ and s $_{\text{Offset1,2}}$, can be computed with the following formulas:

$$s_{\text{Interval1,2}} = 15 - |n|$$

$$s_{\text{Offset1,2}} = |n|$$

Equation **GFLIB_Lut2D_Eq17**

where n is the integer defining the length of the interpolating interval in the range of -1, -2, ... -29.

The computation of the abscissa offset and the interval index can be viewed also in the following way. The input abscissa value can be divided into two parts. The first n most significant bits of the 16-bit halfword, after the sign bit, compose the interval index, in which interpolation is performed. The rest of the bits form the abscissa offset within the interpolating interval. This simple way to calculate the interpolating interval index and the abscissa offset is the consequence of assumption of all interpolating interval lengths equal 2^{-n}.

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 the bilinear interpolation.

CAUTION

The function does not check whether the input values are within a 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.

5.29.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16Out;
GFLIB_LUT2D_T_F16 tr16tMyLut2D = GFLIB_LUT2D_DEFAULT_F16;
tFrac16 pf16Table2D[81] = {FRAC16 (0.9), FRAC16 (0.01), FRAC16 (0.02), FRAC16
(0.03), FRAC16 (0.04), FRAC16 (0.05), FRAC16 (0.06), FRAC16 (0.07), FRAC16 (0.08),
                            FRAC16 (0.1), FRAC16 (0.11), FRAC16 (0.12), FRAC16
(0.13), FRAC16 (0.14), FRAC16 (0.15), FRAC16 (0.16), FRAC16 (0.17), FRAC16 (0.18),
                            FRAC16 (0.2), FRAC16 (0.21), FRAC16 (0.22), FRAC16
```

```
(0.23), FRAC16 (0.24), FRAC16 (0.25), FRAC16 (0.26), FRAC16 (0.27), FRAC16 (0.28),
                                          FRAC16 (0.3), FRAC16 (0.31), FRAC16 (0.32), FRAC16
(0.33), FRAC16 (0.34), FRAC16 (0.35), FRAC16 (0.36), FRAC16 (0.37), FRAC16 (0.38),
                                          FRAC16 (0.4), FRAC16 (0.41), FRAC16 (0.42), FRAC16
(0.43), FRAC16 (0.44), FRAC16 (0.45), FRAC16 (0.46), FRAC16 (0.47), FRAC16 (0.48),
                                          FRAC16 (0.5), FRAC16 (0.51), FRAC16 (0.52), FRAC16
(0.53), FRAC16 (0.54), FRAC16 (0.55), FRAC16 (0.56), FRAC16 (0.57), FRAC16 (0.58),
                                          FRAC16 (0.6), FRAC16 (0.61), FRAC16 (0.62), FRAC16
(0.63), FRAC16 (0.64), FRAC16 (0.65), FRAC16 (0.66), FRAC16 (0.67), FRAC16 (0.68),
                                          FRAC16 (0.7), FRAC16 (0.71), FRAC16 (0.72), FRAC16
(0.73), FRAC16 (0.74), FRAC16 (0.75), FRAC16 (0.76), FRAC16 (0.77), FRAC16 (0.78),
                                          FRAC16 (0.8), FRAC16 (0.81), FRAC16 (0.82), FRAC16
(0.83), FRAC16 (0.84), FRAC16 (0.85), FRAC16 (0.86), FRAC16 (0.87), FRAC16 (0.88)};

               void main(void)
               {
                   // ######################################################################
                   // Pointer is located in the middle of the interpolating data table.
                   // ######################################################################
                   // setting parameters for Lut2D function
                   tr16tMyLut2D.u16ShamOffset1 = (tU16)3;
                   tr16tMyLut2D.u16ShamOffset2 = (tU16)3;
                   tr16tMyLut2D.pf16Table = &(pf16Table2D[40]);

                   // input vector
                   f16In1 = FRAC16 (-0.5);
                   f16In2 = FRAC16 (-0.5);

                   // output should be 0x7333 ~ FRAC16(0.9)
                   f16Out = GFLIB_Lut2D_F16 (f16In1,f16In2,&tr16tMyLut2D);

                   // output should be 0x7333 ~ FRAC16(0.9)
                   f16Out = GFLIB_Lut2D (f16In1,f16In2,&tr16tMyLut2D,F16);

                   // available only if 16-bit fractional implementation selected as default
                   // output should be 0x7333 ~ FRAC16(0.9)
                   f16Out = GFLIB_Lut2D (f16In1,f16In2,&tr16tMyLut2D);

                   // ######################################################################
                   // Pointer is located at the beginning of the interpolating data table.
                   // ######################################################################
                   // setting parameters for Lut2D function
                   tr16tMyLut2D.u16ShamOffset1 = (tU16)3;
                   tr16tMyLut2D.u16ShamOffset2 = (tU16)3;
                   tr16tMyLut2D.pf16Table = &(pf16Table2D[0]);

                   // input vector
                   f16In1 = FRAC16 (0);
                   f16In2 = FRAC16 (0);

                   // output should be 0x7333 ~ FRAC16(0.9)
                   f16Out = GFLIB_Lut2D_F16 (f16In1,f16In2,&tr16tMyLut2D);

                   // output should be 0x7333 ~ FRAC16(0.9)
                   f16Out = GFLIB_Lut2D (f16In1,f16In2,&tr16tMyLut2D,F16);

                   // available only if 16-bit fractional implementation selected as default
                   // output should be 0x7333 ~ FRAC16(0.9)
                   f16Out = GFLIB_Lut2D (f16In1,f16In2,&tr16tMyLut2D);

                   // ######################################################################
                   // Pointer is located at the end of the interpolating data table.
                   // ######################################################################
                   // setting parameters for Lut2D function
                   tr16tMyLut2D.u16ShamOffset1 = (tU16)3;
                   tr16tMyLut2D.u16ShamOffset2 = (tU16)3;
                   tr16tMyLut2D.pf16Table = &(pf16Table2D[80]);

                   // input vector
                   f16In1 = FRAC16 (-1);
```

```
        f16In2 = FRAC16 (-1);

        // output should be 0x7333 ~ FRAC16(0.9)
        f16Out = GFLIB_Lut2D_F16 (f16In1,f16In2,&tr16tMyLut2D);

        // output should be 0x7333 ~ FRAC16(0.9)
        f16Out = GFLIB_Lut2D (f16In1,f16In2,&tr16tMyLut2D,F16);

        // available only if 16-bit fractional implementation selected as default
        // output should be 0x7333 ~ FRAC16(0.9)
        f16Out = GFLIB_Lut2D (f16In1,f16In2,&tr16tMyLut2D);
    }
```

5.29.5 Function GFLIB_Lut2D_FLT

5.29.5.1 Declaration

```
tFloat GFLIB_Lut2D_FLT(tFloat fltIn1, tFloat fltIn2, const GFLIB_LUT2D_T_FLT *const pParam);
```

5.29.5.2 Arguments

Table 5-103. GFLIB_Lut2D_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn1	**input**	First input variable for which 2D interpolation is performed. Input value is in single precision floating data format.
tFloat	fltIn2	**input**	Second input variable for which 2D interpolation is performed. Input value is in single precision floating data format.
const GFLIB_LUT2D_T_FLT *const	pParam	**input**	Pointer to the parameters structure with parameters of the two dimensional look-up table function.

5.29.5.3 Return

The function returns the interpolated value. The output value is in single precision floating point format.

5.29.5.4 Variant Specifics

The interpolating intervals are defined in the table provided by the pfltTable 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. 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:

- the values of the interpolated function are 32 bits long
- the length of each interpolating interval is equal to 2^{-n}, where n is an integer in the range of 1, 2, ... 29
- the abscissa provided for interpolation is 32 bits long

The bilinear interpolation performs the following steps:

1. Count the equidistant interpolating intervals, in which the bilinear interpolation will be performed:

$$\mathrm{fSegmentsX} = (\mathrm{tFloat})(1 << s32\mathrm{ShamOffset1} + 1)$$
$$\mathrm{fSegmentsY} = (\mathrm{tFloat})(1 << s32\mathrm{ShamOffset2} + 1)$$

Equation **GFLIB_Lut2D_Eq4**

where fSegmentsX and fSegmentsY are the interval lengths, the pParam->u32ShamOffset1pParam->u32ShamOffset2 are the shift amounts provided in the parameters structure as the members u32ShamOffset1, u32ShamOffset2. The operator "<<" represents the binary arithmetic left shift. The explicit casting to tFloat data type is required because parameters u32ShamOffset1,2 are unsigne integer numbers which have to be stored as float variables, and it then indicates the position of the first interpolated value in the X-axis and the Y-axis in the table.

2. Compute the position of the interpolated values in the table:

$$\mathrm{s32IntvlX} = (\mathrm{tS32})((\mathrm{fSegmentsX}) \bullet (\mathrm{fltIn1}))$$
$$\mathrm{s32IntvlY} = (\mathrm{tS32})((\mathrm{fSegmentsY}) \bullet (\mathrm{fltIn2}))$$

Equation **GFLIB_Lut2D_Eq5**

where s32IntvlX and s32IntvlY are the coordinates of the first ordinate value at the beginning of the interpolating interval, the fSegmentsX and fSegmentsY are the equidistant interpolating intervals in the both axes and the fltIn1, fltIn2 are the input values provided to the function as arguments. The casting to tS32 data type is a final operation of the equation which cuts the fractional part of the value and then indicates the position of the first interpolated value in the X-axis and the Y-axis in the table.

3. Count the number of the interpolation samples in the X-axis:

$$s32\text{Xrange} = \left(\left(1 < < \text{pParam} \rightarrow s32\text{ShamOffset} + 1\right) + 1\right)$$

Equation **GFLIB_Lut2D_Eq6**

4. Compute the pointer to the first interpolated value z_{11} in the table:

$$\text{psfIntvl} = \left(32 - \text{bit data at address pTable}\right) + s32\text{IntvlX} + \left(s32\text{Xrange} \bullet \text{fltIntvlY}\right)$$

Equation **GFLIB_Lut2D_Eq7**

where the pTable} is the address provided in the parameters structure pParam->fltTable.

5. Compute the interpolated value by the linear interpolation in the X-axis of the y $_1$ value between the ordinates x_1 and x_2 read from the table at the start and the end of the computed interval in the X-axis. The computed abscissa offset is used for the linear interpolation:

$$z_1 = z_{11} + \frac{(x - x_1) \bullet (z_{21} - z_{11})}{x_2 - x_1}$$

Equation **GFLIB_Lut2D_Eq8**

reduced to (division by the interval length is avoided):

$$z_1 = z_{11} + \left(z_{21} - z_{11}\right) \bullet \left(\left(\text{tFloat}\right)\left(x \bullet 2^n\right) - \left(\text{tU32}\right)\left(x \bullet 2^n\right)\right)$$

Equation **GFLIB_Lut2D_Eq9**

interpretation in C language is following:

$$\text{fDX} = \text{fltIn1} * \text{fSegmentsX} - s32\text{IntvlX}$$

$$\text{fltZ1} = \left(*\text{psfIntvl}\right) + \left(\left(\left(*\left(\text{psfIntvl} + 1\right) - \left(*\text{psfIntvl}\right)\right) * \text{fDX}\right)\right)$$

Equation **GFLIB_Lut2D_Eq10**

6. The same computation as shown above in the X-axis by the y $_2$ value:

$$z_2 = z_{12} + \frac{(x - x_1) \bullet (z_{22} - z_{12})}{x_2 - x_1}$$

Equation **GFLIB_Lut2D_Eq11**

reduced to (division by the interval length is avoided):

$$z_2 = z_{12} + (z_{22} - z_{12}) \bullet ((\text{tFloat})(x \bullet 2^n) - (\text{tU32})(x \bullet 2^n))$$

Equation **GFLIB_Lut2D_Eq12**

interpretation in C language is the following:

fltZ2=(*(psfIntvl+s32Xrange))+(((*(psfIntvl+s32Xrange+1))-(*(psfIntvl+s32Xrange)))*(fDX))

Equation **GFLIB_Lut2D_Eq13**

7. Final linear interpolation in the Y-axis:

$$z = z_1 + \frac{(y - y_1) \bullet (z_2 - z_1)}{y_2 - y_1}$$

Equation **GFLIB_Lut2D_Eq14**

reduced to (division by the interval length is avoided):

$$z = z_1 + (z_2 - z_1) \bullet ((\text{tFloat})(y \bullet 2^n) - (\text{tU32})(y \bullet 2^n))$$

Equation **GFLIB_Lut2D_Eq15**

interpretation in C language is the following:

$$\text{return}(\text{fltZ1} + ((\text{fltZ2} - \text{fltZ1}) * (\text{fltIn2} * \text{fSegmentsY} - s\text{32IntvlY})))$$

Equation **GFLIB_Lut2D_Eq16**

The shift amounts shall be provided in the parameters structure pParam->u32ShamOffset1 and pParam->u32ShamOffset2. The address of the table with the data, the pTable}, shall be defined by the parameter structure member pParam->pfltTable.

The offset amounts u32ShamOffset_{12} can be provided by the following formulas:

$$s\text{32ShamOffset1} = |n_1|$$
$$s\text{32ShamOffset2} = |n_2|$$

Equation **GFLIB_Lut2D_Eq17**

where n_1 and n_2 are the integers defining the length of the interpolating interval in the range of 1, 2, ... 29.

This simple way to calculate the interpolating abscissa offsets is the consequence of assuming that all interpolating interval lengths equal 2^{-n1} and 2^{-n2}.

It should be noted that the input abscissa values can be positive or negative. If they are positive then the ordinate values are read as in the ordinary data array, that are at or after the data pointer provided in the parameters structure pParam->pfltTable. However, if they are negative, then the ordinate values are read from the memory, which is located behind the pParam->pfltTable pointer.

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

The function performs a bilinear interpolation.

CAUTION

The function does not check whether the input values are 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. The boundary values are excluded from the input range, and shall not be used as an input values. For a better understanding, please, see the extended code example.

5.29.5.5 Code Example

```
#include "gflib.h"

tFloat fltIn1;
tFloat fltIn2;
tFloat fltOut;
GFLIB_LUT2D_T_FLT trfltMyLut2D = GFLIB_LUT2D_DEFAULT_FLT;
tFloat pfltTable2D[81] = {0.9, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08,
                          0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18,
                          0.2, 0.21, 0.22, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28,
                          0.3, 0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37, 0.38,
                          0.4, 0.41, 0.42, 0.43, 0.44, 0.45, 0.46, 0.47, 0.48,
                          0.5, 0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.57, 0.58,
                          0.6, 0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67, 0.68,
                          0.7, 0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78,
                          0.8, 0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87,
0.88};

void main(void)
{
    // ########################################################################
    // Pointer is located in the middle of the interpolating data table,
```

```
                    // therefore range of the input values is (-0.5,0.5).
                    // ######################################################################
                    // setting parameters for Lut2D function
                    trfltMyLut2D.u32ShamOffset1 = (tU32)3;
                    trfltMyLut2D.u32ShamOffset2 = (tU32)3;
                    trfltMyLut2D.pfltTable = &(pfltTable2D[40]);

                    // input vector
                    fltIn1 = (tFloat)-0.4999999;
                    fltIn2 = (tFloat)-0.4999999;

                    // output should be approximately 0.9
                    fltOut = GFLIB_Lut2D_FLT (fltIn1,fltIn2,&trfltMyLut2D);

                    // output should be approximately 0.9
                    fltOut = GFLIB_Lut2D (fltIn1,fltIn2,&trfltMyLut2D,FLT);

                    // available only if single precision floating point implementation
selected as default
                    // output should be approximately 0.9
                    fltOut = GFLIB_Lut2D (fltIn1,fltIn2,&trfltMyLut2D);

                    // ######################################################################
                    // Pointer is located at the beginning of the interpolating data table,
                    // therefore range of the input values is (0,1).
                    // ######################################################################
                    // setting parameters for Lut2D function
                    trfltMyLut2D.u32ShamOffset1 = (tU32)3;
                    trfltMyLut2D.u32ShamOffset2 = (tU32)3;
                    trfltMyLut2D.pfltTable = &(pfltTable2D[0]);

                    // input vector
                    fltIn1 = (tFloat)0.0000001;
                    fltIn2 = (tFloat)0.0000001;

                    // output should be approximately 0.9
                    fltOut = GFLIB_Lut2D_FLT (fltIn1,fltIn2,&trfltMyLut2D);

                    // output should be approximately 0.9
                    fltOut = GFLIB_Lut2D (fltIn1,fltIn2,&trfltMyLut2D,FLT);

                    // available only if single precision floating point implementation
selected as default
                    // output should be approximately 0.9
                    fltOut = GFLIB_Lut2D (fltIn1,fltIn2,&trfltMyLut2D);

                    // ######################################################################
                    // Pointer is located at the end of the interpolating data table,
                    // therefore range of the input values is (-1,0).
                    // ######################################################################
                    // setting parameters for Lut2D function
                    trfltMyLut2D.u32ShamOffset1 = (tU32)3;
                    trfltMyLut2D.u32ShamOffset2 = (tU32)3;
                    trfltMyLut2D.pfltTable = &(pfltTable2D[80]);

                    // input vector
                    fltIn1 = (tFloat)-0.9999999;
                    fltIn2 = (tFloat)-0.9999999;

                    // output should be approximately 0.9
                    fltOut = GFLIB_Lut2D_FLT (fltIn1,fltIn2,&trfltMyLut2D);

                    // output should be approximately 0.9
                    fltOut = GFLIB_Lut2D (fltIn1,fltIn2,&trfltMyLut2D,FLT);

                    // available only if single precision floating point implementation
selected as default
                    // output should be approximately 0.9
                    fltOut = GFLIB_Lut2D (fltIn1,fltIn2,&trfltMyLut2D);
                }
```

5.30 Function GFLIB_Ramp

The function calculates the up/down ramp with the step increment/decrement defined in the pParam structure.

5.30.1 Description

The GFLIB_Ramp function limits the rate of change of the input signal.

If the desired (input) value is greater than the ramp output value, the function adds the RampUp value to the actual output value. The output cannot be greater than the desired value.

If the desired value is lower than the actual value, the function subtracts the RampDown value from the actual value. The output cannot be lower than the desired value.

Functionality of the implemented ramp algorithm can be explained with use of Figure 5-26

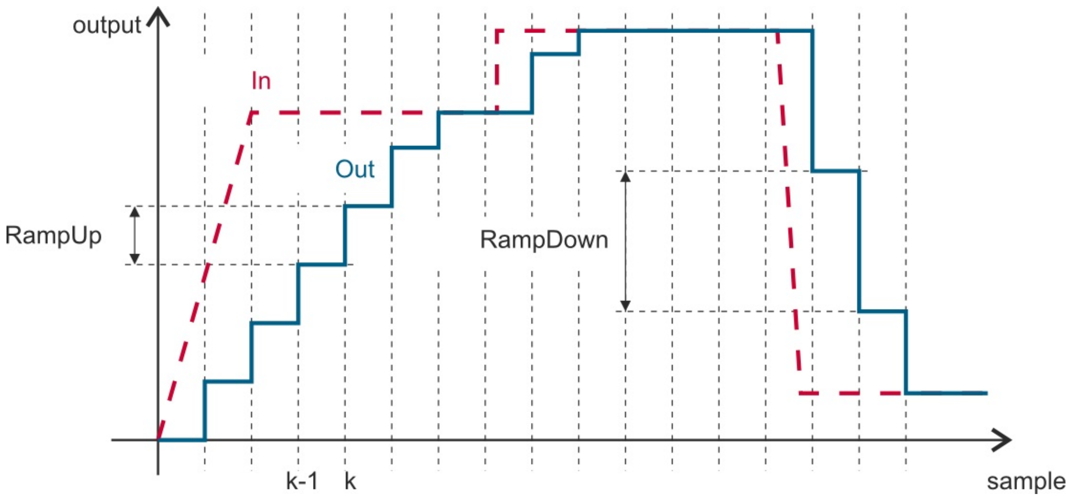

Figure 5-26. GFLIB_Ramp functionality

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 an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.30.2 Re-entrancy

The function is re-entrant.

5.30.3 Function GFLIB_Ramp_F32

5.30.3.1 Declaration

```
tFrac32 GFLIB_Ramp_F32(tFrac32 f32In, GFLIB_RAMP_T_F32 *const pParam);
```

5.30.3.2 Arguments

Table 5-104. 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.

5.30.3.3 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.

5.30.3.4 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;
    // 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);
}
```

5.30.4 Function GFLIB_Ramp_F16

5.30.4.1 Declaration

```
tFrac16 GFLIB_Ramp_F16(tFrac16 f16In, GFLIB_RAMP_T_F16 *const pParam);
```

5.30.4.2 Arguments

Table 5-105. 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.

5.30.4.3 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.

5.30.4.4 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)
    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);
}
```

5.30.5 Function GFLIB_Ramp_FLT

5.30.5.1 Declaration

```
tFloat GFLIB_Ramp_FLT(tFloat fltIn, GFLIB_RAMP_T_FLT *const pParam);
```

5.30.5.2 Arguments

Table 5-106. 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.

5.30.5.3 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 (floating-point overflow, underflow, inexact, invalid operation).

All parameters and states used by the function can be reset during declaration using the GFLIB_RAMP_DEFAULT_FLT macro.

5.30.5.4 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;
GFLIB_RAMP_T_FLT flttrMyRamp = GFLIB_RAMP_DEFAULT_FLT;

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);
    }
```

5.31 Function GFLIB_Sign

This function returns the signum of input value.

5.31.1 Description

The GFLIB_Sign function calculates the sign of the input argument according to the following equation:

$$y_{out} = \begin{cases} 1 & \text{if} \quad x_{in} > 0 \\ 0 & \text{if} \quad x_{in} = 0 \\ -1 & \text{if} \quad x_{in} < 0 \end{cases}$$

Equation **GFLIB_Sign_Eq1**

where:

- y_{out} is the return value
- x_{in} is the input value provided as the In parameter

5.31.2 Re-entrancy

The function is re-entrant.

5.31.3 Function GFLIB_Sign_F32

5.31.3.1 Declaration

```
tFrac32 GFLIB_Sign_F32(tFrac32 f32In);
```

5.31.3.2 Arguments

Table 5-107. GFLIB_Sign_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument.

5.31.3.3 Return

The function returns the sign of the input argument.

5.31.3.4 Variant Specifics

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.

5.31.3.5 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);
}
```

5.31.4 Function GFLIB_Sign_F16

5.31.4.1 Declaration

```
tFrac16 GFLIB_Sign_F16(tFrac16 f16In);
```

5.31.4.2 Arguments

Table 5-108. GFLIB_Sign_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument.

5.31.4.3 Return

The function returns the sign of the input argument.

5.31.4.4 Variant Specifics

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.

5.31.4.5 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);
    }
```

5.31.5 Function GFLIB_Sign_FLT

5.31.5.1 Declaration

```
tFloat GFLIB_Sign_FLT(tFloat fltIn);
```

5.31.5.2 Arguments

Table 5-109. GFLIB_Sign_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument.

5.31.5.3 Return

The function returns the sign of the input argument.

Note

The function may raise floating-point exceptions (floating-point invalid operation).

The input and the output are in single precision floating point data format.

5.31.5.4 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);
}
```

5.32 Function GFLIB_Sin

This function implements polynomial approximation of sine function.

5.32.1 Description

The function GFLIB_Sin calculates the trigonometric sine function using a polynomial approximation.

Note

The input pointer must contain a valid address otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.32.2 Re-entrancy

The function is re-entrant.

5.32.3 Function GFLIB_Sin_F32

5.32.3.1 Declaration

```
tFrac32 GFLIB_Sin_F32(tFrac32 f32In, const GFLIB_SIN_T_F32 *const pParam);
```

5.32.3.2 Arguments

Table 5-110. GFLIB_Sin_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument is a 32-bit number that contains an angle in radians between [- π, π) normalized between [-1, 1).
const GFLIB_SIN_T_F32 *const	pParam	**input**	Pointer to an array of Taylor coefficients.

5.32.3.3 Return

The function returns the sin of the input argument as a fixed point 32-bit number, normalized between [-1, 1).

5.32.3.4 Variant Specifics

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.

Figure 5-27 depicts a floating point *sine* function generated from Matlab and the approximated value of the *sine* function obtained from GFLIB_Sin_F32, plus their difference.

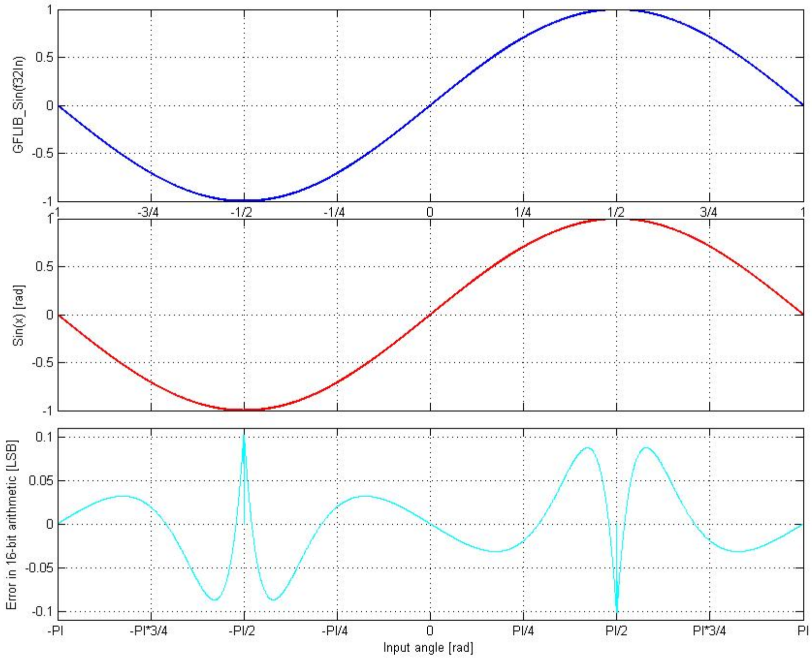

Figure 5-27. sin(x) vs. GFLIB_Sin_F32(f32In)

Note

The function call is slightly different from common approach in the library set. The function can be called in three different ways:

- With implementation postfix (i.e. GFLIB_Sin_F32(f32In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SIN_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Sin(f32In, &pParam, F32), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SIN_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Sin(f32In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_SIN_DEFAULT_F32 approximation coefficients are used.

5.32.3.5 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);
}
```

5.32.4 Function GFLIB_Sin_F16

5.32.4.1 Declaration

```
tFrac16 GFLIB_Sin_F16(tFrac16 f16In, const GFLIB_SIN_T_F16 *const pParam);
```

5.32.4.2 Arguments

Table 5-111. GFLIB_Sin_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument is a 16-bit number that contains an angle in radians between [- π, π) normalized between [-1, 1).
const GFLIB_SIN_T_F16 *const	pParam	**input**	Pointer to an array of Taylor coefficients.

5.32.4.3 Return

The function returns the sin of the input argument as a fixed point 16-bit number, normalized between [-1, 1).

5.32.4.4 Variant Specifics

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_SIN_DEFAULT_F16 structure.

Figure 5-28 depicts a floating point *sine* function generated from Matlab and the approximated value of the *sine* function obtained from GFLIB_Sin_F16, plus their difference.

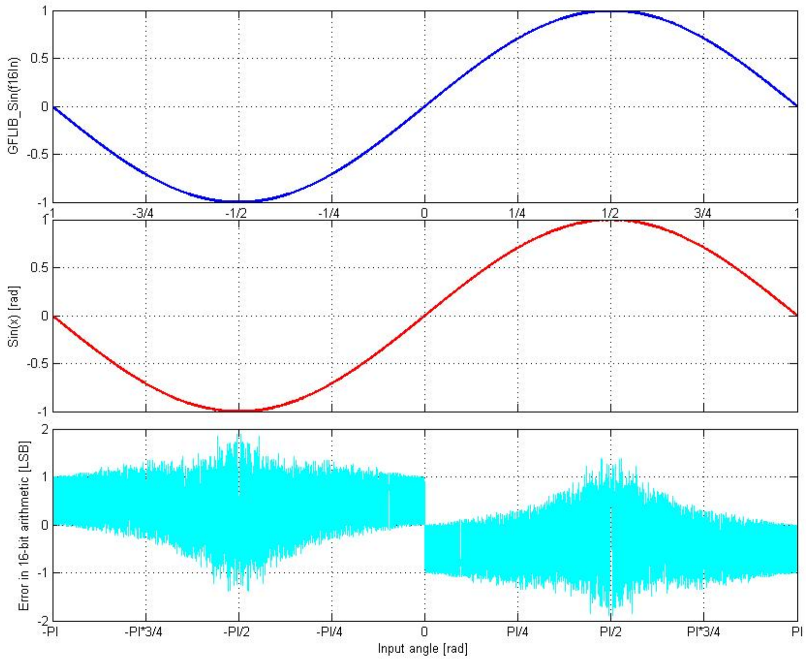

Figure 5-28. sin(x) vs. GFLIB_Sin_F16(f16In)

Note

The function call is slightly different from common approach in the library set. The function can be called in three different ways:

- With implementation postfix (i.e. GFLIB_Sin_F16(f16In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SIN_DEFAULT_F16 symbol. The &pParam parameter is mandatory.

- With additional implementation parameter (i.e. GFLIB_Sin(f16In, &pParam, F16), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SIN_DEFAULT_F16 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Sin(f16In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_SIN_DEFAULT_F16 approximation coefficients are used.

5.32.4.5 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);
}
```

5.32.5 Function GFLIB_Sin_FLT

5.32.5.1 Declaration

```
tFloat GFLIB_Sin_FLT(tFloat fltIn, const GFLIB_SIN_T_FLT *const pParam);
```

5.32.5.2 Arguments

Table 5-112. 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 between [- π, π].
const GFLIB_SIN_T_FLT *const	pParam	**input**	Pointer to an array of approximation coefficients.

5.32.5.3 Return

The function returns the sine of the input argument as a single precision floating point number.

5.32.5.4 Variant Specifics

The function uses a 7th order minimax polynomial approximation; the default polynomial coefficients are provided in the GFLIB_SIN_DEFAULT_FLT structure.

Figure 5-29 depicts a floating point *sine* function generated from Matlab and the approximated value of the *sine* function obtained from GFLIB_Sin_FLT, plus their difference.

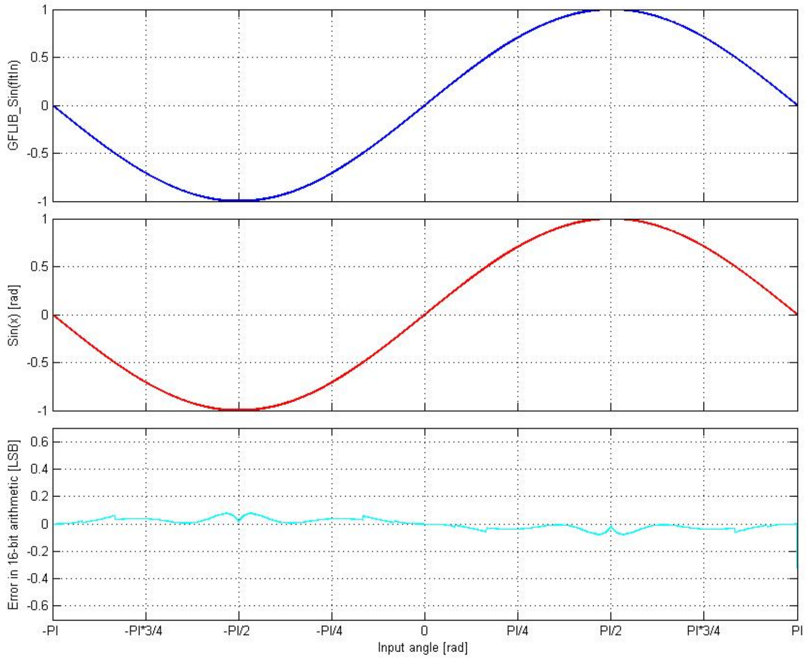

Figure 5-29. sin(x) vs. GFLIB_Sin_FLT(fltIn)

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

The function call is slightly different from common approach in the library set. The function can be called in three different ways:

- With implementation postfix (i.e. GFLIB_Sin_FLT(fltIn, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SIN_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Sin(fltIn, &pParam, FLT), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SIN_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Sin(fltIn), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default

GFLIB_SIN_DEFAULT_FLT approximation coefficients are used.

5.32.5.5 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);
}
```

5.33 Function GFLIB_SinCos

This function implements polynomial approximation of the sine and cosine function.

5.33.1 Description

The function GFLIB_SinCos calculates the trigonometric sine and cosine function using a polynomial approximation. GFLIB_SinCos performs faster than a combination of GFLIB_Sin and GFLIB_Cos.

Note

The input/output pointers must contain valid addresses, otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.33.2 Re-entrancy

The function is re-entrant.

5.33.3 Function GFLIB_SinCos_F32

5.33.3.1 Declaration

```
void GFLIB_SinCos_F32(tFrac32 f32In, SWLIBS_2Syst_F32 *pOut, const GFLIB_SINCOS_T_F32 *const
pParam);
```

5.33.3.2 Arguments

Table 5-113. GFLIB_SinCos_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument is a 32-bit number that contains an angle in radians between [- 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.

5.33.3.3 Return

void

5.33.3.4 Variant Specifics

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. Refer to GFLIB_Sin_F32 and GFLIB_Cos_F32 to see the graphs of output values.

Note

The function call is slightly different from common approach in the library set. The function can be called in three different ways:

- With implementation postfix (i.e. GFLIB_SinCos_F32(f32In, &pOut, &pParam)), where the &pOut is the pointer to output values and &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SINCOS_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_SinCos(f32In, &pOut, &pParam, F32)), where the &pOut is the pointer to output values and &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SINCOS_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_SinCos(f32In, &pOut, &pParam)), where the &pOut is the pointer to output values and &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_SINCOS_DEFAULT_F32 approximation coefficients are used.

5.33.3.5 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);

    // 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);
}
```

5.33.4 Function GFLIB_SinCos_F16

5.33.4.1 Declaration

```
void GFLIB_SinCos_F16(tFrac16 f16In, SWLIBS_2Syst_F16 *pOut, const GFLIB_SINCOS_T_F16 *const
pParam);
```

5.33.4.2 Arguments

Table 5-114. GFLIB_SinCos_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument is a 16-bit number that contains an angle in radians between [- 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.

5.33.4.3 Return

void

5.33.4.4 Variant Specifics

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. Refer to GFLIB_Sin_F16 and GFLIB_Cos_F16 to see the graphs of output values.

Note

Due to effectivity reasons this function is implemented using intrinsic functions and inline assembly and is therefore not ANSI-C compliant. Note that some compilers do not support the enhanced features.

The function call is slightly different from common approach in the library set. The function can be called in three different ways:

- With implementation postfix (i.e. GFLIB_SinCos_F16(f16In, &pOut, &pParam)), where the &pOut is the pointer to output values and &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SINCOS_DEFAULT_F16 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_SinCos(f16In, &pOut, &pParam, F16)), where the &pOut is the pointer to output values and &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SINCOS_DEFAULT_F16 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_SinCos(f16In, &pOut, &pParam)), where the &pOut is the pointer to output values and &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_SINCOS_DEFAULT_F16 approximation coefficients are used.

5.33.4.5 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);
}
```

5.33.5 Function GFLIB_SinCos_FLT

5.33.5.1 Declaration

```
void GFLIB_SinCos_FLT(tFloat fltIn, SWLIBS_2Syst_FLT *pOut, const GFLIB_SINCOS_T_FLT *const
pParam);
```

5.33.5.2 Arguments

Table 5-115. 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 between [- π, π].
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

Table continues on the next page...

Table 5-115. GFLIB_SinCos_FLT arguments (continued)

Type	Name	Direction	Description
			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.

5.33.5.3 Return

void

5.33.5.4 Variant Specifics

The function uses a 7th order minimax polynomial approximation; the default polynomial coefficients are provided in the GFLIB_SINCOS_DEFAULT_FLT structure. Refer to GFLIB_Sin_FLT and GFLIB_Cos_FLT to see the graphs of output values.

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

The function call is slightly different from common approach in the library set. The function can be called in three different ways:

- With implementation postfix (i.e. GFLIB_SinCos_FLT(fltIn, &pOut, &pParam)), where the &pOut is the pointer to output values and &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SINCOS_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_SinCos(fltIn, &pOut, &pParam, FLT)), where the &pOut is the pointer to output values and &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be

replaced with GFLIB_SINCOS_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_SinCos(fltIn, &pOut, &pParam)), where the &pOut is the pointer to output values and &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_SINCOS_DEFAULT_FLT approximation coefficients are used.

5.33.5.5 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
    // ##############################################################

    // output should be:
    // pfltOutput.fltArg1 ~ sin(fltAngle) = 1.0f
    // pfltOutput.fltArg2 ~ cos(fltAngle) = 0f
    GFLIB_SinCos (fltAngle &pfltOutput);
}
```

5.34 Function GFLIB_Sqrt

This function returns the square root of input value.

5.34.1 Description

The GFLIB_Sqrt function calculates the square root of the input value.

5.34.2 Re-entrancy

The function is re-entrant.

5.34.3 Function GFLIB_Sqrt_F32

5.34.3.1 Declaration

```
tFrac32 GFLIB_Sqrt_F32(tFrac32 f32In);
```

5.34.3.2 Arguments

Table 5-116. GFLIB_Sqrt_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	The input value.

5.34.3.3 Return

The function returns the square root of the input value. The return value is within the [0, 1) fraction range.

5.34.3.4 Variant Specifics

The function uses a floating-point square root instruction with appropriate input/output conversions.

Note

The function may raise floating-point exceptions (floating-point underflow, inexact, invalid operation).

The valid input range is [0, 1). Negative inputs will yield undefined results. Due to effectivity reason this function is written as inline assembly and thus is not ANSI-C compliant.

5.34.3.5 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;

void main(void)
{
    // input value = 0.5
    f32In = FRAC32 (0.5);

    // output should be 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);
}
```

5.34.4 Function GFLIB_Sqrt_F16

5.34.4.1 Declaration

```
tFrac16 GFLIB_Sqrt_F16(tFrac16 f16In);
```

5.34.4.2 Arguments

Table 5-117. GFLIB_Sqrt_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	The input value.

5.34.4.3 Return

The function returns the square root of the input value. The return value is within the [0, 1) fraction range.

5.34.4.4 Variant Specifics

The function uses a floating-point square root instruction with appropriate input/output conversions.

Note

The function may raise floating-point exceptions (floating-point underflow, inexact, invalid operation).

The valid input range is [0, 1). Negative inputs will yield undefined results. Due to effectivity reason this function is written as inline assembly and thus is not ANSI-C compliant.

5.34.4.5 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
{
    // input value = 0.5
    f16In = FRAC16 (0.5);

    // output should be 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);
}
```

5.34.5 Function GFLIB_Sqrt_FLT

5.34.5.1 Declaration

```
tFloat GFLIB_Sqrt_FLT(tFloat fltIn);
```

5.34.5.2 Arguments

Table 5-118. GFLIB_Sqrt_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	The input value.

5.34.5.3 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 (floating-point underflow, inexact, invalid operation).

Valid inputs are nonnegative numbers. Due to effectivity reason this function is written using inline assembly and thus is not ANSI-C compliant.

5.34.5.4 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);

    // 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);
}
```

5.35 Function GFLIB_Tan

This function implements polynomial approximation of tangent function.

5.35.1 Description

The function GFLIB_Tan calculates the trigonometric tangent function using a polynomial approximation.

Note

The input pointer must contain a valid address otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.35.2 Re-entrancy

The function is re-entrant.

5.35.3 Function GFLIB_Tan_F32

5.35.3.1 Declaration

```
tFrac32 GFLIB_Tan_F32(tFrac32 f32In, const GFLIB_TAN_T_F32 *const pParam);
```

5.35.3.2 Arguments

Table 5-119. GFLIB_Tan_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument is a 32-bit number that contains an angle in radians between [- π, π) normalized between [-1, 1).
const GFLIB_TAN_T_F32 *const	pParam	**input**	Pointer to an array of Taylor coefficients.

5.35.3.3 Return

The function returns tan(π * f32In) as a fixed point 32-bit number, normalized between [-1, 1).

5.35.3.4 Variant Specifics

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_F32 structure.

Figure 5-30 depicts a floating point *tangent* function generated from Matlab and the approximated value of *the* tangent function obtained from GFLIB_Tan_F32, plus their difference.

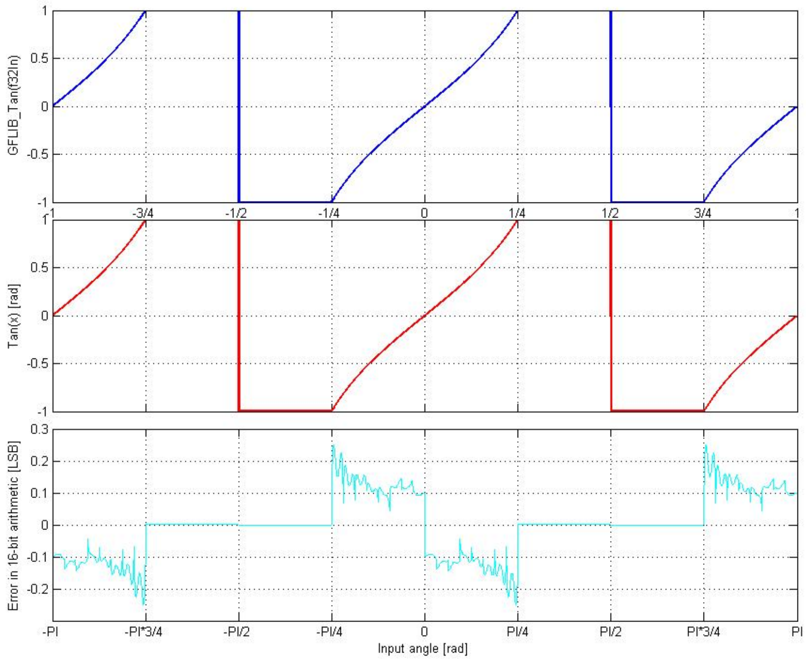

Figure 5-30. tan(x) vs. GFLIB_Tan(f32In)

Note

The function call is slightly different from common approach in the library set. The function can be called in three different ways:

- With implementation postfix (i.e. GFLIB_Tan_F32(f32In, &pParam)), where the &pParam is pointer to

approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_TAN_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Tan(f32In, &pParam, F32), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_TAN_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Tan(f32In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_TAN_DEFAULT_F32 approximation coefficients are used.

5.35.3.5 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);
}
```

5.35.4 Function GFLIB_Tan_F16

5.35.4.1 Declaration

```
tFrac16 GFLIB_Tan_F16(tFrac16 f16In, const GFLIB_TAN_T_F16 *const pParam);
```

5.35.4.2 Arguments

Table 5-120. GFLIB_Tan_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument is a 16-bit number that contains an angle in radians between [- π, π) normalized between [-1, 1).
const GFLIB_TAN_T_F16 *const	pParam	**input**	Pointer to an array of Taylor coefficients.

5.35.4.3 Return

The function returns tan(π * f16In) as a fixed point 16-bit number, normalized between [-1, 1).

5.35.4.4 Variant Specifics

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.

Note

The function call is slightly different from common approach in the library set. The function can be called in three different ways:

- With implementation postfix (i.e. GFLIB_Tan_F16(f16In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_TAN_DEFAULT_F16 symbol. The &pParam parameter is , mandatory.
- With additional implementation parameter (i.e. GFLIB_Tan(f16In, &pParam, F16), where the &pParam is pointer to approximation coefficients. In case the default

approximation coefficients are used, the &pParam must be replaced with GFLIB_TAN_DEFAULT_F16 symbol. The &pParam parameter is mandatory.

- With preselected default implementation (i.e. GFLIB_Tan(f16In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_TAN_DEFAULT_F16 approximation coefficients are used.

5.35.4.5 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);

    // 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);
}
```

5.35.5 Function GFLIB_Tan_FLT

5.35.5.1 Declaration

```
tFloat GFLIB_Tan_FLT(tFloat fltIn, const GFLIB_TAN_T_FLT *const pParam);
```

5.35.5.2 Arguments

Table 5-121. 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 between (- π, π).
const GFLIB_TAN_T_FLT *const	pParam	**input**	Pointer to an array of approximation coefficients.

5.35.5.3 Return

The function returns tan(fltIn) as a single precision floating point number.

5.35.5.4 Variant Specifics

The function uses a rational polynomial approximation; the default polynomial coefficients are provided in the GFLIB_TAN_DEFAULT_FLT structure.

Figure 5-31 depicts a floating point *tangent* function generated from Matlab and the approximated value of the *tangent* function obtained from GFLIB_Tan_FLT, plus the relative error of the approximation in units of ULP.

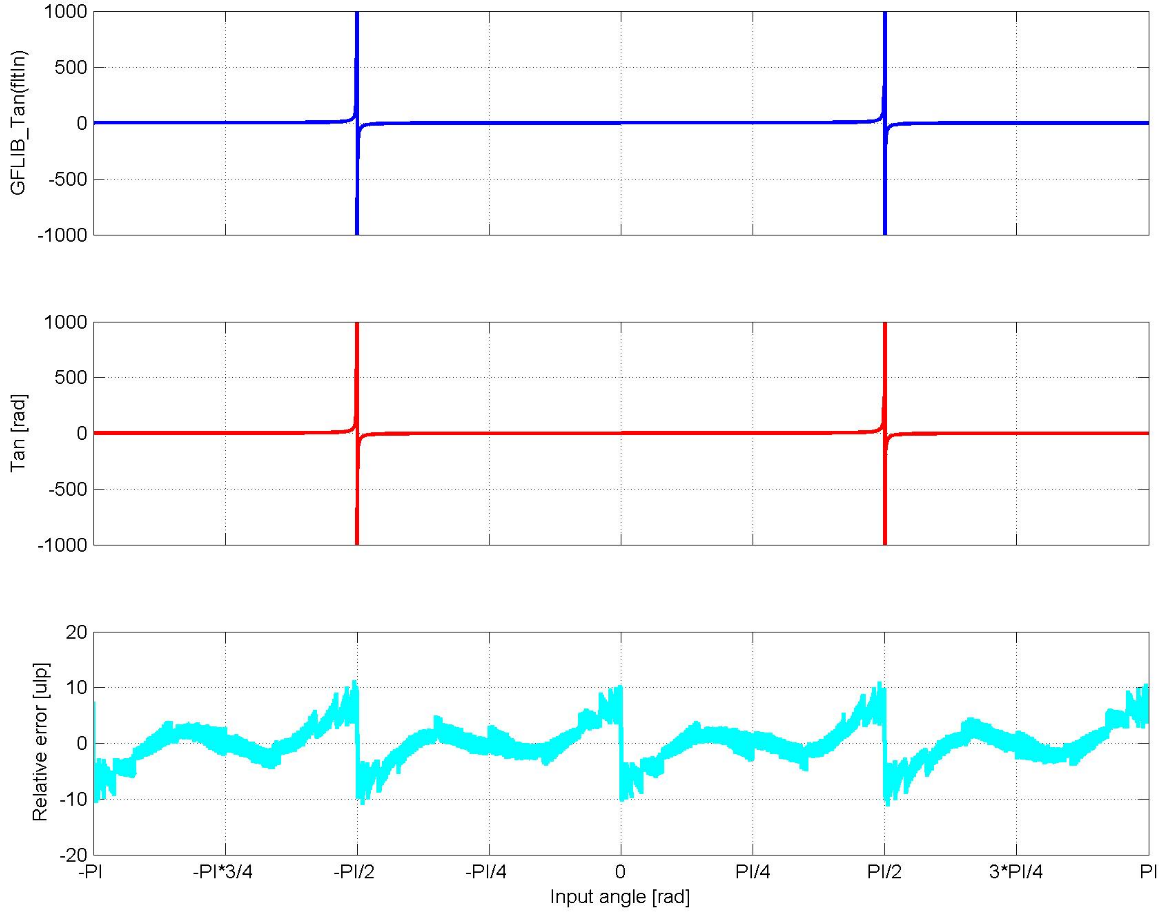

Figure 5-31. tan(x) vs. GFLIB_Tan(fltIn)

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation, zero divide).

The input pointer must contain a valid address otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

The input angle (fltIn) is in single precision floating point format considering the input angle directly in radians. The approximation accuracy is guaranteed only for the input interval $-\pi < fltIn < \pi$. The function should not be used beyond that range. The function call is slightly different from common approach in the library set. The function can be called in three different ways:

- With implementation postfix (i.e. GFLIB_Tan_FLT(fltIn, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_TAN_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Tan(fltIn, &pParam, FLT), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_TAN_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Tan(fltIn, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_TAN_DEFAULT_FLT approximation coefficients are used.

5.35.5.5 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);
}
```

5.36 Function GFLIB_UpperLimit

This function tests whether the input value is below the upper limit.

5.36.1 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 an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.36.2 Re-entrancy

The function is re-entrant.

5.36.3 Function GFLIB_UpperLimit_F32

5.36.3.1 Declaration

```
tFrac32 GFLIB_UpperLimit_F32(tFrac32 f32In, const GFLIB_UPPERLIMIT_T_F32 *const pParam);
```

5.36.3.2 Arguments

Table 5-122. 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.

5.36.3.3 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.

5.36.3.4 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);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = GFLIB_UpperLimit (f32In,&f32trMyUpperLimit);
}
```

5.36.4 Function GFLIB_UpperLimit_F16

5.36.4.1 Declaration

```
tFrac16 GFLIB_UpperLimit_F16(tFrac16 f16In, const GFLIB_UPPERLIMIT_T_F16 *const pParam);
```

5.36.4.2 Arguments

Table 5-123. GFLIB_UpperLimit_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input value.

Table continues on the next page...

Table 5-123. GFLIB_UpperLimit_F16 arguments (continued)

Type	Name	Direction	Description
const GFLIB_UPPERLIMIT_T_F16 *const	pParam	**input**	Pointer to the limits structure.

5.36.4.3 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.

5.36.4.4 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
    // ##############################################################

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = GFLIB_UpperLimit (f16In,&f16trMyUpperLimit);
}
```

5.36.5 Function GFLIB_UpperLimit_FLT

5.36.5.1 Declaration

```
tFloat GFLIB_UpperLimit_FLT(tFloat fltIn, const GFLIB_UPPERLIMIT_T_FLT *const pParam);
```

5.36.5.2 Arguments

Table 5-124. 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.

5.36.5.3 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 (floating-point invalid operation).

5.36.5.4 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);
}
```

5.37 Function GFLIB_VectorLimit

This function limits the magnitude of the input vector.

5.37.1 Description

The GFLIB_VectorLimit function limits the magnitude of the input vector, keeping its direction unchanged. Limitation is performed as follows:

$$y_{out} = \begin{cases} \frac{y_{in}}{\sqrt{x_{in}^2 + y_{in}^2}} \bullet L & \text{if } \sqrt{x_{in}^2 + y_{in}^2} > L \\ y_{in} & \text{if } \sqrt{x_{in}^2 + y_{in}^2} \le L \end{cases}$$

Equation **GFLIB_VectorLimit_Eq1**

$$x_{out} = \begin{cases} \frac{x_{in}}{\sqrt{x_{in}^2 + y_{in}^2}} \bullet L & \text{if } \sqrt{x_{in}^2 + y_{in}^2} > L \\ x_{in} & \text{if } \sqrt{x_{in}^2 + y_{in}^2} \le L \end{cases}$$

Equation **GFLIB_VectorLimit_Eq2**

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.

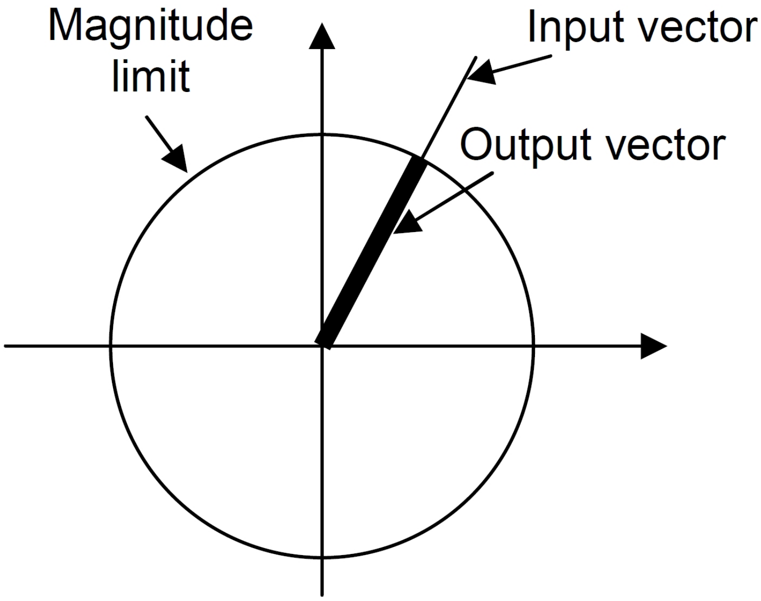

Figure 5-32. Graphical interpretation of the GFLIB_VectorLimit function.

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 an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.37.2 Re-entrancy

The function is re-entrant.

5.37.3 Function GFLIB_VectorLimit_F32

5.37.3.1 Declaration

```
tBool GFLIB_VectorLimit_F32(SWLIBS_2Syst_F32 *const pOut, const SWLIBS_2Syst_F32 *const pIn,
const GFLIB_VECTORLIMIT_T_F32 *const pParam);
```

5.37.3.2 Arguments

Table 5-125. GFLIB_VectorLimit_F32 arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure of the input vector.
SWLIBS_2Syst_F32 *const	pOut	**output**	Pointer to the structure of the limited output vector.
const GFLIB_VECTORLIMIT_T_F32 *const	pParam	**input**	Pointer to the parameters structure.

5.37.3.3 Return

The function will return TRUE if the input vector is being limited or FALSE otherwise.

5.37.3.4 Variant Specifics

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 (floating-point underflow, inexact, invalid operation).

5.37.3.5 Code Example

```
#include "gflib.h"

SWLIBS_2Syst_F32 f32pIn;
```

```
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);
}
```

5.37.4 Function GFLIB_VectorLimit_F16

5.37.4.1 Declaration

```
tBool GFLIB_VectorLimit_F16(SWLIBS_2Syst_F16 *const pOut, const SWLIBS_2Syst_F16 *const pIn,
const GFLIB_VECTORLIMIT_T_F16 *const pParam);
```

5.37.4.2 Arguments

Table 5-126. GFLIB_VectorLimit_F16 arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_F16 *const	pIn	**input**	Pointer to the structure of the input vector.
SWLIBS_2Syst_F16 *const	pOut	**output**	Pointer to the structure of the limited output vector.

Table continues on the next page...

Table 5-126. GFLIB_VectorLimit_F16 arguments (continued)

Type	Name	Direction	Description
const GFLIB_VECTORLIMIT_T_F16 *const	pParam	**input**	Pointer to the parameters structure.

5.37.4.3 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 (floating-point underflow, inexact, invalid operation).

5.37.4.4 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);
    }
```

5.37.5 Function GFLIB_VectorLimit_FLT

5.37.5.1 Declaration

```
tBool GFLIB_VectorLimit_FLT(SWLIBS_2Syst_FLT *const pOut, const SWLIBS_2Syst_FLT *const pIn,
const GFLIB_VECTORLIMIT_T_FLT *const pParam);
```

5.37.5.2 Arguments

Table 5-127. GFLIB_VectorLimit_FLT arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_FLT *const	pIn	**input**	Pointer to the structure of the input vector.
SWLIBS_2Syst_FLT *const	pOut	**output**	Pointer to the structure of the limited output vector.
const GFLIB_VECTORLIMIT_T_FLT *const	pParam	**input**	Pointer to the parameters structure.

5.37.5.3 Return

The function will return TRUE if the input vector is being limited, or FALSE otherwise.

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation, zero divide).

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.

5.37.5.4 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);

    // 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);
}
```

5.38 Function GFLIB_VLog10_FLT

This function calculates a base-10 logarithm of absolute values of an array.

5.38.1 Declaration

```
void GFLIB_VLog10_FLT(tFloat *pInOut, tU32 u32N, const GFLIB_VLOG10_T_FLT *const pParam);
```

5.38.2 Arguments

Table 5-128. 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.

5.38.3 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.

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact).

5.38.4 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);
}
```

5.39 Function GMCLIB_Clark

The function implements the Clarke transformation.

5.39.1 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:

$$i_\alpha = i_A$$

Equation **GMCLIB_Clark_Eq1**

$$i_\beta = (i_B - i_C) \cdot \frac{1}{\sqrt{3}}$$

Equation **GMCLIB_Clark_Eq2**

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 an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.39.2 Re-entrancy

The function is re-entrant.

5.39.3 Function GMCLIB_Clark_F32

5.39.3.1 Declaration

```
void GMCLIB_Clark_F32(SWLIBS_2Syst_F32 *const pOut, const SWLIBS_3Syst_F32 *const pIn);
```

5.39.3.2 Arguments

Table 5-129. 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).

5.39.3.3 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 = 0x5A827999 ~ FRAC32(0.707106781)
    // output should be f32trAlBe.f32Arg2 = 0x5A827999 ~ FRAC32(0.707106781)
    GMCLIB_Clark_F32 (&f32trAlBe,&f32trAbc);

    // output should be f32trAlBe.f32Arg1 = 0x5A827999 ~ FRAC32(0.707106781)
    // output should be 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)
    // output should be f32trAlBe.f32Arg2 = 0x5A827999 ~ FRAC32(0.707106781)
    GMCLIB_Clark (&f32trAlBe,&f32trAbc);
}
```

5.39.4 Function GMCLIB_Clark_F16

5.39.4.1 Declaration

```
void GMCLIB_Clark_F16(SWLIBS_2Syst_F16 *const pOut, const SWLIBS_3Syst_F16 *const pIn);
```

5.39.4.2 Arguments

Table 5-130. 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).

5.39.4.3 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)
    // output should be f16trAlBe.f16Arg2 = 0x5A82 ~ FRAC16(0.707106781)
    GMCLIB_Clark_F16 (&f16trAlBe,&f16trAbc);

    // output should be f16trAlBe.f16Arg1 = 0x5A82 ~ FRAC16(0.707106781)
```

```
    // output should be 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)
    // output should be f16trAlBe.f16Arg2 = 0x5A82 ~ FRAC16(0.707106781)
    GMCLIB_Clark (&f16trAlBe,&f16trAbc);
}
```

5.39.5 Function GMCLIB_Clark_FLT

5.39.5.1 Declaration

```
void GMCLIB_Clark_FLT(SWLIBS_2Syst_FLT *const pOut, const SWLIBS_3Syst_FLT *const pIn);
```

5.39.5.2 Arguments

Table 5-131. 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 (floating-point overflow, underflow, inexact, invalid operation).

5.39.5.3 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
        // output should be flttrAlBe.fltArg2 = 0.707106781
        GMCLIB_Clark_FLT (&flttrAlBe,&flttrAbc);

        // output should be flttrAlBe.fltArg1 = 0.707106781
        // output should be 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
        // output should be flttrAlBe.fltArg2 = 0.707106781
        GMCLIB_Clark (&flttrAlBe,&flttrAbc);
    }
```

5.40 Function GMCLIB_ClarkInv

The function implements the inverse Clarke transformation.

5.40.1 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:

$$i_A = i_\alpha$$

Equation **GMCLIB_ClarkInv_Eq1**

$$i_B = -\frac{1}{2} \cdot i_\alpha + \frac{\sqrt{3}}{2} \cdot i_\beta$$

Equation **GMCLIB_ClarkInv_Eq2**

$$i_C = -\frac{1}{2} \cdot i_\alpha - \frac{\sqrt{3}}{2} \cdot i_\beta$$

Equation **GMCLIB_ClarkInv_Eq3**

Note

The input/output pointers must contain valid addresses, otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.40.2 Re-entrancy

The function is re-entrant.

5.40.3 Function GMCLIB_ClarkInv_F32

5.40.3.1 Declaration

```
void GMCLIB_ClarkInv_F32(SWLIBS_3Syst_F32 *const pOut, const SWLIBS_2Syst_F32 *const pIn);
```

5.40.3.2 Arguments

Table 5-132. 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).

5.40.3.3 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)
        // output should be f32trAbc.f32Arg2 = 0x2120FB83 ~ FRAC32(0.258819045)
        // output should be f32trAbc.f32Arg3 = 0x845C8AE5 ~ FRAC32(-0.965925826)
        GMCLIB_ClarkInv_F32 (&f32trAbc,&f32trAlBe);

        // output should be f32trAbc.f32Arg1 = 0x5A827999 ~ FRAC32(0.707106781)
        // output should be f32trAbc.f32Arg2 = 0x2120FB83 ~ FRAC32(0.258819045)
        // output should be 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)
        // output should be f32trAbc.f32Arg2 = 0x2120FB83 ~ FRAC32(0.258819045)
        // output should be f32trAbc.f32Arg3 = 0x845C8AE5 ~ FRAC32(-0.965925826)
        GMCLIB_ClarkInv (&f32trAbc,&f32trAlBe);
    }
```

5.40.4 Function GMCLIB_ClarkInv_F16

5.40.4.1 Declaration

```
void GMCLIB_ClarkInv_F16(SWLIBS_3Syst_F16 *const pOut, const SWLIBS_2Syst_F16 *const pIn);
```

5.40.4.2 Arguments

Table 5-133. 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).

5.40.4.3 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)
    // output should be f16trAbc.f16Arg2 = 0x2120 ~ FRAC16(0.258819045)
    // output should be f16trAbc.f16Arg3 = 0x845C ~ FRAC16(-0.965925826)
    GMCLIB_ClarkInv_F16 (&f16trAbc,&f16trAlBe);

    // output should be f16trAbc.f16Arg1 = 0x5A82 ~ FRAC16(0.707106781)
    // output should be f16trAbc.f16Arg2 = 0x2120 ~ FRAC16(0.258819045)
    // output should be 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)
    // output should be f16trAbc.f16Arg2 = 0x2120 ~ FRAC16(0.258819045)
    // output should be f16trAbc.f16Arg3 = 0x845C ~ FRAC16(-0.965925826)
    GMCLIB_ClarkInv (&f16trAbc,&f16trAlBe);
}
```

5.40.5 Function GMCLIB_ClarkInv_FLT

5.40.5.1 Declaration

```
void GMCLIB_ClarkInv_FLT(SWLIBS_3Syst_FLT *const pOut, const SWLIBS_2Syst_FLT *const pIn);
```

5.40.5.2 Arguments

Table 5-134. 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 (floating-point overflow, underflow, inexact, invalid operation).

5.40.5.3 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
    // output should be flttrAbc.fltArg2 = 0.258819045
    // output should be flttrAbc.fltArg3 = -0.965925826
    GMCLIB_ClarkInv_FLT (&flttrAbc,&flttrAlBe);

    // output should be flttrAbc.fltArg1 = 0.707106781
    // output should be flttrAbc.fltArg2 = 0.258819045
    // output should be flttrAbc.fltArg3 = -0.965925826
    GMCLIB_ClarkInv (&flttrAbc,&flttrAlBe,FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be flttrAbc.fltArg1 ~ phA = 0.707106781
    // output should be flttrAbc.fltArg2 ~ phB = 0.258819045
    // output should be flttrAbc.fltArg3 ~ phC = -0.965925826
    GMCLIB_ClarkInv (&flttrAbc,&flttrAlBe);
}
```

5.41 Function GMCLIB_DecouplingPMSM

This function calculates the cross-coupling voltages to eliminate the dq axis coupling causing on-linearity of the field oriented control.

5.41.1 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:

$$\begin{bmatrix} u_d \\ u_q \end{bmatrix} = R_s \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} L_d & 0 \\ 0 & L_q \end{bmatrix} \frac{d}{\mathrm{dt}} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \omega_e \begin{bmatrix} -L_q \\ L_d \end{bmatrix} \begin{bmatrix} i_q \\ i_d \end{bmatrix} + \omega_e \psi_{\mathrm{pm}} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

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 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:

$$u_d = R_s i_d + L_d \frac{di_d}{\mathrm{dt}}$$

$$u_q = R_s i_q + L_q \frac{\mathrm{d}i_q}{\mathrm{dt}}$$

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:

$$u_{\mathrm{dcomp}} = -\omega_e L_q i_q$$

$$u_{\mathrm{qcomp}} = \omega_e L_d i_d$$

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:

$$u_{d_{\mathrm{dec}}} = u_d + u_{d_{\mathrm{comnp}}}$$
$$u_{q_{\mathrm{dec}}} = u_q + u_{q_{\mathrm{comp}}}$$

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 an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.41.2 Re-entrancy

The function is re-entrant.

5.41.3 Function GMCLIB_DecouplingPMSM_F32

5.41.3.1 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);
```

5.41.3.2 Arguments

Table 5-135. 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}).

5.41.3.3 Variant Specifics

Substituting GMCLIB_DecouplingPMSM_Eq3 into #eq4_GMCLIB_DecouplingPMSM, and normalizing #eq4_GMCLIB_DecouplingPMSM, results in the following set of equations:

$$u_{df_{dec}} \bullet U_{max} = u_{df} \bullet U_{max} - \omega_{ef} \bullet \Omega_{max} \bullet L_q \bullet i_{qf} \bullet I_{max}$$
$$u_{qf_{dec}} \bullet U_{max} = u_{qf} \bullet U_{max} + \omega_{ef} \bullet \Omega_{max} \bullet L_d \bullet i_{df} \bullet I_{max}$$

Equation **GMCLIB_DecouplingPMSM_Eq5**

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:

$$u_{df_{dec}} = \frac{u_{d_{dec}}}{U_{max}} \qquad u_{qf_{dec}} = \frac{u_{q_{dec}}}{U_{max}}$$
$$u_{df} = \frac{u_d}{U_{max}} \qquad u_{qf} = \frac{u_q}{U_{max}}$$
$$i_{df} = \frac{i_d}{I_{max}} \qquad i_{qf} = \frac{i_q}{I_{max}}$$
$$\omega_{ef} = \frac{\omega_e}{\Omega_{max}}$$

Equation **GMCLIB_DecouplingPMSM_Eq6**

Further, rearranging GMCLIB_DecouplingPMSM_Eq5 results in:

$$u_{df_{dec}} = u_{df} - \omega_{ef} \bullet i_{qf} \frac{L_q \bullet \Omega_{max} \bullet I_{max}}{U_{max}} = u_{df} - \omega_{ef} \bullet i_{qf} \bullet k_d$$
$$u_{qf_{dec}} = u_{qf} - \omega_{ef} \bullet i_{df} \frac{L_d \bullet \Omega_{max} \bullet I_{max}}{U_{max}} = u_{qf} - \omega_{ef} \bullet i_{df} \bullet k_q$$

Equation **GMCLIB_DecouplingPMSM_Eq7**

where k_d and k_q are coefficients calculated as:

$$k_d = L_q \bullet \Omega_{max} \bullet \frac{I_{max}}{U_{max}}$$
$$k_q = L_d \bullet \Omega_{max} \bullet \frac{I_{max}}{U_{max}}$$

Equation **GMCLIB_DecouplingPMSM_Eq8**

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:

$$k_{d\_shift} = \mathrm{ceil}\left(\frac{\log(k_d)}{\log(2)}\right)$$
$$k_{q\_shift} = \mathrm{ceil}\left(\frac{\log(k_q)}{\log(2)}\right)$$

Equation **GMCLIB_DecouplingPMSM_Eq9**

Using scaling coefficients GMCLIB_DecouplingPMSM_Eq9, the fractional representation of coefficients k_d and k_q from GMCLIB_DecouplingPMSM_Eq8 are derived as follows:

$$k_{df} = k_d \bullet 2^{-k_{d\_shift}}$$
$$k_{qf} = k_q \bullet 2^{-k_{q\_shift}}$$

Equation **GMCLIB_DecouplingPMSM_Eq10**

Substituting GMCLIB_DecouplingPMSM_Eq8 - GMCLIB_DecouplingPMSM_Eq10 into GMCLIB_DecouplingPMSM_Eq7 results in the final form of the equation set, actually implemented in the GMCLIB_DecouplingPMSM_F32 function:

$$u_{df_{dec}} = u_{df} - \omega_{ef} \bullet i_{qf} \bullet k_{df} \bullet 2^{k_{d\_shift}}$$
$$u_{qf_{dec}} = u_{qf} + \omega_{ef} \bullet i_{df} \bullet k_{qf} \bullet 2^{k_{q\_shift}}$$

Equation **GMCLIB_DecouplingPMSM_Eq11**

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.

5.41.3.4 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]
                  // output should be 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]
                  // output should be 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]
                  // output should be f32trUDecDQ.f32Arg2 ~ 0x66666666 ~ FRAC32(0.8)*50V
~=40[V]
                  GMCLIB_DecouplingPMSM (&f32trUDecDQ,&f32trUDQ,&f32trIDQ,f32We,&f32trDec);
                }
```

5.41.4 Function GMCLIB_DecouplingPMSM_F16

5.41.4.1 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);
```

5.41.4.2 Arguments

Table 5-136. 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.

Table continues on the next page...

Table 5-136. GMCLIB_DecouplingPMSM_F16 arguments (continued)

Type	Name	Direction	Description
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}).

5.41.4.3 Variant Specifics

Substituting GMCLIB_DecouplingPMSM_Eq3 into #eq4_GMCLIB_DecouplingPMSM, and normalizing #eq4_GMCLIB_DecouplingPMSM, results in the following set of equations:

$$u_{df_{dec}} \bullet U_{max} = u_{df} \bullet U_{max} - \omega_{ef} \bullet \Omega_{max} \bullet L_q \bullet i_{qf} \bullet I_{max}$$
$$u_{qf_{dec}} \bullet U_{max} = u_{qf} \bullet U_{max} + \omega_{ef} \bullet \Omega_{max} \bullet L_d \bullet i_{df} \bullet I_{max}$$

Equation **GMCLIB_DecouplingPMSM_Eq5**

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:

$$\begin{aligned} u_{df_{dec}} &= \frac{u_{d_{dec}}}{U_{max}} & u_{qf_{dec}} &= \frac{u_{q_{dec}}}{U_{max}} \\ u_{df} &= \frac{u_d}{U_{max}} & u_{qf} &= \frac{u_q}{U_{max}} \\ i_{df} &= \frac{i_d}{I_{max}} & i_{qf} &= \frac{i_q}{I_{max}} \\ \omega_{ef} &= \frac{\omega_e}{\Omega_{max}} \end{aligned}$$

Equation **GMCLIB_DecouplingPMSM_Eq6**

Further, rearranging GMCLIB_DecouplingPMSM_Eq5 results in:

$$u_{\mathrm{df}_{\mathrm{dec}}} = u_{\mathrm{df}} - \omega_{\mathrm{ef}} \bullet i_{\mathrm{qf}} \frac{L_q \bullet \Omega_{\max} \bullet I_{\max}}{U_{\max}} = u_{\mathrm{df}} - \omega_{\mathrm{ef}} \bullet i_{\mathrm{qf}} \bullet k_d$$
$$u\mathrm{qf}_{\mathrm{dec}} = u_{\mathrm{qf}} - \omega_{\mathrm{ef}} \bullet i_{\mathrm{df}} \frac{L_d \bullet \Omega_{\max} \bullet I_{\max}}{U_{\max}} = u_{\mathrm{qf}} - \omega_{\mathrm{ef}} \bullet i_{\mathrm{df}} \bullet k_q$$

Equation **GMCLIB_DecouplingPMSM_Eq7**

where k $_d$ and k $_q$ are coefficients calculated as:

$$k_d = L_q \bullet \Omega_{\max} \bullet \frac{I_{\max}}{U_{\max}}$$
$$k_q = L_d \bullet \Omega_{\max} \bullet \frac{I_{\max}}{U_{\max}}$$

Equation **GMCLIB_DecouplingPMSM_Eq8**

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:

$$k_{d\_\mathrm{shift}} = \mathrm{ceil}\left(\frac{\log(k_d)}{\log(2)}\right)$$
$$k_{q\_\mathrm{shift}} = \mathrm{ceil}\left(\frac{\log(k_q)}{\log(2)}\right)$$

Equation **GMCLIB_DecouplingPMSM_Eq9**

Using scaling coefficients GMCLIB_DecouplingPMSM_Eq9, the fractional representation of coefficients k $_d$ and k $_q$ from GMCLIB_DecouplingPMSM_Eq8 are derived as follows:

$$k_{\mathrm{df}} = k_d \bullet 2^{-k_{d\_\mathrm{shift}}}$$
$$k_{qf} = k_q \bullet 2^{-k_{q\_\mathrm{shift}}}$$

Equation **GMCLIB_DecouplingPMSM_Eq10**

Substituting GMCLIB_DecouplingPMSM_Eq8 - GMCLIB_DecouplingPMSM_Eq10 into GMCLIB_DecouplingPMSM_Eq7 results in the final form of the equation set, actually implemented in the GMCLIB_DecouplingPMSM_F16 function:

$$u_{df_{\text{dec}}} = u_{\text{df}} - \omega_{\text{ef}} \bullet i_{\text{qf}} \bullet k_{\text{df}} \bullet 2^{k_{d\_\text{shift}}}$$

$$u_{qf_{\text{dec}}} = u_{\text{qf}} + \omega_{\text{ef}} \bullet i_{\text{df}} \bullet k_{\text{qf}} \bullet 2^{k_{q\_\text{shift}}}$$

Equation **GMCLIB_DecouplingPMSM_Eq11**

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.

5.41.4.4 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]
  //output should be 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]
  //output should be 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]
                 //output should be f16trUDecDQ.f16Arg2 ~ 0x6666 ~ FRAC16(0.8)*50V ~=40[V]
                 GMCLIB_DecouplingPMSM (&f16trUDecDQ,&f16trUDQ,&f16trIDQ,f16We,&f16trDec);
               }
```

5.41.5 Function GMCLIB_DecouplingPMSM_FLT

5.41.5.1 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);
```

5.41.5.2 Arguments

Table 5-137. 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 (floating-point overflow, underflow, inexact, invalid operation).

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.

5.41.5.3 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;
  // 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]
  // output should be flttrUDecDQ.fltArg2 ~= 40[V]
  GMCLIB_DecouplingPMSM_FLT
(&flttrUDecDQ,&flttrUDQ,&flttrIDQ,fltWe,&flttrDec);

  // output should be flttrUDecDQ.fltArg1 ~= -35[V]
  // output should be 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]
  // output should be flttrUDecDQ.fltArg2 ~= 40[V]
  GMCLIB_DecouplingPMSM (&flttrUDecDQ,&flttrUDQ,&flttrIDQ,fltWe,&flttrDec);
}
```

5.42 Function GMCLIB_ElimDcBusRip

This function implements the DC Bus voltage ripple elimination.

5.42.1 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 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 an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.42.2 Re-entrancy

The function is re-entrant.

5.42.3 Function GMCLIB_ElimDcBusRip_F32

5.42.3.1 Declaration

```
void GMCLIB_ElimDcBusRip_F32(SWLIBS_2Syst_F32 *const pOut, const SWLIBS_2Syst_F32 *const pIn,
const GMCLIB_ELIMDCBUSRIP_T_F32 *const pParam);
```

5.42.3.2 Arguments

Table 5-138. 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.

5.42.3.3 Return

Function returns no value.

5.42.3.4 Variant Specifics

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:

$$U_{req\_new} = \frac{U_{req} \cdot U_{DC\_BUS\_MAX}}{U_{DC\_BUS\_ACTUAL}} = 50V$$

Equation **GMCLIB_ElimDcBusRip_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:

$$U_{req\_new} = \frac{U_{req} \cdot U_{DC\_BUS\_MAX}}{U_{DC\_BUS\_ACTUAL}} = 55.5V$$

Equation **GMCLIB_ElimDcBusRip_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:

- for the α-component:

$$u_{\alpha}^{*} = \begin{cases} \frac{f32ModIndex \bullet u_{\alpha}}{f32ArgDcBusMsr/2} & \text{if} \quad abs\left(f32ModIndex \bullet u_{\alpha}\right) < \frac{f32ArgDcBusMsr}{2} \\ sign\left(u_{\alpha}\right) & \text{otherwise} \end{cases}$$

Equation **GMCLIB_ElimDcBusRip_Eq3**

- for the β-component:

$$u_{\beta}^{*} = \begin{cases} \frac{f32ModIndex \bullet u_{\beta}}{f32ArgDcBusMsr/2} & \text{if} \quad abs\left(f32ModIndex \bullet u_{\beta}\right) < \frac{f32ArgDcBusMsr}{2} \\ sign\left(u_{\beta}\right) & \text{otherwise} \end{cases}$$

Equation **GMCLIB_ElimDcBusRip_Eq4**

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.

It should be noted that although the modulation index (see the parameters structure, the f32ModIndex member) is assumed to be equal to or greater than zero, the possible values are restricted to those values resulting from the use of Space Vector Modulation techniques.

In order to correctly handle the discontinuity at f32ArgDcBusMsr approaching 0, and for efficiency reasons, the function will assign 0 to the output duty cycle ratios if the f32ArgDcBusMsr is below the threshold of 2^{-15}. In other words, the 16 least significant bits of the f32DcBusMsr are ignored. Also, the computed output of the u_{α}* and u_{β}* components may have an inaccuracy in the 16 least significant bits.

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.

5.42.3.5 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);
}
```

5.42.4 Function GMCLIB_ElimDcBusRip_F16

5.42.4.1 Declaration

```
void GMCLIB_ElimDcBusRip_F16(SWLIBS_2Syst_F16 *const pOut, const SWLIBS_2Syst_F16 *const pIn,
const GMCLIB_ELIMDCBUSRIP_T_F16 *const pParam);
```

5.42.4.2 Arguments

Table 5-139. 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.

5.42.4.3 Return

Function returns no value.

5.42.4.4 Variant Specifics

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:

$$U_{req\_new} = \frac{U_{req} \cdot U_{DC\_BUS\_MAX}}{U_{DC\_BUS\_ACTUAL}} = 50V$$

Equation **GMCLIB_ElimDcBusRip_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]
- voltage to be applied to the PWM modulator to generate U_{reg}=50[V] on the inverter phase output:

$$U_{\mathrm{req\_new}} = \frac{U_{\mathrm{req}} \bullet U_{\mathrm{DC\_BUS\_MAX}}}{U_{\mathrm{DC\_BUS\_ACTUAL}}} = 55.5V$$

Equation **GMCLIB_ElimDcBusRip_Eq2**

$$u_{\alpha}^{*} = \begin{cases} \frac{f16\mathrm{ModIndex} \bullet u_{\alpha}}{f16\mathrm{ArgDcBusMsr}/2} & \text{if} \quad \mathrm{abs}\left(f16\mathrm{ModIndex} \bullet u_{\alpha}\right) < \frac{f16\mathrm{ArgDcBusMsr}}{2} \\ \mathrm{sign}\left(u_{\alpha}\right) & \text{otherwise} \end{cases}$$

Equation **GMCLIB_ElimDcBusRip_Eq3**

- for the β-component:

$$u_{\beta}^{*} = \begin{cases} \frac{f16\mathrm{ModIndex} \bullet u_{\beta}}{f16\mathrm{ArgDcBusMsr}/2} & \text{if} \quad \mathrm{abs}\left(f16\mathrm{ModIndex} \bullet u_{\beta}\right) < \frac{f16\mathrm{ArgDcBusMsr}}{2} \\ \mathrm{sign}\left(u_{\beta}\right) & \text{otherwise} \end{cases}$$

Equation **GMCLIB_ElimDcBusRip_Eq4**

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.

It should be noted that although the modulation index (see the parameters structure, the f16ModIndex member) is assumed to be equal to or greater than zero, the possible values are restricted to those values resulting from the use of Space Vector Modulation techniques.

In order to correctly handle the discontinuity at f16ArgDcBusMsr approaching 0, and for efficiency reasons, the function will assign 0 to the output duty cycle ratios if the f16ArgDcBusMsr is below the threshold of 2^{-15}. In other words, the 16 least significant bits of the f16DcBusMsr are ignored. Also, the computed output of the u_{α}* and u_{β}* components may have an inaccuracy in the 16 least significant bits.

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.

5.42.4.5 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
  // 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);
                }
```

5.42.5 Function GMCLIB_ElimDcBusRip_FLT

5.42.5.1 Declaration

```
void GMCLIB_ElimDcBusRip_FLT(SWLIBS_2Syst_FLT *const pOut, const SWLIBS_2Syst_FLT *const pIn,
const GMCLIB_ELIMDCBUSRIP_T_FLT *const pParam);
```

5.42.5.2 Arguments

Table 5-140. 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.
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.
const GMCLIB_ELIMDCBUSRIP_T_FLT *const	pParam	**input**	Pointer to the parameters structure.

5.42.5.3 Return

Function returns no value.

5.42.5.4 Variant Specifics

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:

$$U_{\text{req\_new}} = \frac{U_{\text{req}} \bullet U_{\text{DC\_BUS\_MAX}}}{U_{\text{DC\_BUS\_ACTUAL}}} = 50V$$

Equation **GMCLIB_ElimDcBusRip_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:

$$U_{\text{req\_new}} = \frac{U_{\text{req}} \bullet U_{\text{DC\_BUS\_MAX}}}{U_{\text{DC\_BUS\_ACTUAL}}} = 55.5V$$

Equation **GMCLIB_ElimDcBusRip_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:

- for the α-component:

$$u_{\alpha}^{*} = \begin{cases} \frac{\text{fltModIndex} \bullet u_{\alpha}}{\text{fltArgDcBusMsr}/2} & \text{if} \quad \text{abs}\left(\text{fltModIndex} \bullet u_{\alpha}\right) < \frac{\text{fltArgDcBusMsr}}{2} \\ \text{sign}\left(u_{\alpha}\right) & \text{otherwise} \end{cases}$$

Equation **GMCLIB_ElimDcBusRip_Eq3**

- for the β-component:

$$u_{\beta}^{*} = \begin{cases} \frac{\text{fltModIndex} \bullet u_{\beta}}{\text{fltArgDcBusMsr}/2} & \text{if} \quad \text{abs}\left(\text{fltModIndex} \bullet u_{\beta}\right) < \frac{\text{fltArgDcBusMsr}}{2} \\ \text{sign}\left(u_{\beta}\right) & \text{otherwise} \end{cases}$$

Equation **GMCLIB_ElimDcBusRip_Eq4**

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.

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.

It should be noted that although the modulation index (see the parameters structure pParam, the fltModIndex member) is assumed to be equal to or greater than zero, the possible values are restricted to those values resulting from the use of Space Vector Modulation techniques.

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation, zero divide).

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.

5.42.5.5 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
```

```
        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);
      }
```

5.43 Function GMCLIB_Park

This function implements the calculation of Park transformation.

5.43.1 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:

$$d = \cos(\theta_e) \bullet \alpha + \sin(\theta_e) \bullet \beta$$

Equation **GMCLIB_Park_Eq1**

$$q = -\sin(\theta_e) \bullet \alpha + \cos(\theta_e) \bullet \beta$$

Equation **GMCLIB_Park_Eq2**

where θ_e represents the electrical position of the rotor flux.

Note

The input/output pointers must contain valid addresses, otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.43.2 Re-entrancy

The function is re-entrant.

5.43.3 Function GMCLIB_Park_F32

5.43.3.1 Declaration

```
void GMCLIB_Park_F32(SWLIBS_2Syst_F32 *pOut, const SWLIBS_2Syst_F32 *const pInAngle, const
SWLIBS_2Syst_F32 *const pIn);
```

5.43.3.2 Arguments

Table 5-141. 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.
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α- β).

Note

The inputs and the outputs are normalized to fit in the range [-1, 1).

5.43.3.3 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);
}
```

5.43.4 Function GMCLIB_Park_F16

5.43.4.1 Declaration

```
void GMCLIB_Park_F16(SWLIBS_2Syst_F16 *pOut, const SWLIBS_2Syst_F16 *const pInAngle, const
SWLIBS_2Syst_F16 *const pIn);
```

5.43.4.2 Arguments

Table 5-142. 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

Due to effectivity reasons this function is implemented using intrinsic functions and is therefore not ANSI-C compliant. Note that some compilers do not support the enhanced features.

The inputs and the outputs are normalized to fit in the range [-1, 1).

5.43.4.3 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
  // ##############################################################

  // output should be
  // tr16Dq.f16Arg1 ~ d = 0x505E
  // tr16Dq.f16Arg2 ~ q = 0x1C38
  GMCLIB_Park (&tr16Dq,&tr16Angle,&tr16AlBe);
}
```

5.43.5 Function GMCLIB_Park_FLT

5.43.5.1 Declaration

void GMCLIB_Park_FLT(SWLIBS_2Syst_FLT *pOut, const SWLIBS_2Syst_FLT *const pInAngle, const SWLIBS_2Syst_FLT *const pIn);

5.43.5.2 Arguments

Table 5-143. 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 (floating-point overflow, underflow, inexact, invalid operation).

5.43.5.3 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);

  // 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);
}
```

5.44 Function GMCLIB_ParkInv

This function implements the inverse Park transformation.

5.44.1 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:

$$\alpha = \cos(\theta_e) \bullet d - \sin(\theta_e) \bullet q$$

Equation **GMCLIB_ParkInv_Eq1**

$$\beta = \sin(\theta_e) \bullet d + \cos(\theta_e) \bullet q$$

Equation **GMCLIB_ParkInv_Eq2**

where θ_e represents the electrical position of the rotor flux.

Note

The input/output pointers must contain valid addresses, otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.44.2 Re-entrancy

The function is re-entrant.

5.44.3 Function GMCLIB_ParkInv_F32

5.44.3.1 Declaration

```
void GMCLIB_ParkInv_F32(SWLIBS_2Syst_F32 *const pOut, const SWLIBS_2Syst_F32 *const pInAngle,
const SWLIBS_2Syst_F32 *const pIn);
```

5.44.3.2 Arguments

Table 5-144. 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).
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).

5.44.3.3 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);
}
```

5.44.4 Function GMCLIB_ParkInv_F16

5.44.4.1 Declaration

`void GMCLIB_ParkInv_F16(SWLIBS_2Syst_F16 *const pOut, const SWLIBS_2Syst_F16 *const pInAngle, const SWLIBS_2Syst_F16 *const pIn);`

5.44.4.2 Arguments

Table 5-145. 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).

5.44.4.3 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);
}
```

5.44.5 Function GMCLIB_ParkInv_FLT

5.44.5.1 Declaration

```
void GMCLIB_ParkInv_FLT(SWLIBS_2Syst_FLT *const pOut, const SWLIBS_2Syst_FLT *const pInAngle,
const SWLIBS_2Syst_FLT *const pIn);
```

5.44.5.2 Arguments

Table 5-146. 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 (floating-point overflow, underflow, inexact, invalid operation).

The inputs and the outputs are in single precision floating point data format.

5.44.5.3 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
  // implementation selected as default
  // ##############################################################

  // output should be:
  // trfltAlBe.fltArg1 ~ alpha = -0.495119387
  // trfltAlBe.fltArg2 ~ beta  = 0.433521139
  GMCLIB_ParkInv (&trfltAlBe,&trfltAngle,&trfltDq);
}
```

5.45 Function GMCLIB_SvmStd

This function calculates the duty-cycle ratios using the Standard Space Vector Modulation technique.

5.45.1 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 5-33.

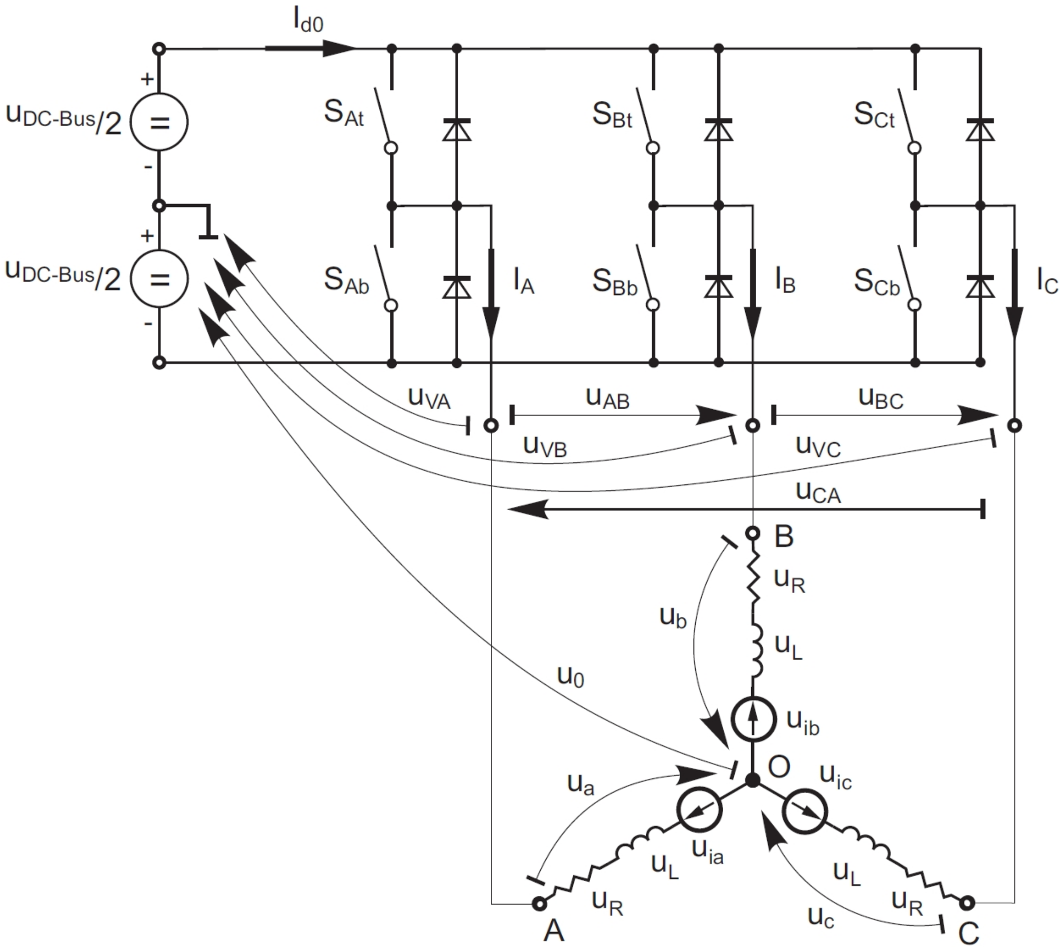

Figure 5-33. Power stage schematic diagram

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 5-33), eight possible switching states (detailed in Figure 5-34) are feasible.

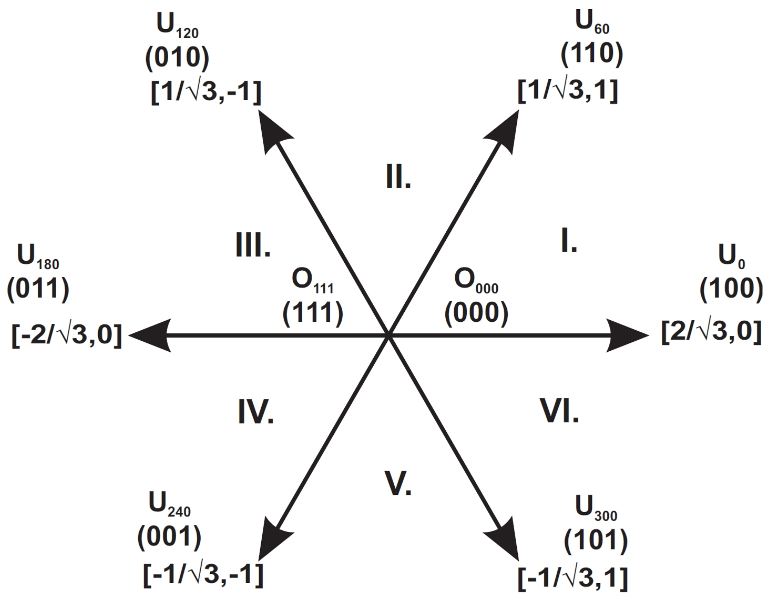

Figure 5-34. Basic space vectors

These states, together with the resulting instantaneous output line-to-line and phase voltages, are listed in Table 5-147.

Table 5-147. 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.

$$U_\alpha = \frac{2}{3}\left(U_a - \frac{U_b}{2} - \frac{U_c}{2}\right)$$

Equation **GMCLIB_SvmStd_Eq1**

$$U_\beta = \frac{2}{3}\left(0 + \frac{\sqrt{3}\,U_b}{2} - \frac{\sqrt{3}\,U_C}{2}\right)$$

Equation **GMCLIB_SvmStd_Eq2**

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 5-148.

Table 5-148. 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 5-34 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.

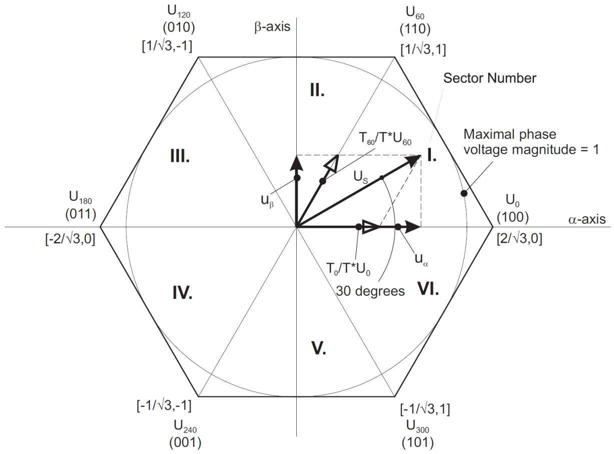

Figure 5-35. Projection of reference voltage vector in sector I

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 5-35 and Figure 5-36.

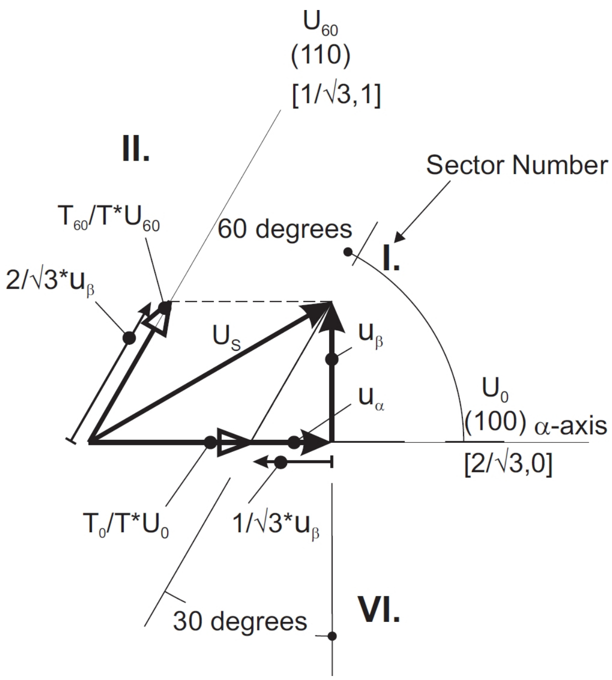

Figure 5-36. Detail of the voltage vector projection in sector I

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:

$$T = T_{60} + T_0 + T_{null}$$

Equation **GMCLIB_SvmStd_Eq3**

$$U_S = \frac{T_{60}}{T} U_{60} + \frac{T_0}{T} U_0$$

Equation **GMCLIB_SvmStd_Eq4**

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:

$$u_\beta = \frac{T_{60}}{\mathrm{T}} \,|\, U_{60} \,|\, \sin\ 60^\circ$$

Equation **GMCLIB_SvmStd_Eq5**

$$u_\alpha = \frac{T_0}{T} \,|\, U_0 \,|\, + \frac{u_\beta}{\tan 60^\circ}$$

Equation **GMCLIB_SvmStd_Eq6**

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, equation GMCLIB_SvmStd_Eq5 and equation GMCLIB_SvmStd_Eq6 can be rearranged for the unknown duty-cycle ratios T_{60}/T and T_0/T:

$$\frac{T_{60}}{T} = u_\beta$$

Equation **GMCLIB_SvmStd_Eq7**

$$\frac{T_0}{T} = \frac{1}{2}\left(\sqrt{3}\,u_\alpha - u_\beta\right)$$

Equation **GMCLIB_SvmStd_Eq8**

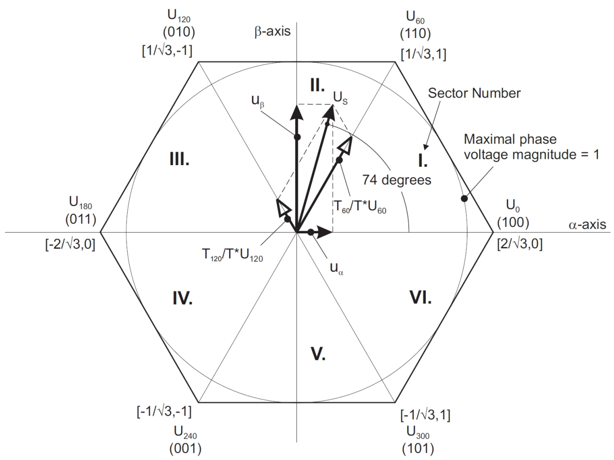

Figure 5-37. Projection of the reference voltage vector in sector II

Sector II is depicted in Figure 5-37. 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:

$$T = T_{120} + T_{60} + T_{\text{null}}$$

Equation **GMCLIB_SvmStd_Eq9**

$$U_S = \frac{T_{120}}{T} U_{120} + \frac{T_{60}}{T} U_{60}$$

Equation **GMCLIB_SvmStd_Eq10**

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 are formed from the auxiliary components termed A and B. The graphical representation of the auxiliary components is shown in Figure 5-38.

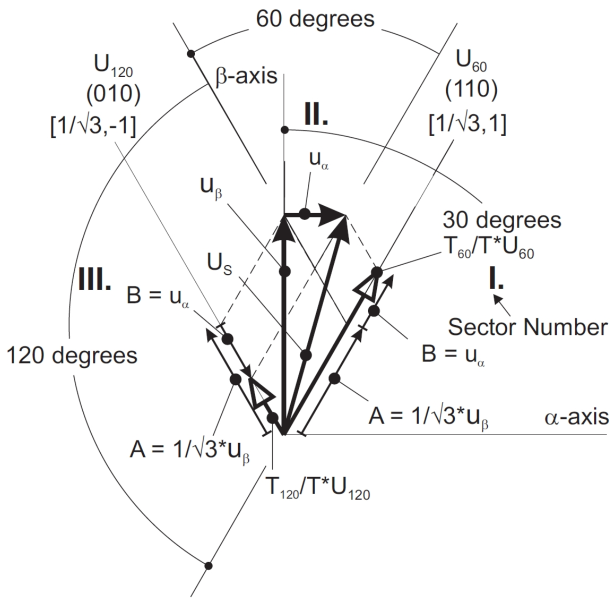

Figure 5-38. Detail of the voltage vector projection in sector II

The equations describing those auxiliary time-duration components are:

$$\frac{\sin 30°}{\sin 120°} = \frac{A}{u_\alpha}$$

Equation **GMCLIB_SvmStd_Eq11**

$$\frac{\sin 60°}{\sin 120°} = \frac{B}{u_\alpha}$$

Equation **GMCLIB_SvmStd_Eq12**

Equation GMCLIB_SvmStd_Eq11 and equation GMCLIB_SvmStd_Eq12 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.

$$A = \frac{1}{\sqrt{3}} u_\beta$$

Equation **GMCLIB_SvmStd_Eq13**

$$B = u_\alpha$$

Equation **GMCLIB_SvmStd_Eq14**

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 equation GMCLIB_SvmStd_Eq13 and equation GMCLIB_SvmStd_Eq14, as follows:

$$\frac{T_{120}}{T} |U_{120}| = A - B$$

Equation **GMCLIB_SvmStd_Eq15**

$$\frac{T_{60}}{T} |U_{60}| = A + B$$

Equation **GMCLIB_SvmStd_Eq16**

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:

$$\frac{T_{120}}{T} = \frac{1}{2}\left(u_\beta - \sqrt{3}\, u_\alpha\right)$$

Equation **GMCLIB_SvmStd_Eq17**

$$\frac{T_{60}}{T} = \frac{1}{2}\left(u_\beta + \sqrt{3}\, u_\alpha\right)$$

Equation **GMCLIB_SvmStd_Eq18**

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:

$$X = u_\beta$$

Equation **GMCLIB_SvmStd_Eq19**

$$Y = \frac{1}{2}\left(u_\beta + \sqrt{3}\,u_\alpha\right)$$

Equation **GMCLIB_SvmStd_Eq20**

$$Z = \frac{1}{2}\left(u_\beta - \sqrt{3}\,u_\alpha\right)$$

Equation **GMCLIB_SvmStd_Eq21**

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 5-149.

Table 5-149. 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 equation GMCLIB_SvmStd_Eq19, Y equation GMCLIB_SvmStd_Eq20 and Z equation GMCLIB_SvmStd_Eq21, 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.

$$u_{\mathrm{ref1}} = u_\beta$$

Equation **GMCLIB_SvmStd_Eq22**

$$u_{\mathrm{ref2}} = \frac{1}{2}\left(-u_{\beta} + \sqrt{3}\,u_{\alpha}\right)$$

Equation **GMCLIB_SvmStd_Eq23**

$$u_{\mathrm{ref3}} = \frac{1}{2}\left(-u_{\beta} - \sqrt{3}\,u_{\alpha}\right)$$

Equation **GMCLIB_SvmStd_Eq24**

The modified Inverse Clark Transformation projects the quadrature-u_{β} component into u_{ref1}, as shown in Figure 5-39 and Figure 5-40, whereas voltages generated by the conventional Inverse Clark Transformation project the u_{α} component into u_{ref1}.

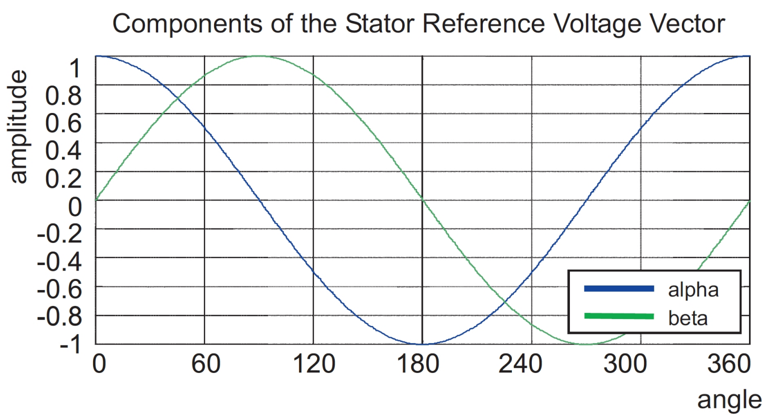

Figure 5-39. Direct-ua and quadrature-ub components of stator reference voltage

Figure 5-39 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.

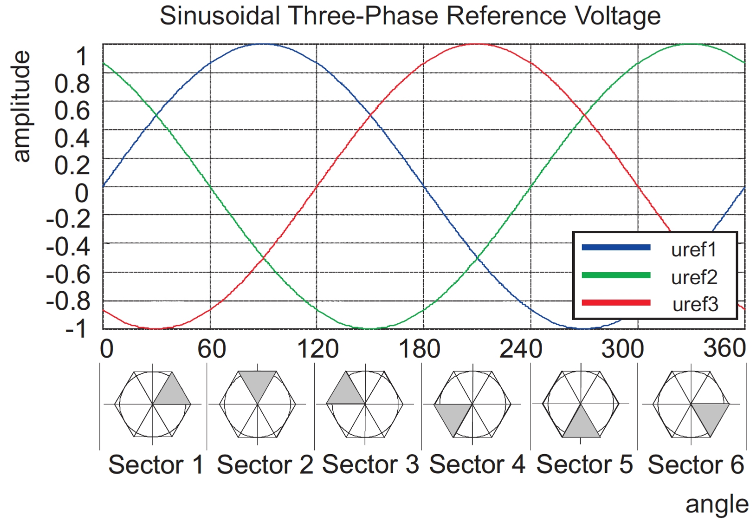

Figure 5-40. Reference Voltages u_ref1, u_ref2 and u_ref3

The Sector Identification Tree, shown in Figure 5-41, can be a numerical solution of the approach shown in Figure 5-40.

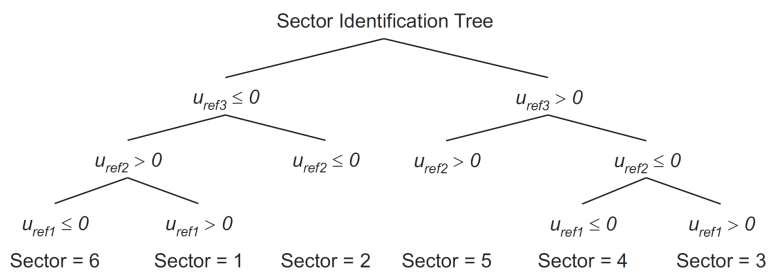

Figure 5-41. Identification of the sector number

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 5-35, 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 5-40. 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 5-38. 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:

$$t_1 = \frac{T - t\_1 - t\_2}{2}$$

Equation **GMCLIB_SvmStd_Eq25**

$$t_2 = t_1 + t\_1$$

Equation **GMCLIB_SvmStd_Eq26**

$$t_3 = t_2 + t\_2$$

Equation **GMCLIB_SvmStd_Eq27**

where T is the switching period, t_1 and t_2 are the duty-cycle ratios (see Table 5-149) of the basic space vectors, given for the respective sector. Equation GMCLIB_SvmStd_Eq25, equation GMCLIB_SvmStd_Eq26 and equation GMCLIB_SvmStd_Eq27 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 5-150.

Table 5-150. 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 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 5-42

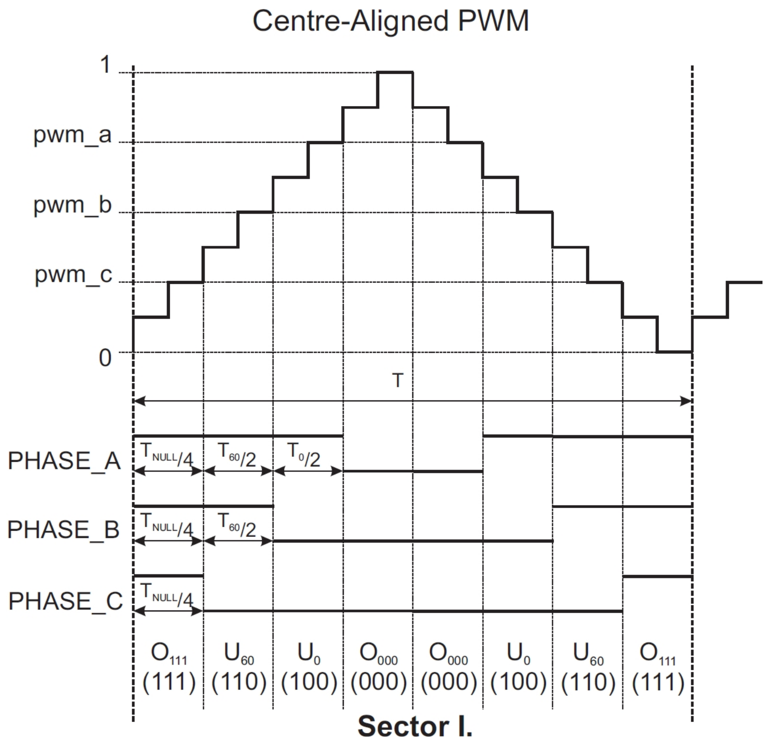

Figure 5-42. Standard space vector modulation technique - centre-aligned PWM

Note

The input/output pointers must contain valid addresses, otherwise an exception may occur (Data TLB Error, Data Storage, Alignment, Machine Check).

5.45.2 Re-entrancy

The function is re-entrant.

5.45.3 Function GMCLIB_SvmStd_F32

5.45.3.1 Declaration

```
tU32 GMCLIB_SvmStd_F32(SWLIBS_3Syst_F32 *pOut, const SWLIBS_2Syst_F32 *const pIn);
```

5.45.3.2 Arguments

Table 5-151. 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.

5.45.3.3 Return

The function returns a 32-bit value in format INT, representing the actual space sector which contains the stator reference vector U_s.

5.45.3.4 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);
}
```

5.45.4 Function GMCLIB_SvmStd_F16

5.45.4.1 Declaration

```
tU16 GMCLIB_SvmStd_F16(SWLIBS_3Syst_F16 *pOut, const SWLIBS_2Syst_F16 *const pIn);
```

5.45.4.2 Arguments

Table 5-152. 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.

5.45.4.3 Return

The function returns a 16-bit value in format INT, representing the actual space sector which contains the stator reference vector U_S.

5.45.4.4 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);
  }
```

5.45.5 Function GMCLIB_SvmStd_FLT

5.45.5.1 Declaration

```
tU32 GMCLIB_SvmStd_FLT(SWLIBS_3Syst_FLT *pOut, const SWLIBS_2Syst_FLT *const pIn);
```

5.45.5.2 Arguments

Table 5-153. 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.

5.45.5.3 Return

The function returns a 32-bit value in format INT, representing the actual space sector which contains the stator reference vector U_{s}.

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

5.45.5.4 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
  // 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);
}
```

5.46 Function MLIB_Abs

This function returns absolute value of input parameter.

5.46.1 Description

This inline function returns the absolute value of input parameter.

5.46.2 Re-entrancy

The function is re-entrant.

5.46.3 Function MLIB_Abs_F32

5.46.3.1 Declaration

```
INLINE tFrac32 MLIB_Abs_F32(register tFrac32 f32In);
```

5.46.3.2 Arguments

Table 5-154. MLIB_Abs_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value.

5.46.3.3 Return

Absolute value of input parameter.

5.46.3.4 Variant Specifics

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:

$$f32\text{Out} = \begin{cases} f32\text{In} & \text{if} \quad f32\text{In} \geq 0 \\ (-f32\text{In}) & \text{if} \quad f32\text{In} < 0 \end{cases}$$

Equation **MLIB_Abs_Eq1**

Note

Due to effectivity reason this function is written as inline assembly, and thus is not ANSI-C compliant.

5.46.3.5 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);
}
```

5.46.4 Function MLIB_Abs_F16

5.46.4.1 Declaration

```
INLINE tFrac16 MLIB_Abs_F16(register tFrac16 f16In);
```

5.46.4.2 Arguments

Table 5-155. MLIB_Abs_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	Input value.

5.46.4.3 Return

Absolute value of input parameter.

5.46.4.4 Variant Specifics

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:

$$f16Out = \begin{cases} f16In & \text{if } f16In \geq 0 \\ (-f16In) & \text{if } f16In < 0 \end{cases}$$

Equation **MLIB_Abs_Eq1**

Note

Due to effectivity reason this function is written as inline assembly, and thus is not ANSI-C compliant.

5.46.4.5 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);
}
```

5.46.5 Function MLIB_Abs_FLT

5.46.5.1 Declaration

```
INLINE tFloat MLIB_Abs_FLT(register tFloat fltIn);
```

5.46.5.2 Arguments

Table 5-156. MLIB_Abs_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn	**input**	Input value.

5.46.5.3 Return

Absolute value of input parameter.

5.46.5.4 Variant Specifics

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:

$$fltOut = \begin{cases} fltIn & \text{if} \quad fltIn \geq 0 \\ (-fltIn) & \text{if} \quad fltIn < 0 \end{cases}$$

Equation **MLIB_Abs_Eq1**

Note

The function may raise floating-point exceptions (floating-point inexact, invalid operation).

Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

5.46.5.5 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);
}
```

5.47 Function MLIB_AbsSat

This function returns absolute value of input parameter and saturate if necessary.

5.47.1 Description

This inline function returns the absolute value of input parameter and saturates if necessary.

5.47.2 Re-entrancy

The function is re-entrant.

5.47.3 Function MLIB_AbsSat_F32

5.47.3.1 Declaration

```
INLINE tFrac32 MLIB_AbsSat_F32(register tFrac32 f32In);
```

5.47.3.2 Arguments

Table 5-157. MLIB_AbsSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value.

5.47.3.3 Return

Absolute value of input parameter, saturated if necessary.

5.47.3.4 Variant Specifics

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:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad |f32In| < FRAC32\_MIN \\ |f32In| & \text{if} \quad FRAC32\_MIN \le |f32In| \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad |f32In| > FRAC32\_MAX \end{cases}$$

Equation **MLIB_AbsSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

5.47.3.5 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_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);
}
```

5.47.4 Function MLIB_AbsSat_F16

5.47.4.1 Declaration

```
INLINE tFrac16 MLIB_AbsSat_F16(register tFrac16 f16In);
```

5.47.4.2 Arguments

Table 5-158. MLIB_AbsSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	Input value.

5.47.4.3 Return

Absolute value of input parameter, saturated if necessary.

5.47.4.4 Variant Specifics

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:

$$f16Out = \begin{cases} FRAC16\_MIN & \text{if} \quad |f16In| < FRAC16\_MIN \\ |f16In| & \text{if} \quad FRAC16\_MIN \le |f16In| \le FRAC16\_MAX \\ FRAC16\_MAX & \text{if} \quad |f16In| > FRAC16\_MAX \end{cases}$$

Equation **MLIB_AbsSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

5.47.4.5 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_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);
}
```

5.48 Function MLIB_Add

This function returns sum of two input parameters.

5.48.1 Description

This inline function returns the sum of two input values.

5.48.2 Re-entrancy

The function is re-entrant.

5.48.3 Function MLIB_Add_F32

5.48.3.1 Declaration

```
INLINE tFrac32 MLIB_Add_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

5.48.3.2 Arguments

Table 5-159. 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.

5.48.3.3 Return

Sum of two input values.

5.48.3.4 Variant Specifics

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:

$$f32Out = \left(f32In1 + f32In2\right)$$

Equation **MLIB_Add_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.48.3.5 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);
}
```

5.48.4 Function MLIB_Add_F16

5.48.4.1 Declaration

```
INLINE tFrac16 MLIB_Add_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.48.4.2 Arguments

Table 5-160. 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.

5.48.4.3 Return

Sum of two input values.

5.48.4.4 Variant Specifics

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.

The output of the function is defined by the following simple equation:

$$f16Out = \left(f16In1 + f16In2\right)$$

Equation **MLIB_Add_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.48.4.5 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);
}
```

5.48.5 Function MLIB_Add_FLT

5.48.5.1 Declaration

```
INLINE tFloat MLIB_Add_FLT(register tFloat fltIn1, register tFloat fltIn2);
```

5.48.5.2 Arguments

Table 5-161. 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.

5.48.5.3 Return

Sum of two input values.

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.48.5.4 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);
}
```

5.49 Function MLIB_AddSat

This function returns sum of two input parameters and saturate if necessary.

5.49.1 Description

This inline function returns the sum of two input values and saturates the result if necessary.

5.49.2 Re-entrancy

The function is re-entrant.

5.49.3 Function MLIB_AddSat_F32

5.49.3.1 Declaration

```
INLINE tFrac32 MLIB_AddSat_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

5.49.3.2 Arguments

Table 5-162. 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.

5.49.3.3 Return

Sum of two input values, saturated if necessary.

5.49.3.4 Variant Specifics

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:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad (f32In1 + f32In2) < FRAC32\_MIN \\ (f32In1 + f32In2) & \text{if} \quad FRAC32\_MIN \leq (f32In1 + f32In2) \leq FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad (f32In1 + f32In2) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_AddSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

5.49.3.5 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);
}
```

5.49.4 Function MLIB_AddSat_F16

5.49.4.1 Declaration

```
INLINE tFrac16 MLIB_AddSat_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.49.4.2 Arguments

Table 5-163. 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.

5.49.4.3 Return

Sum of two input values, saturated if necessary.

5.49.4.4 Variant Specifics

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:

$$f16Out = \begin{cases} FRAC16\_MIN & \text{if} \quad (f16In1 + f16In2) < FRAC16\_MIN \\ (f16In1 + f16In2) & \text{if} \quad FRAC16\_MIN \leq (f16In1 + f16In2) \leq FRAC16\_MAX \\ FRAC16\_MAX & \text{if} \quad (f16In1 + f16In2) > FRAC16\_MAX \end{cases}$$

Equation **MLIB_AddSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

5.49.4.5 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
  // ##############################################################

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_AddSat (f16In1, f16In2);
}
```

5.50 Function MLIB_Convert

This function converts the input value to different representation with scale.

5.50.1 Description

This inline function converts the input value to a different data type. The second input argument represents the scale factor.

5.50.2 Re-entrancy

The function is re-entrant.

5.50.3 Function MLIB_Convert_F32F16

5.50.3.1 Declaration

```
INLINE tFrac32 MLIB_Convert_F32F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.50.3.2 Arguments

Table 5-164. 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.

5.50.3.3 Return

Converted input value in 32-bit fractional format.

5.50.3.4 Variant Specifics

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:

$$f32Out = \begin{cases} (tFrac32)\frac{f16In1}{|f16In2|} & \text{if} \ (f16In2) < (tFrac16)0 \\ (tFrac32)(f16In1 \bullet f16In2) & \text{if} \ (f16In2) \geq (tFrac16)0 \end{cases}$$

Equation **MLIB_Convert_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.50.3.5 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);
}
```

5.50.4 Function MLIB_Convert_F32FLT

5.50.4.1 Declaration

```
INLINE tFrac32 MLIB_Convert_F32FLT(register tFloat fltIn1, register tFloat fltIn2);
```

5.50.4.2 Arguments

Table 5-165. 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.

5.50.4.3 Return

Converted input value in 32-bit fractional format.

5.50.4.4 Variant Specifics

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 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:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad (fltIn1 \bullet fltIn2) < FRAC32\_MIN \\ \frac{fltIn1 \bullet fltIn2}{(tFloat)INT32\_MAX+1} & \text{if} \quad FRAC32\_MIN \leq (fltIn1 \bullet fltIn2) \leq FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad (fltIn1 \bullet fltIn2) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_Convert_Eq1**

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.50.4.5 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);
    }
```

5.50.5 Function MLIB_Convert_F16F32

5.50.5.1 Declaration

```
INLINE tFrac16 MLIB_Convert_F16F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

5.50.5.2 Arguments

Table 5-166. 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.

5.50.5.3 Return

Converted input value in 16-bit fractional format.

5.50.5.4 Variant Specifics

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:

$$f16Out = \begin{cases} (\mathrm{tFrac16})\frac{f32In1}{|f32In2|} & \text{if } (f32In2) < (\mathrm{tFrac32})0 \\ (\mathrm{tFrac16})(f32In1 \bullet f32In2) & \text{if } (f32In2) \geq (\mathrm{tFrac32})0 \end{cases}$$

Equation **MLIB_Convert_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.50.5.5 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
    f16Out = MLIB_Convert_F16F32 (f32In1, f32In2);

    // output should be FRAC16(0.5) = 0x4000
    f16Out = MLIB_Convert (f32In1, f32In2, F16F32);
}
```

5.50.6 Function MLIB_Convert_F16FLT

5.50.6.1 Declaration

```
INLINE tFrac16 MLIB_Convert_F16FLT(register tFloat fltIn1, register tFloat fltIn2);
```

5.50.6.2 Arguments

Table 5-167. 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.

5.50.6.3 Return

Converted input value in 16-bit fractional format.

5.50.6.4 Variant Specifics

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:

$$f16Out = \begin{cases} FRAC16\_MIN & \text{if} \quad (fltIn1 \bullet fltIn2) < FRAC16\_MIN \\ fltIn1 \bullet fltIn2 \bullet ((tFloat)INT16\_MAX + 1) & \text{if} \quad FRAC16\_MIN \leq (fltIn1 \bullet fltIn2) \leq FRAC16\_MAX \\ FRAC16\_MAX & \text{if} \quad (fltIn1 \bullet fltIn2) > FRAC16\_MAX \end{cases}$$

Equation **MLIB_Convert_Eq1**

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.50.6.5 Code Example

```
#include "mlib.h"
```

```
tFloat fltIn1, fltIn2;
tFrac16 f16Out;

void main(void)
{
    // 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);
}
```

5.50.7 Function MLIB_Convert_FLTF16

5.50.7.1 Declaration

```
INLINE tFloat MLIB_Convert_FLTF16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.50.7.2 Arguments

Table 5-168. 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.

5.50.7.3 Return

Converted input value in single precision floating point format.

5.50.7.4 Variant Specifics

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:

$$fltOut = \begin{cases} \frac{f16In1 \cdot f16In2}{(tFloat)INT16\_MAX+1} & \text{if} \quad f16In2 \geq (tFrac16)0 \\ (tFloat)1 & \text{if} \quad f16In2 < (tFrac16)0 \ \& \ f16In1 \geq (tFrac16)0 \ \& \ f16In1 \geq -f16In2 \\ (tFloat)-1 & \text{if} \quad f16In2 < (tFrac16)0 \ \& \ f16In1 < (tFrac16)0 \ \& \ f16In1 \leq f16In2 \\ \frac{(tFloat)f16In1}{(tFloat)f16In2} & \text{if} \quad \text{otherwise} \end{cases}$$

Equation **MLIB_Convert_Eq1**

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation, zero divide).

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.50.7.5 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);
    }
```

5.50.8 Function MLIB_Convert_FLTF32

5.50.8.1 Declaration

```
INLINE tFloat MLIB_Convert_FLTF32(register tFrac32 f32In1, register tFrac32 f32In2);
```

5.50.8.2 Arguments

Table 5-169. 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.

5.50.8.3 Return

Converted input value in single precision floating point format.

5.50.8.4 Variant Specifics

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:

$$\mathrm{fltOut} = \begin{cases} \frac{f32\mathrm{In1} \bullet f32\mathrm{In2}}{(\mathrm{tFloat})\mathrm{INT32\_MAX}+1} & \text{if} \quad f32\mathrm{In2} \geq (\mathrm{tFrac32})0 \\ (\mathrm{tFloat})1 & \text{if} \quad f32\mathrm{In2} < (\mathrm{tFrac32})0 \;\&\; f32\mathrm{In1} \geq (\mathrm{tFrac32})0 \;\&\; f32\mathrm{In1} \geq -f32\mathrm{In2} \\ (\mathrm{tFloat}) - 1 & \text{if} \quad f32\mathrm{In2} < (\mathrm{tFrac32})0 \;\&\; f32\mathrm{In1} < (\mathrm{tFrac32})0 \;\&\; f32\mathrm{In1} \leq f32\mathrm{In2} \\ \frac{(\mathrm{tFloat})f32\mathrm{In1}}{(\mathrm{tFloat})f32\mathrm{In2}} & \text{if} \quad \text{otherwise} \end{cases}$$

Equation **MLIB_Convert_Eq1**

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation, zero divide).

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.50.8.5 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);

    // 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);
}
```

5.51 Function MLIB_ConvertPU

This function converts the input value to a different data type.

5.51.1 Description

This inline function converts the input value to a different data type.

5.51.2 Re-entrancy

The function is re-entrant.

5.51.3 Function MLIB_ConvertPU_F32F16

5.51.3.1 Declaration

```
INLINE tFrac32 MLIB_ConvertPU_F32F16(register tFrac16 f16In);
```

5.51.3.2 Arguments

Table 5-170. MLIB_ConvertPU_F32F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	Input value in 16-bit fractional format to be converted.

5.51.3.3 Return

Converted input value in 32-bit fractional format.

5.51.3.4 Variant Specifics

The input value is considered as 16-bit fractional data type and output value is considered as 32-bit fractional data type. 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:

$$f32Out = (tFrac32) f16In$$

Equation **MLIB_ConvertPU_Eq1**

Note

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.51.3.5 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);
}
```

5.51.4 Function MLIB_ConvertPU_F32FLT

5.51.4.1 Declaration

```
INLINE tFrac32 MLIB_ConvertPU_F32FLT(register tFloat fltIn);
```

5.51.4.2 Arguments

Table 5-171. MLIB_ConvertPU_F32FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn	**input**	Input value in single precision floating point format to be converted.

5.51.4.3 Return

Converted input value in 32-bit fractional format.

5.51.4.4 Variant Specifics

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:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad (fltIn) < FRAC32\_MIN \\ \frac{fltIn}{(tFloat)INT32\_MAX+1} & \text{if} \quad FRAC32\_MIN \le fltIn \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad (fltIn) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_ConvertPU_Eq1**

Note

The function may raise floating-point exceptions (floating-point inexact, invalid operation).

Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

5.51.4.5 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);
}
```

5.51.5 Function MLIB_ConvertPU_F16F32

5.51.5.1 Declaration

```
INLINE tFrac16 MLIB_ConvertPU_F16F32(register tFrac32 f32In);
```

5.51.5.2 Arguments

Table 5-172. MLIB_ConvertPU_F16F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value in 32-bit fractional format to be converted.

5.51.5.3 Return

Converted input value in 16-bit fractional format.

5.51.5.4 Variant Specifics

The input value is considered as 32-bit fractional data type and output value is considered as 16-bit fractional data type. 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 without any detection.

The output of the function is defined by the following simple equation:

$$f16Out = (tFrac16)f32In$$

Equation **MLIB_ConvertPU_Eq1**

Note

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.51.5.5 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_ConvertPU_F16F32 (f32In);

        // output should be FRAC16(0.25) = 0x2000
        f16Out = MLIB_ConvertPU (f32In, F16F32);
    }
```

5.51.6 Function MLIB_ConvertPU_F16FLT

5.51.6.1 Declaration

```
INLINE tFrac16 MLIB_ConvertPU_F16FLT(register tFloat fltIn);
```

5.51.6.2 Arguments

Table 5-173. MLIB_ConvertPU_F16FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn	**input**	Input value in single precision floating point format to be converted.

5.51.6.3 Return

Converted input value in 16-bit fractional format.

5.51.6.4 Variant Specifics

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:

$$f16Out = \begin{cases} FRAC16\_MIN & \text{if} \quad (fltIn) < FRAC16\_MIN \\ \frac{fltIn}{(tFloat)INT16\_MAX+1} & \text{if} \quad FRAC16\_MIN \le fltIn \le FRAC16\_MAX \\ FRAC16\_MAX & \text{if} \quad (fltIn) > FRAC16\_MAX \end{cases}$$

Equation **MLIB_ConvertPU_Eq1**

Note

The function may raise floating-point exceptions (floating-point inexact, invalid operation).

Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

5.51.6.5 Code Example

```
#include "mlib.h"

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);
}
```

5.51.7 Function MLIB_ConvertPU_FLTF16

5.51.7.1 Declaration

```
INLINE tFloat MLIB_ConvertPU_FLTF16(register tFrac16 f16In);
```

5.51.7.2 Arguments

Table 5-174. MLIB_ConvertPU_FLTF16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	Input value in 16-bit fractional format to be converted.

5.51.7.3 Return

Converted input value in single precision floating point format.

5.51.7.4 Variant Specifics

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_Eq1**

Note

The function may raise floating-point exceptions (floating-point inexact, invalid operation).

Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

5.51.7.5 Code Example

```
#include "mlib.h"

tF16 f16In;
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);
}
```

5.51.8 Function MLIB_ConvertPU_FLTF32

5.51.8.1 Declaration

```
INLINE tFloat MLIB_ConvertPU_FLTF32(register tFrac32 f32In);
```

5.51.8.2 Arguments

Table 5-175. MLIB_ConvertPU_FLTF32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value in 32-bit fractional format to be converted.

5.51.8.3 Return

Converted input value in single precision floating point format.

5.51.8.4 Variant Specifics

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_Eq1**

Note

The function may raise floating-point exceptions (floating-point inexact, invalid operation).

Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

5.51.8.5 Code Example

```
#include "mlib.h"

tF32 f32In;
```

```
tFloat fltOut;

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);
}
```

5.52 Function MLIB_Div

This function divides the first parameter by the second one.

5.52.1 Description

This inline function returns the division of two input values. The first input value is numerator and the second input value is denominator.

5.52.2 Re-entrancy

The function is re-entrant.

5.52.3 Function MLIB_Div_F32

5.52.3.1 Declaration

```
INLINE tFrac32 MLIB_Div_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

5.52.3.2 Arguments

Table 5-176. MLIB_Div_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Numerator of division.
register tFrac32	f32In2	**input**	Denominator of division.

5.52.3.3 Return

Division of two input values.

5.52.3.4 Variant Specifics

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.

The output of the function is defined by the following simple equation:

$$f32\text{Out} = \begin{cases} \text{FRAC32\_MIN} & \text{if } (f32\text{In2} = 0) \,\&\, (f32\text{In1} \leq 0) \\ \dfrac{f32\text{In1}}{f32\text{In2}} & \text{if } \quad f32\text{In2} \neq 0 \\ \text{FRAC32\_MAX} & \text{if } (f32\text{In2} = 0) \,\&\, (f32\text{In1} > 0) \end{cases}$$

Equation **MLIB_Div_Eq1**

Note

Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

CAUTION

Due to effectivity reason the division is calculated in 16-bit precision.

5.52.3.5 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);
}
```

5.52.4 Function MLIB_Div_F16

5.52.4.1 Declaration

```
INLINE tFrac16 MLIB_Div_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.52.4.2 Arguments

Table 5-177. MLIB_Div_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Numerator of division.
register tFrac16	f16In2	**input**	Denominator of division.

5.52.4.3 Return

Division of two input values.

5.52.4.4 Variant Specifics

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.

The output of the function is defined by the following simple equation:

$$f16\mathrm{Out} = \begin{cases} \mathrm{FRAC32\_MIN} & \text{if} \quad (f32\mathrm{In2} = 0) \,\&\, (f32\mathrm{In1} \leq 0) \\ \dfrac{f32\mathrm{In1}}{f32\mathrm{In2}} & \text{if} \quad f32\mathrm{In2} \neq 0 \\ \mathrm{FRAC32\_MAX} & \text{if} \quad (f32\mathrm{In2} = 0) \,\&\, (f32\mathrm{In1} > 0) \end{cases}$$

Equation **MLIB_Div_Eq1**

Note

Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

5.52.4.5 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);
}
```

5.52.5 Function MLIB_Div_FLT

5.52.5.1 Declaration

```
INLINE tFloat MLIB_Div_FLT(register tFloat fltIn1, register tFloat fltIn2);
```

5.52.5.2 Arguments

Table 5-178. MLIB_Div_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn1	**input**	Numerator of division.
register tFloat	fltIn2	**input**	Denominator of division.

5.52.5.3 Return

Division of two input values.

5.52.5.4 Variant Specifics

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:

$$\mathrm{fltOut} = \begin{cases} \mathrm{FLOAT\_MIN} & \text{if} \quad (\mathrm{fltIn2} = 0) \,\&\, (\mathrm{fltIn1} \leq 0) \\ \frac{\mathrm{fltIn1}}{\mathrm{fltIn2}} & \text{if} \quad \mathrm{fltIn2} \neq 0 \\ \mathrm{FLOAT\_MAX} & \text{if} \quad (\mathrm{fltIn2} = 0) \,\&\, (\mathrm{fltIn1} > 0) \end{cases}$$

Equation **MLIB_Div_Eq1**

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation, zero divide).

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.52.5.5 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);
}
```

5.53 Function MLIB_DivSat

This function divides the first parameter by the second one and saturate.

5.53.1 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.

5.53.2 Re-entrancy

The function is re-entrant.

5.53.3 Function MLIB_DivSat_F32

5.53.3.1 Declaration

```
INLINE tFrac32 MLIB_DivSat_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

5.53.3.2 Arguments

Table 5-179. MLIB_DivSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Numerator of division.
register tFrac32	f32In2	**input**	Denominator of division.

5.53.3.3 Return

Division of two input values, saturated if necessary.

5.53.3.4 Variant Specifics

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:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad \frac{f32In1}{f32In2} < FRAC32\_MIN \\ \frac{f32In1}{f32In2} & \text{if} \quad FRAC32\_MIN \leq \frac{f32In1}{f32In2} \leq FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad \frac{f32In1}{f32In2} > FRAC32\_MAX \end{cases}$$

Equation **MLIB_DivSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

5.53.3.5 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);
}
```

5.53.4 Function MLIB_DivSat_F16

5.53.4.1 Declaration

```
INLINE tFrac16 MLIB_DivSat_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.53.4.2 Arguments

Table 5-180. MLIB_DivSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Numerator of division.
register tFrac16	f16In2	**input**	Denominator of division.

5.53.4.3 Return

Division of two input values, saturated if necessary.

5.53.4.4 Variant Specifics

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:

$$f16Out = \begin{cases} FRAC16\_MIN & \text{if} \quad \frac{f16In1}{f16In2} < FRAC16\_MIN \\ \frac{f16In1}{f16In2} & \text{if} \quad FRAC16\_MIN \le \frac{f16In1}{f16In2} \le FRAC16\_MAX \\ FRAC16\_MAX & \text{if} \quad \frac{f16In1}{f16In2} > FRAC16\_MAX \end{cases}$$

Equation **MLIB_DivSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

5.53.4.5 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_DivSat_F16 (f16In1, f16In2);

  // 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);
}
```

5.54 Function MLIB_Mac

This function implements the multiply accumulate function.

5.54.1 Description

This inline function returns the multiplied second and third input value with adding of first input value.

5.54.2 Re-entrancy

The function is re-entrant.

5.54.3 Function MLIB_Mac_F32

5.54.3.1 Declaration

```
INLINE tFrac32 MLIB_Mac_F32(register tFrac32 f32In1, register tFrac32 f32In2, register
tFrac32 f32In3);
```

5.54.3.2 Arguments

Table 5-181. 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.

5.54.3.3 Return

Multiplied second and third input value with adding of first input value.

5.54.3.4 Variant Specifics

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:

$$f32Out = \left(f32In1 + \left(f32In2 \bullet f32In3\right)\right)$$

Equation **MLIB_Mac_Eq1**

Note

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.54.3.5 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);
}
```

5.54.4 Function MLIB_Mac_F32F16F16

5.54.4.1 Declaration

```
INLINE tFrac32 MLIB_Mac_F32F16F16(register tFrac32 f32In1, register tFrac16 f16In2, register
tFrac16 f16In3);
```

5.54.4.2 Arguments

Table 5-182. 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.

5.54.4.3 Return

Multiplied second and third input value with adding of first input value.

5.54.4.4 Variant Specifics

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:

$$f32Out = (f32In1 + (f16In2 \bullet f16In3))$$

Equation **MLIB_Mac_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

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.54.4.5 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);

}
```

5.54.5 Function MLIB_Mac_F16

5.54.5.1 Declaration

```
INLINE tFrac16 MLIB_Mac_F16(register tFrac16 f16In1, register tFrac16 f16In2, register
tFrac16 f16In3);
```

5.54.5.2 Arguments

Table 5-183. 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.

5.54.5.3 Return

Multiplied second and third input value with adding of first input value.

5.54.5.4 Variant Specifics

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:

$$f16Out = \left(f16In1 + \left(f16In2 \bullet f16In3\right)\right)$$

Equation **MLIB_Mac_Eq1**

Note

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.54.5.5 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
  f16Out = MLIB_Mac (f16In1, f16In2, f16In3);
}
```

5.54.6 Function MLIB_Mac_FLT

5.54.6.1 Declaration

```
INLINE tFloat MLIB_Mac_FLT(register tFloat fltIn1, register tFloat fltIn2, register tFloat fltIn3);
```

5.54.6.2 Arguments

Table 5-184. 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.

5.54.6.3 Return

Multiplied second and third input value with adding of first input value.

5.54.6.4 Variant Specifics

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:

$$fltOut = (fltIn1 + (fltIn2 \bullet fltIn3))$$

Equation **MLIB_Mac_Eq1**

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

5.54.6.5 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;

  // 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);
      }
```

5.55 Function MLIB_MacSat

This function implements the multiply accumulate function saturated if necessary.

5.55.1 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.

5.55.2 Re-entrancy

The function is re-entrant.

5.55.3 Function MLIB_MacSat_F32

5.55.3.1 Declaration

```
INLINE tFrac32 MLIB_MacSat_F32(register tFrac32 f32In1, register tFrac32 f32In2, register
tFrac32 f32In3);
```

5.55.3.2 Arguments

Table 5-185. 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.

5.55.3.3 Return

Multiplied second and third input value with adding of first input value. The output value is saturated if necessary.

5.55.3.4 Variant Specifics

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:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad (f32In1 + (f32In2 \bullet f32In3)) < FRAC32\_MIN \\ (f32In1 + (f32In2 \bullet f32In3)) & \text{if} \quad FRAC32\_MIN \leq (f32In1 + (f32In2 \bullet f32In3)) \leq FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad (f32In1 + (f32In2 \bullet f32In3)) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_MacSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline assembly and thus is not ANSI-C compliant.

5.55.3.5 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);
}
```

5.55.4 Function MLIB_MacSat_F32F16F16

5.55.4.1 Declaration

```
INLINE tFrac32 MLIB_MacSat_F32F16F16(register tFrac32 f32In1, register tFrac16 f16In2,
register tFrac16 f16In3);
```

5.55.4.2 Arguments

Table 5-186. 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.
register tFrac16	f16In3	**input**	Second value to be multiplied.

5.55.4.3 Return

Multiplied second and third input value with adding of first input value. The output value is saturated if necessary.

5.55.4.4 Variant Specifics

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:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad (f32In1 + (f16In2 \bullet f16In3)) < FRAC32\_MIN \\ (f32In1 + (f16In2 \bullet f16In3)) & \text{if} \quad FRAC32\_MIN \le (f32In1 + (f16In2 \bullet f16In3)) \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad (f32In1 + (f16In2 \bullet f16In3)) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_MacSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline assembly and thus is not ANSI-C compliant.

5.55.4.5 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);
}
```

5.55.5 Function MLIB_MacSat_F16

5.55.5.1 Declaration

```
INLINE tFrac16 MLIB_MacSat_F16(register tFrac16 f16In1, register tFrac16 f16In2, register
tFrac16 f16In3);
```

5.55.5.2 Arguments

Table 5-187. 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.

5.55.5.3 Return

Multiplied second and third input value with adding of first input value. The output value is saturated if necessary.

5.55.5.4 Variant Specifics

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:

$$f16Out = \begin{cases} \text{FRAC16\_MIN} & \text{if} \quad (f16In1 + (f16In2 \bullet f16In3)) < \text{FRAC16\_MIN} \\ (f16In1 + (f16In2 \bullet f16In3)) & \text{if} \quad \text{FRAC16\_MIN} \leq (f16In1 + (f16In2 \bullet f16In3)) \leq \text{FRAC16\_MAX} \\ \text{FRAC16\_MAX} & \text{if} \quad (f16In1 + (f16In2 \bullet f16In3)) > \text{FRAC16\_MAX} \end{cases}$$

Equation **MLIB_MacSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline assembly and thus is not ANSI-C compliant.

5.55.5.5 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
    // ################################################################

    // output should be FRAC16(0.3025) = 0x26B8
    f16Out = MLIB_MacSat (f16In1, f16In2, f16In3);
}
```

5.56 Function MLIB_Mnac

This function implements the multiply-subtract function. This function implements the multiply-subtract function.

5.56.1 Description

This inline function returns the multiplied second and third input value with subtracted first input value.

5.56.2 Re-entrancy

The function is re-entrant.

5.56.3 Function MLIB_Mnac_F32

5.56.3.1 Declaration

```
INLINE tFrac32 MLIB_Mnac_F32(register tFrac32 f32In1, register tFrac32 f32In2, register
tFrac32 f32In3);
```

5.56.3.2 Arguments

Table 5-188. 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.

5.56.3.3 Return

Multiplied second and third input value with subtracted first input value.

5.56.3.4 Variant Specifics

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:

$$f32Out = \left(-f32In1 + \left(f32In2 \bullet f32In3\right)\right)$$

Equation **MLIB_Mnac_Eq1**

Note

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.56.3.5 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
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);
}
```

5.56.4 Function MLIB_Mnac_F32F16F16

5.56.4.1 Declaration

```
INLINE tFrac32 MLIB_Mnac_F32F16F16(register tFrac32 f32In1, register tFrac16 f16In2, register
tFrac16 f16In3);
```

5.56.4.2 Arguments

Table 5-189. 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.

5.56.4.3 Return

Multiplied second and third input value with subtracted first input value.

5.56.4.4 Variant Specifics

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:

$$f32Out = \left(-f32In1 + \left(f16In2 \bullet f16In3\right)\right)$$

Equation **MLIB_Mnac_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

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.56.4.5 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);

}
```

5.56.5 Function MLIB_Mnac_F16

5.56.5.1 Declaration

```
INLINE tFrac16 MLIB_Mnac_F16(register tFrac16 f16In1, register tFrac16 f16In2, register
tFrac16 f16In3);
```

5.56.5.2 Arguments

Table 5-190. 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.

5.56.5.3 Return

Multiplied second and third input value with subtracted first input value.

5.56.5.4 Variant Specifics

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:

$$f16Out = \left(-f16In1 + \left(f16In2 \bullet f16In3\right)\right)$$

Equation **MLIB_Mnac_Eq1**

Note

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.56.5.5 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);
  }
```

5.56.6 Function MLIB_Mnac_FLT

5.56.6.1 Declaration

```
INLINE tFloat MLIB_Mnac_FLT(register tFloat fltIn1, register tFloat fltIn2, register tFloat
fltIn3);
```

5.56.6.2 Arguments

Table 5-191. 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.

5.56.6.3 Return

Multiplied second and third input value with subtracted first input value.

5.56.6.4 Variant Specifics

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:

$$fltOut = (-fltIn1 + fltIn2 \bullet fltIn3)$$

Equation **MLIB_Mnac_Eq1**

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

5.56.6.5 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
  // ##############################################################

  // output should be -1.8477294e-08
  fltOut = MLIB_Mnac (fltIn1, fltIn2, fltIn3);
}
```

5.57 Function MLIB_Msu

This function implements the multiply-subtract-from function.

5.57.1 Description

This inline function returns the first input value from which the multiplication result of the second and third input values is subtracted.

5.57.2 Re-entrancy

The function is re-entrant.

5.57.3 Function MLIB_Msu_F32

5.57.3.1 Declaration

```
INLINE tFrac32 MLIB_Msu_F32(register tFrac32 f32In1, register tFrac32 f32In2, register
tFrac32 f32In3);
```

5.57.3.2 Arguments

Table 5-192. 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.

5.57.3.3 Return

First input value from which the multiplication result of the second and third input values is subtracted.

5.57.3.4 Variant Specifics

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:

$$f32Out = (f32In1 - (f32In2 \bullet f32In3))$$

Equation **MLIB_Msu_Eq1**

Note

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.57.3.5 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);
}
```

5.57.4 Function MLIB_Msu_F32F16F16

5.57.4.1 Declaration

```
INLINE tFrac32 MLIB_Msu_F32F16F16(register tFrac32 f32In1, register tFrac16 f16In2, register
tFrac16 f16In3);
```

5.57.4.2 Arguments

Table 5-193. 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.

5.57.4.3 Return

First input value from which the multiplication result of the second and third input values is subtracted.

5.57.4.4 Variant Specifics

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:

$$f32Out = (f32In1 - (f16In2 \bullet f16In3))$$

Equation **MLIB_Msu_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

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.57.4.5 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);

}
```

5.57.5 Function MLIB_Msu_F16

5.57.5.1 Declaration

```
INLINE tFrac16 MLIB_Msu_F16(register tFrac16 f16In1, register tFrac16 f16In2, register
tFrac16 f16In3);
```

5.57.5.2 Arguments

Table 5-194. 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.

5.57.5.3 Return

First input value from which the multiplication result of the second and third input values is subtracted.

5.57.5.4 Variant Specifics

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:

$$f16Out = \left(f16In1 - \left(f16In2 \cdot f16In3\right)\right)$$

Equation **MLIB_Msu_Eq1**

Note

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.57.5.5 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
  // ##############################################################

  // output should be FRAC16(0.1875) = 0x1800
  f16Out = MLIB_Msu (f16In1, f16In2, f16In3);
}
```

5.57.6 Function MLIB_Msu_FLT

5.57.6.1 Declaration

```
INLINE tFloat MLIB_Msu_FLT(register tFloat fltIn1, register tFloat fltIn2, register tFloat
fltIn3);
```

5.57.6.2 Arguments

Table 5-195. 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.

5.57.6.3 Return

First input value from which the multiplication result of the second and third input values is subtracted.

5.57.6.4 Variant Specifics

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:

$$fltOut = (fltIn1 - fltIn2 \bullet fltIn3)$$

Equation **MLIB_Msu_Eq1**

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

5.57.6.5 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
  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);
}
```

5.58 Function MLIB_Mul

This function multiplies two input parameters.

5.58.1 Description

This inline function multiplies the two input values.

5.58.2 Re-entrancy

The function is re-entrant.

5.58.3 Function MLIB_Mul_F32

5.58.3.1 Declaration

```
INLINE tFrac32 MLIB_Mul_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

5.58.3.2 Arguments

Table 5-196. 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).

5.58.3.3 Return

Fractional multiplication of the input arguments.

5.58.3.4 Variant Specifics

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:

$$f32Out = f32In1 \bullet f32In2$$

Equation **MLIB_Mul_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.58.3.5 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);
}
```

5.58.4 Function MLIB_Mul_F32F16F16

5.58.4.1 Declaration

```
INLINE tFrac32 MLIB_Mul_F32F16F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.58.4.2 Arguments

Table 5-197. 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).

5.58.4.3 Return

Fractional multiplication of the input arguments.

5.58.4.4 Variant Specifics

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:

$$f32Out = f16In1 \bullet f16In2$$

Equation **MLIB_Mul_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.58.4.5 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);
}
```

5.58.5 Function MLIB_Mul_F16

5.58.5.1 Declaration

```
INLINE tFrac16 MLIB_Mul_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.58.5.2 Arguments

Table 5-198. 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).

5.58.5.3 Return

Fractional multiplication of the input arguments.

5.58.5.4 Variant Specifics

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:

$$f16Out = f16In1 \bullet f16In2$$

Equation **MLIB_Mul_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.58.5.5 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);


    // ###############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x1000 = FRAC16(0.125)
    f16Out = MLIB_Mul (f16In1,f16In2);
  }
```

5.58.6 Function MLIB_Mul_FLT

5.58.6.1 Declaration

```
INLINE tFloat MLIB_Mul_FLT(register tFloat fltIn1, register tFloat fltIn2);
```

5.58.6.2 Arguments

Table 5-199. 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.

5.58.6.3 Return

Floating point multiplication of the input arguments.

5.58.6.4 Variant Specifics

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:

$$\mathrm{fltOut} = \mathrm{fltIn1} \bullet \mathrm{fltIn2}$$

Equation **MLIB_Mul_Eq1**

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.58.6.5 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 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);
}
```

5.59 Function MLIB_MulSat

This function multiplies two input parameters and saturate if necessary.

5.59.1 Description

This inline function multiplies the two input values and saturates the result.

5.59.2 Re-entrancy

The function is re-entrant.

5.59.3 Function MLIB_MulSat_F32

5.59.3.1 Declaration

```
INLINE tFrac32 MLIB_MulSat_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

5.59.3.2 Arguments

Table 5-200. 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).

5.59.3.3 Return

Fractional multiplication of the input arguments.

5.59.3.4 Variant Specifics

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:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad (f32In1 \cdot f32In2) < FRAC32\_MIN \\ (f32In1 \cdot f32In2) & \text{if} \quad FRAC32\_MIN \le (f32In1 \cdot f32In2) \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad (f32In1 \cdot f32In2) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_MulSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline assembly and thus is not ANSI-C compliant.

5.59.3.5 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 0x4ccccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat_F32 (f32In1,f32In2);

  // output should be 0x4ccccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat (f32In1,f32In2,F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be 0x4ccccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat (f32In1,f32In2);
}
```

5.59.4 Function MLIB_MulSat_F32F16F16

5.59.4.1 Declaration

```
INLINE tFrac32 MLIB_MulSat_F32F16F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.59.4.2 Arguments

Table 5-201. 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).

5.59.4.3 Return

Fractional multiplication of the input arguments.

5.59.4.4 Variant Specifics

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:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad (f16In1 \bullet f16In2) < FRAC32\_MIN \\ (f16In1 \bullet f16In2) & \text{if} \quad FRAC32\_MIN \le (f16In1 \bullet f16In2) \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad (f16In1 \bullet f16In2) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_MulSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline assembly and thus is not ANSI-C compliant.

5.59.4.5 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 0x4ccccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat_F32F16F16 (f16In1,f16In2);

  // output should be 0x4ccccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat (f32In1,f32In2,F32F16f16);
}
```

5.59.5 Function MLIB_MulSat_F16

5.59.5.1 Declaration

```
INLINE tFrac16 MLIB_MulSat_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.59.5.2 Arguments

Table 5-202. MLIB_MulSat_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).

5.59.5.3 Return

Fractional multiplication of the input arguments.

5.59.5.4 Variant Specifics

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:

$$f16Out = \begin{cases} FRAC16\_MIN & \text{if} \quad (f16In1 \bullet f16In2) < FRAC16\_MIN \\ (f16In1 \bullet f16In2) & \text{if} \quad FRAC16\_MIN \le (f16In1 \bullet f16In2) \le FRAC16\_MAX \\ FRAC16\_MAX & \text{if} \quad (f16In1 \bullet f16In2) > FRAC16\_MAX \end{cases}$$

Equation **MLIB_MulSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline assembly and thus is not ANSI-C compliant.

5.59.5.5 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);
}
```

5.60 Function MLIB_Neg

This function returns negative value of input parameter. This function returns negative value of input parameter.

5.60.1 Description

This inline function returns the negative value of input parameter.

5.60.2 Re-entrancy

The function is re-entrant.

5.60.3 Function MLIB_Neg_F32

5.60.3.1 Declaration

```
INLINE tFrac32 MLIB_Neg_F32(register tFrac32 f32In);
```

5.60.3.2 Arguments

Table 5-203. MLIB_Neg_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value which negative value should be returned.

5.60.3.3 Return

Negative value of input parameter.

5.60.3.4 Variant Specifics

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_Eq1**

Note

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.60.3.5 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);

  // ################################################################
```

```
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be FRAC32(-0.25) = 0xA0000000
    f32Out = MLIB_Neg (f32In);
}
```

5.60.4 Function MLIB_Neg_F16

5.60.4.1 Declaration

```
INLINE tFrac16 MLIB_Neg_F16(register tFrac16 f16In);
```

5.60.4.2 Arguments

Table 5-204. MLIB_Neg_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	Input value which negative value should be returned.

5.60.4.3 Return

Negative value of input parameter.

5.60.4.4 Variant Specifics

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_Eq1**

Note

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.60.4.5 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);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be FRAC16(-0.25) = 0xA000
  f16Out = MLIB_Neg (f16In);
}
```

5.60.5 Function MLIB_Neg_FLT

5.60.5.1 Declaration

```
INLINE tFloat MLIB_Neg_FLT(register tFloat fltIn);
```

5.60.5.2 Arguments

Table 5-205. MLIB_Neg_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn	**input**	Input value which negative value should be returned.

5.60.5.3 Return

Negative value of input parameter.

5.60.5.4 Variant Specifics

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_Eq1**

Note

Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.60.5.5 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_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);
}
```

5.61 Function MLIB_NegSat

This function returns negative value of input parameter and saturate if necessary.

5.61.1 Description

This inline function returns the negative value of input parameter and saturates the result if necessary.

5.61.2 Re-entrancy

The function is re-entrant.

5.61.3 Function MLIB_NegSat_F32

5.61.3.1 Declaration

```
INLINE tFrac32 MLIB_NegSat_F32(register tFrac32 f32In);
```

5.61.3.2 Arguments

Table 5-206. MLIB_NegSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value which negative value should be returned.

5.61.3.3 Return

Negative value of input parameter.

5.61.3.4 Variant Specifics

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:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad -(f32In) < FRAC32\_MIN \\ -(f32In) & \text{if} \quad FRAC32\_MIN \leq -(f32In) \leq FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad -(f32In) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_NegSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

5.61.3.5 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);
}
```

5.61.4 Function MLIB_NegSat_F16

5.61.4.1 Declaration

```
INLINE tFrac16 MLIB_NegSat_F16(register tFrac16 f16In);
```

5.61.4.2 Arguments

Table 5-207. MLIB_NegSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	Input value which negative value should be returned.

5.61.4.3 Return

Negative value of input parameter.

5.61.4.4 Variant Specifics

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:

$$f16Out = \begin{cases} FRAC16\_MIN & \text{if} \quad -(f16In) < FRAC16\_MIN \\ -(f16In) & \text{if} \quad FRAC16\_MIN \le -(f16In) \le FRAC16\_MAX \\ FRAC16\_MAX & \text{if} \quad -(f16In) > FRAC16\_MAX \end{cases}$$

Equation **MLIB_NegSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

5.61.4.5 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);
}
```

5.62 Function MLIB_Norm

This function returns the number of left shifts needed to normalize the input parameter.

5.62.1 Description

The function returns the number of left shifts needed to remove redundant leading sign bits.

5.62.2 Re-entrancy

The function is re-entrant.

5.62.3 Function MLIB_Norm_F32

5.62.3.1 Declaration

```
INLINE tU16 MLIB_Norm_F32(register tFrac32 f32In);
```

5.62.3.2 Arguments

Table 5-208. MLIB_Norm_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	The first value to be normalized.

5.62.3.3 Return

The number of left shift needed to normalize the argument.

Note

Due to effectivity reason this function is implemented as inline assembly and thus is not ANSI-C compliant.

5.62.3.4 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);
}
```

5.62.4 Function MLIB_Norm_F16

5.62.4.1 Declaration

```
INLINE tU16 MLIB_Norm_F16(register tFrac16 f16In);
```

5.62.4.2 Arguments

Table 5-209. MLIB_Norm_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	The first value to be normalized.

5.62.4.3 Return

The number of left shift needed to normalize the argument.

Note

Due to effectivity reason this function is implemented as inline assembly and thus is not ANSI-C compliant.

5.62.4.4 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);
}
```

5.63 Function MLIB_RndSat_F16F32

This function rounds the input value to the nearest saturated value in the output format.

5.63.1 Declaration

```
INLINE tFrac16 MLIB_RndSat_F16F32(register tFrac32 f32In);
```

5.63.2 Arguments

Table 5-210. MLIB_RndSat_F16F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value.

5.63.3 Return

Rounded saturated value.

5.63.4 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.

5.63.5 Re-entrancy

The function is re-entrant.

5.63.6 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);
}
```

5.64 Function MLIB_Round

The function rounds the input and saturates.

5.64.1 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.

5.64.2 Re-entrancy

The function is re-entrant.

5.64.3 Function MLIB_Round_F32

5.64.3.1 Declaration

```
INLINE tFrac32 MLIB_Round_F32(register tFrac32 f32In1, register tU16 u16In2);
```

5.64.3.2 Arguments

Table 5-211. 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.

5.64.3.3 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. Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.64.3.4 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);
    }
```

5.64.4 Function MLIB_Round_F16

5.64.4.1 Declaration

```
INLINE tFrac16 MLIB_Round_F16(register tFrac16 f16In1, register tU16 u16In2);
```

5.64.4.2 Arguments

Table 5-212. 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.

5.64.4.3 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. Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.64.4.4 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,F16);

    // ##############################################################
    // 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
    f16In1 = FRAC16 (-0.375);
    // second input = 13
    u16In2 = (tU16)13;

    // output should be 0xE000 ~ FRAC16(-0.25)
    f16Out = MLIB_Round_F16 (f16In1,u16In2);
}
```

5.65 Function MLIB_ShBi

This function shifts the first argument to left or right by number defined by second argument.

5.65.1 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.

5.65.2 Re-entrancy

The function is re-entrant.

5.65.3 Function MLIB_ShBi_F32

5.65.3.1 Declaration

```
INLINE tFrac32 MLIB_ShBi_F32(register tFrac32 f32In1, register tS16 s16In2);
```

5.65.3.2 Arguments

Table 5-213. 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.

5.65.3.3 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. Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.65.3.4 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_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);
}
```

5.65.4 Function MLIB_ShBi_F16

5.65.4.1 Declaration

```
INLINE tFrac16 MLIB_ShBi_F16(register tFrac16 f16In1, register tS16 s16In2);
```

5.65.4.2 Arguments

Table 5-214. 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.

5.65.4.3 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. Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.65.4.4 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
    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);
}
```

5.66 Function MLIB_ShBiSat

This function shifts the first argument to left or right by number defined by second argument and saturate if necessary.

5.66.1 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.

5.66.2 Re-entrancy

The function is re-entrant.

5.66.3 Function MLIB_ShBiSat_F32

5.66.3.1 Declaration

```
INLINE tFrac32 MLIB_ShBiSat_F32(register tFrac32 f32In1, register tS16 s16In2);
```

5.66.3.2 Arguments

Table 5-215. 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.

5.66.3.3 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. Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.66.3.4 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);
}
```

5.66.4 Function MLIB_ShBiSat_F16

5.66.4.1 Declaration

```
INLINE tFrac16 MLIB_ShBiSat_F16(register tFrac16 f16In1, register tS16 s16In2);
```

5.66.4.2 Arguments

Table 5-216. 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.

5.66.4.3 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. Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.66.4.4 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
    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);
}
```

5.67 Function MLIB_ShL

This function shifts the first parameter to left by number defined by second parameter.
This function shifts the first parameter to left by number defined by second parameter.

5.67.1 Description

This function shifts the first argument to the left by the amount specified in the second argument. Overflow is not detected.

5.67.2 Re-entrancy

The function is re-entrant.

5.67.3 Function MLIB_ShL_F32

5.67.3.1 Declaration

```
INLINE tFrac32 MLIB_ShL_F32(register tFrac32 f32In1, register tU16 u16In2);
```

5.67.3.2 Arguments

Table 5-217. 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.

5.67.3.3 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. Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.67.3.4 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);
}
```

5.67.4 Function MLIB_ShL_F16

5.67.4.1 Declaration

```
INLINE tFrac16 MLIB_ShL_F16(register tFrac16 f16In1, register tU16 u16In2);
```

5.67.4.2 Arguments

Table 5-218. 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.

5.67.4.3 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. Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.67.4.4 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);
}
```

5.68 Function MLIB_ShLSat

This function shifts the first parameter to left by number defined by second parameter and saturate if necessary.

5.68.1 Description

This function shifts the first argument to the left by the amount specified in the second argument. The result is saturated.

5.68.2 Re-entrancy

The function is re-entrant.

5.68.3 Function MLIB_ShLSat_F32

5.68.3.1 Declaration

```
INLINE tFrac32 MLIB_ShLSat_F32(register tFrac32 f32In1, register tU16 u16In2);
```

5.68.3.2 Arguments

Table 5-219. 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.

5.68.3.3 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. Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

5.68.3.4 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);
}
```

5.68.4 Function MLIB_ShLSat_F16

5.68.4.1 Declaration

```
INLINE tFrac16 MLIB_ShLSat_F16(register tFrac16 f16In1, register tU16 u16In2);
```

5.68.4.2 Arguments

Table 5-220. 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.

5.68.4.3 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. Due to effectivity reason this function is implemented as inline assembly, and thus is not ANSI-C compliant.

5.68.4.4 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);
}
```

5.69 Function MLIB_ShR

This function shifts the first parameter to right by number defined by second parameter.

5.69.1 Description

This function shifts the first argument to the right by the amount specified in the second argument.

5.69.2 Re-entrancy

The function is re-entrant.

5.69.3 Function MLIB_ShR_F32

5.69.3.1 Declaration

```
INLINE tFrac32 MLIB_ShR_F32(register tFrac32 f32In1, register tU16 u16In2);
```

5.69.3.2 Arguments

Table 5-221. 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.

5.69.3.3 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. Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.69.3.4 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);
}
```

5.69.4 Function MLIB_ShR_F16

5.69.4.1 Declaration

```
INLINE tFrac16 MLIB_ShR_F16(register tFrac16 f16In1, register tU16 u16In2);
```

5.69.4.2 Arguments

Table 5-222. 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.

5.69.4.3 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. Due to effectivity reason this function is implemented as inline, and thus is not ANSI-C compliant.

5.69.4.4 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);
}
```

5.70 Function MLIB_Sub

This function subtracts the second parameter from the first one.

5.70.1 Description

The second argument is subtracted from the first one.

5.70.2 Re-entrancy

The function is re-entrant.

5.70.3 Function MLIB_Sub_F32

5.70.3.1 Declaration

```
INLINE tFrac32 MLIB_Sub_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

5.70.3.2 Arguments

Table 5-223. 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).

5.70.3.3 Return

The subtraction of the second argument from the first argument.

5.70.3.4 Variant Specifics

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:

$$f32Out = \left(f32In1 - f32In2\right)$$

Equation **MLIB_Sub_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.70.3.5 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_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);
}
```

5.70.4 Function MLIB_Sub_F16

5.70.4.1 Declaration

```
INLINE tFrac16 MLIB_Sub_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.70.4.2 Arguments

Table 5-224. 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).

5.70.4.3 Return

The subtraction of the second argument from the first argument.

5.70.4.4 Variant Specifics

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:

$$f16Out = \left(f16In1 - f16In2\right)$$

Equation **MLIB_Sub_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.70.4.5 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_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);
}
```

5.70.5 Function MLIB_Sub_FLT

5.70.5.1 Declaration

```
INLINE tFloat MLIB_Sub_FLT(register tFloat fltIn1, register tFloat fltIn2);
```

5.70.5.2 Arguments

Table 5-225. 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.

5.70.5.3 Return

The subtraction of the second argument from the first argument.

5.70.5.4 Variant Specifics

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_Eq1**

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.70.5.5 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);
}
```

5.71 Function MLIB_SubSat

This function subtracts the second parameter from the first one and saturate if necessary.

5.71.1 Description

The second argument is subtracted from the first one. The result is saturated if necessary

5.71.2 Re-entrancy

The function is re-entrant.

5.71.3 Function MLIB_SubSat_F32

5.71.3.1 Declaration

```
INLINE tFrac32 MLIB_SubSat_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

5.71.3.2 Arguments

Table 5-226. 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).

5.71.3.3 Return

The subtraction of the second argument from the first argument.

5.71.3.4 Variant Specifics

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:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad (f32In1 - f32In2) < FRAC32\_MIN \\ (f32In1 - f32In2) & \text{if} \quad FRAC32\_MIN \le (f32In1 - f32In2) \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad (f32In1 - f32In2) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_SubSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline assembly and thus is not ANSI-C compliant.

5.71.3.5 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);
}
```

5.71.4 Function MLIB_SubSat_F16

5.71.4.1 Declaration

```
INLINE tFrac16 MLIB_SubSat_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

5.71.4.2 Arguments

Table 5-227. 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).

5.71.4.3 Return

The subtraction of the second argument from the first argument.

5.71.4.4 Variant Specifics

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:

$$f16Out = \begin{cases} FRAC16\_MIN & \text{if} \quad (f16In1 - f16In2) < FRAC16\_MIN \\ (f16In1 - f16In2) & \text{if} \quad FRAC16\_MIN \leq (f16In1 - f16In2) \leq FRAC16\_MAX \\ FRAC16\_MAX & \text{if} \quad (f16In1 - f16In2) > FRAC16\_MAX \end{cases}$$

Equation **MLIB_SubSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.71.4.5 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);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be 0x2000
  f16Out = MLIB_SubSat (f16In1,f16In2);
}
```

5.72 Function MLIB_VMac

This function implements the vector multiply accumulate function. This function implements the vector multiply accumulate function.

5.72.1 Description

This inline function returns the dot product of four input values.

5.72.2 Re-entrancy

The function is re-entrant.

5.72.3 Function MLIB_VMac_F32

5.72.3.1 Declaration

```
INLINE tFrac32 MLIB_VMac_F32(register tFrac32 f32In1, register tFrac32 f32In2, register
tFrac32 f32In3, register tFrac32 f32In4);
```

5.72.3.2 Arguments

Table 5-228. 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.

5.72.3.3 Return

Vector multiplied input values with addition.

5.72.3.4 Variant Specifics

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 vector multiply-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:

$$f32Out = (f32In1 \bullet f32In2) + (f32In3 \bullet f32In4)$$

Equation **MLIB_VMac_Eq1**

Note

Due to effectivity reason this function is implemented as inline assembly and thus is not ANSI-C compliant.

5.72.3.5 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);

  // 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);
}
```

5.72.4 Function MLIB_VMac_F32F16F16

5.72.4.1 Declaration

```
INLINE tFrac32 MLIB_VMac_F32F16F16(register tFrac16 f16In1, register tFrac16 f16In2, register
tFrac16 f16In3, register tFrac16 f16In4);
```

5.72.4.2 Arguments

Table 5-229. 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.

5.72.4.3 Return

Vector multiplied input values with addition.

5.72.4.4 Variant Specifics

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 vector multiply-add of input values is outside the [-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = (f16In1 \bullet f16In2) + (f16In3 \bullet f16In4)$$

Equation **MLIB_VMac_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.72.4.5 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);
}
```

5.72.5 Function MLIB_VMac_F16

5.72.5.1 Declaration

```
INLINE tFrac16 MLIB_VMac_F16(register tFrac16 f16In1, register tFrac16 f16In2, register
tFrac16 f16In3, register tFrac16 f16In4);
```

5.72.5.2 Arguments

Table 5-230. 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.

5.72.5.3 Return

Vector multiplied input values with addition.

5.72.5.4 Variant Specifics

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 vector multiply-add of input values is outside the [-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f16Out = (f16In1 \bullet f16In2) + (f16In3 \bullet f16In4)$$

Equation **MLIB_VMac_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

5.72.5.5 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);
}
```

5.72.6 Function MLIB_VMac_FLT

5.72.6.1 Declaration

```
INLINE tFloat MLIB_VMac_FLT(register tFloat fltIn1, register tFloat fltIn2, register tFloat
fltIn3, register tFloat fltIn4);
```

5.72.6.2 Arguments

Table 5-231. 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.

5.72.6.3 Return

Vector multiplied input values with addition.

5.72.6.4 Variant Specifics

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:

$$\mathrm{flt0ut} = (\mathrm{fltIn1} \bullet \mathrm{fltIn2}) + (\mathrm{fltIn3} \bullet \mathrm{flt6In4})$$

Equation **MLIB_VMac_Eq1**

Note

The function may raise floating-point exceptions (floating-point overflow, underflow, inexact, invalid operation).

Due to effectivity reason this function is implemented as inline assembly and thus is not ANSI-C compliant.

5.72.6.5 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);

  // 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);
}
```

5.73 Function SWLIBS_GetVersion

This function returns the information about AMMCLIB version.

5.73.1 Declaration

```
const SWLIBS_VERSION_T * SWLIBS_GetVersion();
```

5.73.2 Return

The function returns the information about the version of Motor Control Library Set.

5.73.3 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

5.73.4 Reentrancy

The function is reentrant.

Chapter 6

6.1 Typedefs Index

Table 6-1. Quick typedefs reference

Type	Name	Description
typedef unsigned char	tBool	basic boolean type
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
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

Chapter 7
Compound Data Types

Table 7-1. Compound data types overview

Name	Description
AMCLIB_BEMF_OBSRV_DQ_T_F16	Observer configuration structure.
AMCLIB_BEMF_OBSRV_DQ_T_F32	Observer configuration structure.
AMCLIB_BEMF_OBSRV_DQ_T_FLT	Observer configuration structure.
AMCLIB_TRACK_OBSRV_T_F16	Structure containing the estimated position, estimated velocity and the algorithm parameters.
AMCLIB_TRACK_OBSRV_T_F32	Structure containing the estimated position, estimated velocity and the algorithm parameters.
AMCLIB_TRACK_OBSRV_T_FLT	Structure containing the estimated position, estimated velocity and the algorithm parameters.
GDFLIB_FILTER_IIR1_COEFF_T_F16	Sub-structure containing filter coefficients.
GDFLIB_FILTER_IIR1_COEFF_T_F32	Sub-structure containing filter coefficients.
GDFLIB_FILTER_IIR1_COEFF_T_FLT	Sub-structure containing filter coefficients.
GDFLIB_FILTER_IIR1_T_F16	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_IIR1_T_F32	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_IIR1_T_FLT	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_IIR2_COEFF_T_F16	Sub-structure containing filter coefficients.
GDFLIB_FILTER_IIR2_COEFF_T_F32	Sub-structure containing filter coefficients.
GDFLIB_FILTER_IIR2_COEFF_T_FLT	Sub-structure containing filter coefficients.
GDFLIB_FILTER_IIR2_T_F16	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_IIR2_T_F32	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_IIR2_T_FLT	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_MA_T_F16	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_MA_T_F32	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_MA_T_FLT	Structure containing filter buffer and coefficients.
GDFLIB_FILTERFIR_PARAM_T_F16	Structure containing parameters of the filter.
GDFLIB_FILTERFIR_PARAM_T_F32	Structure containing parameters of the filter.
GDFLIB_FILTERFIR_PARAM_T_FLT	Structure containing parameters of the filter.
GDFLIB_FILTERFIR_STATE_T_F16	Structure containing the current state of the filter.
GDFLIB_FILTERFIR_STATE_T_F32	Structure containing the current state of the filter.
GDFLIB_FILTERFIR_STATE_T_FLT	Structure containing the current state of the filter.

Table continues on the next page...

Table 7-1. Compound data types overview (continued)

Name	Description
GFLIB_ACOS_T_F16	Default approximation coefficients datatype for arccosine approximation.
GFLIB_ACOS_T_F32	Default approximation coefficients datatype for arccosine approximation.
GFLIB_ACOS_T_FLT	Default approximation coefficients datatype for arccosine approximation.
GFLIB_ACOS_TAYLOR_COEF_T_F16	Array of approximation coefficients for piece-wise polynomial.
GFLIB_ACOS_TAYLOR_COEF_T_F32	Array of approximation coefficients for piece-wise polynomial.
GFLIB_ASIN_T_F16	Default approximation coefficients datatype for arcsine approximation.
GFLIB_ASIN_T_F32	Default approximation coefficients datatype for arcsine approximation.
GFLIB_ASIN_T_FLT	Default approximation coefficients datatype for arcsine approximation.
GFLIB_ASIN_TAYLOR_COEF_T_F16	Array of approximation coefficients for piece-wise polynomial.
GFLIB_ASIN_TAYLOR_COEF_T_F32	Array of approximation coefficients for piece-wise polynomial.
GFLIB_ATAN_T_F16	Structure containing eight sub-structures with polynomial coefficients to cover all sub-intervals.
GFLIB_ATAN_T_F32	Structure containing eight sub-structures with polynomial coefficients to cover all sub-intervals.
GFLIB_ATAN_T_FLT	Structure containing the approximation coefficients.
GFLIB_ATAN_TAYLOR_COEF_T_F16	Array of polynomial approximation coefficients for one sub-interval.
GFLIB_ATAN_TAYLOR_COEF_T_F32	Array of minimax polynomial approximation coefficients for one sub-interval.
GFLIB_ATANYXSHIFTED_T_F16	Structure containing the parameter for the AtanYXShifted function.
GFLIB_ATANYXSHIFTED_T_F32	Structure containing the parameter for the AtanYXShifted function.
GFLIB_ATANYXSHIFTED_T_FLT	Structure containing the parameter for the AtanYXShifted function.
GFLIB_CONTROLLER_PI_P_T_F16	Structure containing parameters and states of the parallel form PI controller.
GFLIB_CONTROLLER_PI_P_T_F32	Structure containing parameters and states of the parallel form PI controller.
GFLIB_CONTROLLER_PI_P_T_FLT	Structure containing parameters and states of the parallel form PI controller.
GFLIB_CONTROLLER_PI_R_T_F16	Structure containing parameters and states of the recurrent form PI controller.
GFLIB_CONTROLLER_PI_R_T_F32	Structure containing parameters and states of the recurrent form PI controller.
GFLIB_CONTROLLER_PI_R_T_FLT	Structure containing parameters and states of the recurrent form PI controller.
GFLIB_CONTROLLER_PIAW_P_T_F16	Structure containing parameters and states of the parallel form PI controller with anti-windup.
GFLIB_CONTROLLER_PIAW_P_T_F32	Structure containing parameters and states of the parallel form PI controller with anti-windup.
GFLIB_CONTROLLER_PIAW_P_T_FLT	Structure containing parameters and states of the parallel form PI controller with anti-windup.
GFLIB_CONTROLLER_PIAW_R_T_F16	Structure containing parameters and states of the recurrent form PI controller with anti-windup.
GFLIB_CONTROLLER_PIAW_R_T_F32	Structure containing parameters and states of the recurrent form PI controller with anti-windup.
GFLIB_CONTROLLER_PIAW_R_T_FLT	Structure containing parameters and states of the recurrent form PI controller with anti-windup.
GFLIB_COS_T_F16	Array of four 16-bit elements for storing coefficients of the Taylor polynomial.

Table continues on the next page...

Table 7-1. Compound data types overview (continued)

Name	Description
GFLIB_COS_T_F32	Array of five 32-bit elements for storing coefficients of the Taylor polynomial.
GFLIB_COS_T_FLT	Array of three single precision floating point elements for storing coefficients of the floating point optimized minimax approximation polynomial.
GFLIB_HYST_T_F16	Structure containing parameters and states for the hysteresis function.
GFLIB_HYST_T_F32	Structure containing parameters and states for the hysteresis function.
GFLIB_HYST_T_FLT	Structure containing parameters and states for the hysteresis function.
GFLIB_INTEGRATOR_TR_T_F16	Structure containing integrator parameters and coefficients.
GFLIB_INTEGRATOR_TR_T_F32	Structure containing integrator parameters and coefficients.
GFLIB_INTEGRATOR_TR_T_FLT	Structure containing integrator parameters and coefficients.
GFLIB_LIMIT_T_F16	Structure containing the limits.
GFLIB_LIMIT_T_F32	Structure containing the limits.
GFLIB_LIMIT_T_FLT	Structure containing the limits.
GFLIB_LOG10_T_FLT	Array of single precision floating point elements for storing the coefficients of the floating point log10 approximation polynomial.
GFLIB_LOWERLIMIT_T_F16	Structure containing the lower limit.
GFLIB_LOWERLIMIT_T_F32	Structure containing the lower limit.
GFLIB_LOWERLIMIT_T_FLT	Structure containing the lower limit.
GFLIB_LUT1D_T_F16	Structure containing 1D look-up table parameters.
GFLIB_LUT1D_T_F32	Structure containing 1D look-up table parameters.
GFLIB_LUT1D_T_FLT	Structure containing 1D look-up table parameters.
GFLIB_LUT2D_T_F16	Structure containing 2D look-up table parameters.
GFLIB_LUT2D_T_F32	Structure containing 2D look-up table parameters.
GFLIB_LUT2D_T_FLT	Structure containing 2D look-up table parameters.
GFLIB_RAMP_T_F16	Structure containing increment/decrement coefficients and state value for the ramp function implemented in GFLIB_Ramp.
GFLIB_RAMP_T_F32	Structure containing increment/decrement coefficients and state value for the ramp function implemented in GFLIB_Ramp.
GFLIB_RAMP_T_FLT	Structure containing increment/decrement coefficients and state value for the ramp function implemented in GFLIB_Ramp.
GFLIB_SIN_T_F16	Array of four 16-bit elements for storing coefficients of the Taylor polynomial.
GFLIB_SIN_T_F32	Array of five 32-bit elements for storing coefficients of the Taylor polynomial.
GFLIB_SIN_T_FLT	Array of three single precision floating point elements for storing coefficients of the floating point optimized minimax approximation polynomial.
GFLIB_SINCOS_T_F16	Array of four 16-bit elements for storing coefficients of the Taylor polynomial.
GFLIB_SINCOS_T_F32	Array of five 32-bit elements for storing coefficients of the Taylor polynomial.
GFLIB_SINCOS_T_FLT	Array of three single precision floating point elements for storing coefficients of the floating point optimized minimax approximation polynomial.
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

Table continues on the next page...

Table 7-1. Compound data types overview (continued)

Name	Description
	four polynomial coefficients for each sub-interval, are stored in this (GFLIB_TAN_T_F16) structure.
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.
GFLIB_TAN_T_FLT	Polynomial coefficient for fractional approximation in single precision floating point format.
GFLIB_TAN_TAYLOR_COEF_T_F16	Structure containing four polynomial coefficients for one sub-interval.
GFLIB_TAN_TAYLOR_COEF_T_F32	Structure containing four polynomial coefficients for one sub-interval.
GFLIB_UPPERLIMIT_T_F16	Structure containing the upper limit.
GFLIB_UPPERLIMIT_T_F32	Structure containing the upper limit.
GFLIB_UPPERLIMIT_T_FLT	Structure containing the upper limit.
GFLIB_VECTORLIMIT_T_F16	Structure containing the limit.
GFLIB_VECTORLIMIT_T_F32	Structure containing the limit.
GFLIB_VECTORLIMIT_T_FLT	Structure containing the limit.
GFLIB_VLOG10_T_FLT	Array of single precision floating point elements for storing the coefficients of the floating point log10 approximation polynomial.
GMCLIB_DECOUPLINGPMSM_T_F16	Structure containing coefficients for calculation of the decoupling.
GMCLIB_DECOUPLINGPMSM_T_F32	Structure containing coefficients for calculation of the decoupling.
GMCLIB_DECOUPLINGPMSM_T_FLT	Structure containing coefficients for calculation of the decoupling.
GMCLIB_ELIMDCBUSRIP_T_F16	Structure containing the PWM modulation index and the measured value of the DC bus voltage.
GMCLIB_ELIMDCBUSRIP_T_F32	Structure containing the PWM modulation index and the measured value of the DC bus voltage.
GMCLIB_ELIMDCBUSRIP_T_FLT	Structure containing the PWM modulation index and the measured value of the DC bus voltage.
SWLIBS_2Syst_F16	Array of two standard 16-bit fractional arguments.
SWLIBS_2Syst_F32	Array of two standard 32-bit fractional arguments.
SWLIBS_2Syst_FLT	Array of two standard single precision floating point arguments.
SWLIBS_3Syst_F16	Array of three standard 16-bit fractional arguments.
SWLIBS_3Syst_F32	Array of three standard 32-bit fractional arguments.
SWLIBS_3Syst_FLT	Array of three standard single precision floating point arguments.
SWLIBS_VERSION_T	Motor Control Library Set identification structure.

7.1 AMCLIB_BEMF_OBSRV_DQ_T_F16

```
#include <AMCLIB_BemfObsrvDQ.h>
```

7.1.1 Description

Observer configuration structure.

7.1.2 Compound Type Members

Table 7-2. 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.
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	Scailing coefficient for current I_frac, fractional format normalized to fit into (-2^{15}, $2^{15}-1$).
tFrac16	f16UGain	Scailing coefficient for voltage U_frac, fractional format normalized to fit into (-2^{15}, $2^{15}-1$).
tFrac16	f16WIGain	Scailing coefficient for angular speed WI_frac, fractional format normalized to fit into (-2^{15}, $2^{15}-1$).
tFrac16	f16EGain	Scailing coefficient for back-EMF E_frac, fractional format normalized to fit into (-2^{15}, $2^{15}-1$).
tS16	s16Shift	Common gain shift, integer format [-14, 14]. Function accuracy guaranteed only for range [-1, 1].

7.2 AMCLIB_BEMF_OBSRV_DQ_T_F32

```
#include <AMCLIB_BemfObsrvDQ.h>
```

7.2.1 Description

Observer configuration structure.

7.2.2 Compound Type Members

Table 7-3. 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
SWLIBS_2Syst_F16	pIObsrvIn_1H	Inputs of RL circuit at step k-1 - high word
tFrac32	f32IGain	Scailing coefficient for current I_frac, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tFrac32	f32UGain	Scailing coefficient for voltage U_frac, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tFrac32	f32WIGain	Scailing coefficient for angular speed WI_frac, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tFrac32	f32EGain	Scailing coefficient for back-EMF E_frac, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.
tS16	s16Shift	Common gain shift, integer format [-14, 14]. Function accuracy guaranteed only for range [-1, 1].

7.3 AMCLIB_BEMF_OBSRV_DQ_T_FLT

```
#include <AMCLIB_BemfObsrvDQ.h>
```

7.3.1 Description

Observer configuration structure.

7.3.2 Compound Type Members

Table 7-4. 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	Scailing coefficient for current I, single precision floating point format.
tFloat	fltUGain	Scailing coefficient for voltage U, single precision floating point format.
tFloat	fltWIGain	Scailing coefficient for angular speed WI, single precision floating point format.
tFloat	fltEGain	Scailing coefficient for back-EMF E, single precision floating point format.

7.4 AMCLIB_TRACK_OBSRV_T_F16

```
#include <AMCLIB_TrackObsrv.h>
```

7.4.1 Description

Structure containing the estimated position, estimated velocity and the algorithm parameters.

7.4.2 Compound Type Members

Table 7-5. 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.

7.5 AMCLIB_TRACK_OBSRV_T_F32

```
#include <AMCLIB_TrackObsrv.h>
```

7.5.1 Description

Structure containing the estimated position, estimated velocity and the algorithm parameters.

7.5.2 Compound Type Members

Table 7-6. 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.

7.6 AMCLIB_TRACK_OBSRV_T_FLT

```
#include <AMCLIB_TrackObsrv.h>
```

7.6.1 Description

Structure containing the estimated position, estimated velocity and the algorithm parameters.

7.6.2 Compound Type Members

Table 7-7. AMCLIB_TRACK_OBSRV_T_FLT members description

Type	Name	Description
GFLIB_CONTROLLER_PIAW_R_T_FLT	pParamPI	Observer PIrAW controller parameters.

Table continues on the next page...

Table 7-7. AMCLIB_TRACK_OBSRV_T_FLT members description (continued)

Type	Name	Description
GFLIB_INTEGRATOR_TR_T_FLT	pParamInteg	Observer integrator parameters.

7.7 GDFLIB_FILTER_IIR1_COEFF_T_F16

```
#include <GDFLIB_FilterIIR1.h>
```

7.7.1 Description

Sub-structure containing filter coefficients.

7.7.2 Compound Type Members

Table 7-8. 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}, $2^{15}-1$).
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}, $2^{15}-1$).

7.8 GDFLIB_FILTER_IIR1_COEFF_T_F32

```
#include <GDFLIB_FilterIIR1.h>
```

7.8.1 Description

Sub-structure containing filter coefficients.

7.8.2 Compound Type Members

Table 7-9. 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).

7.9 GDFLIB_FILTER_IIR1_COEFF_T_FLT

```
#include <GDFLIB_FilterIIR1.h>
```

7.9.1 Description

Sub-structure containing filter coefficients.

7.9.2 Compound Type Members

Table 7-10. 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.
tFloat	fltA1	A1 coefficient of an IIR1 filter. The parameter is in full range single precision floating point format.

7.10 GDFLIB_FILTER_IIR1_T_F16

```
#include <GDFLIB_FilterIIR1.h>
```

7.10.1 Description

Structure containing filter buffer and coefficients.

7.10.2 Compound Type Members

Table 7-11. 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}, $2^{15}-1$).

7.11 GDFLIB_FILTER_IIR1_T_F32

```
#include <GDFLIB_FilterIIR1.h>
```

7.11.1 Description

Structure containing filter buffer and coefficients.

7.11.2 Compound Type Members

Table 7-12. 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}, $2^{31}-1$).

7.12 GDFLIB_FILTER_IIR1_T_FLT

```
#include <GDFLIB_FilterIIR1.h>
```

7.12.1 Description

Structure containing filter buffer and coefficients.

7.12.2 Compound Type Members

Table 7-13. 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.

7.13 GDFLIB_FILTER_IIR2_COEFF_T_F16

```
#include <GDFLIB_FilterIIR2.h>
```

7.13.1 Description

Sub-structure containing filter coefficients.

7.13.2 Compound Type Members

Table 7-14. 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).

7.14 GDFLIB_FILTER_IIR2_COEFF_T_F32

```
#include <GDFLIB_FilterIIR2.h>
```

7.14.1 Description

Sub-structure containing filter coefficients.

7.14.2 Compound Type Members

Table 7-15. 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).

7.15 GDFLIB_FILTER_IIR2_COEFF_T_FLT

```
#include <GDFLIB_FilterIIR2.h>
```

7.15.1 Description

Sub-structure containing filter coefficients.

7.15.2 Compound Type Members

Table 7-16. 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.

7.16 GDFLIB_FILTER_IIR2_T_F16

```
#include <GDFLIB_FilterIIR2.h>
```

7.16.1 Description

Structure containing filter buffer and coefficients.

7.16.2 Compound Type Members

Table 7-17. 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 $(-2^{15}, 2^{15}-1)$.

7.17 GDFLIB_FILTER_IIR2_T_F32

```
#include <GDFLIB_FilterIIR2.h>
```

7.17.1 Description

Structure containing filter buffer and coefficients.

7.17.2 Compound Type Members

Table 7-18. 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 $(-2^{31}, 2^{31}-1)$.

7.18 GDFLIB_FILTER_IIR2_T_FLT

```
#include <GDFLIB_FilterIIR2.h>
```

7.18.1 Description

Structure containing filter buffer and coefficients.

7.18.2 Compound Type Members

Table 7-19. 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.

7.19 GDFLIB_FILTER_MA_T_F16

```
#include <GDFLIB_FilterMA.h>
```

7.19.1 Description

Structure containing filter buffer and coefficients.

7.19.2 Compound Type Members

Table 7-20. GDFLIB_FILTER_MA_T_F16 members description

Type	Name	Description
tFrac32	f32Acc	Filter accumulator.
tU16	u16NSamples	Recalculated smoothing factor [0, 15].

7.20 GDFLIB_FILTER_MA_T_F32

```
#include <GDFLIB_FilterMA.h>
```

7.20.1 Description

Structure containing filter buffer and coefficients.

7.20.2 Compound Type Members

Table 7-21. GDFLIB_FILTER_MA_T_F32 members description

Type	Name	Description
tFrac32	f32Acc	Filter accumulator.
tU16	u16NSamples	Recalculated smoothing factor [0, 31].

7.21 GDFLIB_FILTER_MA_T_FLT

```
#include <GDFLIB_FilterMA.h>
```

7.21.1 Description

Structure containing filter buffer and coefficients.

7.21.2 Compound Type Members

Table 7-22. GDFLIB_FILTER_MA_T_FLT members description

Type	Name	Description
tFloat	fltAcc	Filter accumulator.
tFloat	fltLambda	Smoothing factor [0, 1].

7.22 GDFLIB_FILTERFIR_PARAM_T_F16

```
#include <GDFLIB_FilterFIR.h>
```

7.22.1 Description

Structure containing parameters of the filter.

7.22.2 Compound Type Members

Table 7-23. GDFLIB_FILTERFIR_PARAM_T_F16 members description

Type	Name	Description
tU16	u16Order	FIR filter order, must be 1 or more.
const tFrac16 *	pCoefBuf	FIR filter coefficients buffer.

7.23 GDFLIB_FILTERFIR_PARAM_T_F32

```
#include <GDFLIB_FilterFIR.h>
```

7.23.1 Description

Structure containing parameters of the filter.

7.23.2 Compound Type Members

Table 7-24. GDFLIB_FILTERFIR_PARAM_T_F32 members description

Type	Name	Description
tU32	u32Order	FIR filter order, must be 1 or more.
const tFrac32 *	pCoefBuf	FIR filter coefficients buffer.

7.24 GDFLIB_FILTERFIR_PARAM_T_FLT

```
#include <GDFLIB_FilterFIR.h>
```

7.24.1 Description

Structure containing parameters of the filter.

7.24.2 Compound Type Members

Table 7-25. GDFLIB_FILTERFIR_PARAM_T_FLT members description

Type	Name	Description
tU32	u32Order	FIR filter order, must be 1 or more.
const tFloat *	pCoefBuf	FIR filter coefficients buffer.

7.25 GDFLIB_FILTERFIR_STATE_T_F16

```
#include <GDFLIB_FilterFIR.h>
```

7.25.1 Description

Structure containing the current state of the filter.

7.25.2 Compound Type Members

Table 7-26. GDFLIB_FILTERFIR_STATE_T_F16 members description

Type	Name	Description
tU16	u16Idx	Input buffer index.
tFrac16 *	pInBuf	Pointer to the input buffer.

7.26 GDFLIB_FILTERFIR_STATE_T_F32

```
#include <GDFLIB_FilterFIR.h>
```

7.26.1 Description

Structure containing the current state of the filter.

7.26.2 Compound Type Members

Table 7-27. GDFLIB_FILTERFIR_STATE_T_F32 members description

Type	Name	Description
tU32	u32Idx	Input buffer index.
tFrac32 *	pInBuf	Pointer to the input buffer.

7.27 GDFLIB_FILTERFIR_STATE_T_FLT

```
#include <GDFLIB_FilterFIR.h>
```

7.27.1 Description

Structure containing the current state of the filter.

7.27.2 Compound Type Members

Table 7-28. GDFLIB_FILTERFIR_STATE_T_FLT members description

Type	Name	Description
tU32	u32Idx	Input buffer index.
tFloat *	pInBuf	Pointer to the input buffer.

7.28 GFLIB_ACOS_T_F16

```
#include <GFLIB_Acos.h>
```

7.28.1 Description

Default approximation coefficients datatype for arccosine approximation.

7.28.2 Compound Type Members

Table 7-29. 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.

7.29 GFLIB_ACOS_T_F32

```
#include <GFLIB_Acos.h>
```

7.29.1 Description

Default approximation coefficients datatype for arccosine approximation.

7.29.2 Compound Type Members

Table 7-30. 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.

7.30 GFLIB_ACOS_T_FLT

```
#include <GFLIB_Acos.h>
```

7.30.1 Description

Default approximation coefficients datatype for arccosine approximation.

7.30.2 Compound Type Members

Table 7-31. GFLIB_ACOS_T_FLT members description

Type	Name	Description
const tFloat	fltA	Array of approximation coefficients.

7.31 GFLIB_ACOS_TAYLOR_COEF_T_F16

```
#include <GFLIB_Acos.h>
```

7.31.1 Description

Array of approximation coefficients for piece-wise polynomial.

7.31.2 Compound Type Members

Table 7-32. 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.

7.32 GFLIB_ACOS_TAYLOR_COEF_T_F32

```
#include <GFLIB_Acos.h>
```

7.32.1 Description

Array of approximation coefficients for piece-wise polynomial.

7.32.2 Compound Type Members

Table 7-33. 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.

7.33 GFLIB_ASIN_T_F16

```
#include <GFLIB_Asin.h>
```

7.33.1 Description

Default approximation coefficients datatype for arcsine approximation.

7.33.2 Compound Type Members

Table 7-34. 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.

7.34 GFLIB_ASIN_T_F32

```
#include <GFLIB_Asin.h>
```

7.34.1 Description

Default approximation coefficients datatype for arcsine approximation.

7.34.2 Compound Type Members

Table 7-35. 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.

7.35 GFLIB_ASIN_T_FLT

```
#include <GFLIB_Asin.h>
```

7.35.1 Description

Default approximation coefficients datatype for arcsine approximation.

7.35.2 Compound Type Members

Table 7-36. GFLIB_ASIN_T_FLT members description

Type	Name	Description
const tFloat	fltA	Default approximation coefficients datatype for arcsine approximation.

7.36 GFLIB_ASIN_TAYLOR_COEF_T_F16

```
#include <GFLIB_Asin.h>
```

7.36.1 Description

Array of approximation coefficients for piece-wise polynomial.

7.36.2 Compound Type Members

Table 7-37. GFLIB_ASIN_TAYLOR_COEF_T_F16 members description

Type	Name	Description
const tFrac16	f16A	Array of approximation coefficients for piece-wise polynomial.

7.37 GFLIB_ASIN_TAYLOR_COEF_T_F32

```
#include <GFLIB_Asin.h>
```

7.37.1 Description

Array of approximation coefficients for piece-wise polynomial.

7.37.2 Compound Type Members

Table 7-38. 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.

7.38 GFLIB_ATAN_T_F16

```
#include <GFLIB_Atan.h>
```

7.38.1 Description

Structure containing eight sub-structures with polynomial coefficients to cover all sub-intervals.

7.38.2 Compound Type Members

Table 7-39. 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.

7.39 GFLIB_ATAN_T_F32

```
#include <GFLIB_Atan.h>
```

7.39.1 Description

Structure containing eight sub-structures with polynomial coefficients to cover all sub-intervals.

7.39.2 Compound Type Members

Table 7-40. 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.

7.40 GFLIB_ATAN_T_FLT

```
#include <GFLIB_Atan.h>
```

7.40.1 Description

Structure containing the approximation coefficients.

7.40.2 Compound Type Members

Table 7-41. GFLIB_ATAN_T_FLT members description

Type	Name	Description
const tFloat	fltA	Structure containing the approximation coefficients.

7.41 GFLIB_ATAN_TAYLOR_COEF_T_F16

```
#include <GFLIB_Atan.h>
```

7.41.1 Description

Array of polynomial approximation coefficients for one sub-interval.

7.41.2 Compound Type Members

Table 7-42. GFLIB_ATAN_TAYLOR_COEF_T_F16 members description

Type	Name	Description
const tFrac16	f16A	Array of polynomial approximation coefficients for one sub-interval.

7.42 GFLIB_ATAN_TAYLOR_COEF_T_F32

```
#include <GFLIB_Atan.h>
```

7.42.1 Description

Array of minimax polynomial approximation coefficients for one sub-interval.

7.42.2 Compound Type Members

Table 7-43. GFLIB_ATAN_TAYLOR_COEF_T_F32 members description

Type	Name	Description
const tFrac32	f32A	Array of minimax polynomial approximation coefficients for one sub-interval.

7.43 GFLIB_ATANYXSHIFTED_T_F16

```
#include <GFLIB_AtanYXShifted.h>
```

7.43.1 Description

Structure containing the parameter for the AtanYXShifted function.

7.43.2 Compound Type Members

Table 7-44. 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.

7.44 GFLIB_ATANYXSHIFTED_T_F32

```
#include <GFLIB_AtanYXShifted.h>
```

7.44.1 Description

Structure containing the parameter for the AtanYXShifted function.

7.44.2 Compound Type Members

Table 7-45. 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.

7.45 GFLIB_ATANYXSHIFTED_T_FLT

```
#include <GFLIB_AtanYXShifted.h>
```

7.45.1 Description

Structure containing the parameter for the AtanYXShifted function.

7.45.2 Compound Type Members

Table 7-46. 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.

7.46 GFLIB_CONTROLLER_PI_P_T_F16

```
#include <GFLIB_ControllerPIp.h>
```

7.46.1 Description

Structure containing parameters and states of the parallel form PI controller.

7.46.2 Compound Type Members

Table 7-47. 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.

7.47 GFLIB_CONTROLLER_PI_P_T_F32

```
#include <GFLIB_ControllerPIp.h>
```

7.47.1 Description

Structure containing parameters and states of the parallel form PI controller.

7.47.2 Compound Type Members

Table 7-48. 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.

7.48 GFLIB_CONTROLLER_PI_P_T_FLT

```
#include <GFLIB_ControllerPIp.h>
```

7.48.1 Description

Structure containing parameters and states of the parallel form PI controller.

7.48.2 Compound Type Members

Table 7-49. 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.

7.49 GFLIB_CONTROLLER_PI_R_T_F16

```
#include <GFLIB_ControllerPIr.h>
```

7.49.1 Description

Structure containing parameters and states of the recurrent form PI controller.

7.49.2 Compound Type Members

Table 7-50. 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	Internal controller accumulator.
tFrac16	f16InErrK1	Controller input from the previous calculation step.
tU16	u16NShift	Scaling factor for the controller coefficients, integer format [0, 15].

7.50 GFLIB_CONTROLLER_PI_R_T_F32

```
#include <GFLIB_ControllerPIr.h>
```

7.50.1 Description

Structure containing parameters and states of the recurrent form PI controller.

7.50.2 Compound Type Members

Table 7-51. 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	Internal controller accumulator.
tFrac32	f32InErrK1	Controller input from the previous calculation step.
tU16	u16NShift	Scaling factor for the controller coefficients, integer format [0, 31].

7.51 GFLIB_CONTROLLER_PI_R_T_FLT

```
#include <GFLIB_ControllerPIr.h>
```

7.51.1 Description

Structure containing parameters and states of the recurrent form PI controller.

7.51.2 Compound Type Members

Table 7-52. 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	Internal controller accumulator, single precision floating point format.
tFloat	fltInErrK1	Controller input from the previous calculation step, single precision floating point format.

7.52 GFLIB_CONTROLLER_PIAW_P_T_F16

```
#include <GFLIB_ControllerPIpAW.h>
```

7.52.1 Description

Structure containing parameters and states of the parallel form PI controller with anti-windup.

7.52.2 Compound Type Members

Table 7-53. 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.

7.53 GFLIB_CONTROLLER_PIAW_P_T_F32

```
#include <GFLIB_ControllerPIpAW.h>
```

7.53.1 Description

Structure containing parameters and states of the parallel form PI controller with anti-windup.

7.53.2 Compound Type Members

Table 7-54. GFLIB_CONTROLLER_PIAW_P_T_F32 members description

Type	Name	Description
tFrac32	f32PropGain	Proportional Gain, fractional format normalized to fit into $(-2^{31}, 2^{31}-1)$.

Table continues on the next page...

Table 7-54. GFLIB_CONTROLLER_PIAW_P_T_F32 members description (continued)

Type	Name	Description
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.

7.54 GFLIB_CONTROLLER_PIAW_P_T_FLT

```
#include <GFLIB_ControllerPIpAW.h>
```

7.54.1 Description

Structure containing parameters and states of the parallel form PI controller with anti-windup.

7.54.2 Compound Type Members

Table 7-55. GFLIB_CONTROLLER_PIAW_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	fltLowerLimit	Lower Limit of the controller, single precision floating point format.
tFloat	fltUpperLimit	Upper Limit of the controller, single precision floating point format.

Table continues on the next page...

Table 7-55. GFLIB_CONTROLLER_PIAW_P_T_FLT members description (continued)

Type	Name	Description
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.

7.55 GFLIB_CONTROLLER_PIAW_R_T_F16

```
#include <GFLIB_ControllerPIrAW.h>
```

7.55.1 Description

Structure containing parameters and states of the recurrent form PI controller with anti-windup.

7.55.2 Compound Type Members

Table 7-56. 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	Internal controller accumulator.
tFrac16	f16InErrK1	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 factor for the controller coefficients, integer format [0, 15].

7.56 GFLIB_CONTROLLER_PIAW_R_T_F32

```
#include <GFLIB_ControllerPIrAW.h>
```

7.56.1 Description

Structure containing parameters and states of the recurrent form PI controller with anti-windup.

7.56.2 Compound Type Members

Table 7-57. 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	Internal controller accumulator.
tFrac32	f32InErrK1	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 factor for the controller coefficients, integer format [0, 31].

7.57 GFLIB_CONTROLLER_PIAW_R_T_FLT

```
#include <GFLIB_ControllerPIrAW.h>
```

7.57.1 Description

Structure containing parameters and states of the recurrent form PI controller with anti-windup.

7.57.2 Compound Type Members

Table 7-58. 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.
tFloat	fltAcc	Internal controller accumulator, single precision floating point format.
tFloat	fltInErrK1	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.

7.58 GFLIB_COS_T_F16

```
#include <GFLIB_Cos.h>
```

7.58.1 Description

Array of four 16-bit elements for storing coefficients of the Taylor polynomial.

7.58.2 Compound Type Members

Table 7-59. GFLIB_COS_T_F16 members description

Type	Name	Description
tFrac16	f16A	Array of four 16-bit elements for storing coefficients of the Taylor polynomial.

7.59 GFLIB_COS_T_F32

```
#include <GFLIB_Cos.h>
```

7.59.1 Description

Array of five 32-bit elements for storing coefficients of the Taylor polynomial.

7.59.2 Compound Type Members

Table 7-60. GFLIB_COS_T_F32 members description

Type	Name	Description
tFrac32	f32A	Array of five 32-bit elements for storing coefficients of the Taylor polynomial.

7.60 GFLIB_COS_T_FLT

```
#include <GFLIB_Cos.h>
```

7.60.1 Description

Array of three single precision floating point elements for storing coefficients of the floating point optimized minimax approximation polynomial.

7.60.2 Compound Type Members

Table 7-61. 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.

7.61 GFLIB_HYST_T_F16

```
#include <GFLIB_Hyst.h>
```

7.61.1 Description

Structure containing parameters and states for the hysteresis function.

7.61.2 Compound Type Members

Table 7-62. 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 higher than the upper threshold.
tFrac16	f16OutState	Actual state of the output.

7.62 GFLIB_HYST_T_F32

```
#include <GFLIB_Hyst.h>
```

7.62.1 Description

Structure containing parameters and states for the hysteresis function.

7.62.2 Compound Type Members

Table 7-63. 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 higher than the upper threshold.
tFrac32	f32OutState	Actual state of the output.

7.63 GFLIB_HYST_T_FLT

```
#include <GFLIB_Hyst.h>
```

7.63.1 Description

Structure containing parameters and states for the hysteresis function.

7.63.2 Compound Type Members

Table 7-64. 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 higher than the upper threshold.
tFloat	fltOutState	Actual state of the output.

7.64 GFLIB_INTEGRATOR_TR_T_F16

```
#include <GFLIB_IntegratorTR.h>
```

7.64.1 Description

Structure containing integrator parameters and coefficients.

7.64.2 Compound Type Members

Table 7-65. GFLIB_INTEGRATOR_TR_T_F16 members description

Type	Name	Description
tFrac32	f32State	Integrator state value.

Table continues on the next page...

Table 7-65. GFLIB_INTEGRATOR_TR_T_F16 members description (continued)

Type	Name	Description
tFrac16	f16InK1	Input value in step k-1.
tFrac16	f16C1	Integrator coefficient = $(E_{MAX}/T_s)(U_{MAX}*2)*(2^{-u16NShift})$.
tU16	u16NShift	Scaling factor for the integrator coefficient f16C1, integer format [0, 15].

7.65 GFLIB_INTEGRATOR_TR_T_F32

```
#include <GFLIB_IntegratorTR.h>
```

7.65.1 Description

Structure containing integrator parameters and coefficients.

7.65.2 Compound Type Members

Table 7-66. GFLIB_INTEGRATOR_TR_T_F32 members description

Type	Name	Description
tFrac32	f32State	Integrator state value.
tFrac32	f32InK1	Input value in step k-1.
tFrac32	f32C1	Integrator coefficient = $(E_{MAX}/T_s)(U_{MAX}*2)*(2^{-u16NShift})$.
tU16	u16NShift	Scaling factor for the integrator coefficient f32C1, integer format [0, 15].

7.66 GFLIB_INTEGRATOR_TR_T_FLT

```
#include <GFLIB_IntegratorTR.h>
```

7.66.1 Description

Structure containing integrator parameters and coefficients.

7.66.2 Compound Type Members

Table 7-67. GFLIB_INTEGRATOR_TR_T_FLT members description

Type	Name	Description
tFloat	fltState	Integrator state value, single precision floating point format.
tFloat	fltInK1	Input value in step k-1, single precision floating point format.
tFloat	fltC1	Integrator coefficient, single precision floating point format.

7.67 GFLIB_LIMIT_T_F16

```
#include <GFLIB_Limit.h>
```

7.67.1 Description

Structure containing the limits.

7.67.2 Compound Type Members

Table 7-68. 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.

7.68 GFLIB_LIMIT_T_F32

```
#include <GFLIB_Limit.h>
```

7.68.1 Description

Structure containing the limits.

7.68.2 Compound Type Members

Table 7-69. 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.

7.69 GFLIB_LIMIT_T_FLT

```
#include <GFLIB_Limit.h>
```

7.69.1 Description

Structure containing the limits.

7.69.2 Compound Type Members

Table 7-70. 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.

7.70 GFLIB_LOG10_T_FLT

```
#include <GFLIB_Log10.h>
```

7.70.1 Description

Array of single precision floating point elements for storing the coefficients of the floating point log10 approximation polynomial.

7.70.2 Compound Type Members

Table 7-71. 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.

7.71 GFLIB_LOWERLIMIT_T_F16

```
#include <GFLIB_LowerLimit.h>
```

7.71.1 Description

Structure containing the lower limit.

7.71.2 Compound Type Members

Table 7-72. GFLIB_LOWERLIMIT_T_F16 members description

Type	Name	Description
tFrac16	f16LowerLimit	Value determining the lower limit threshold.

7.72 GFLIB_LOWERLIMIT_T_F32

```
#include <GFLIB_LowerLimit.h>
```

7.72.1 Description

Structure containing the lower limit.

7.72.2 Compound Type Members

Table 7-73. GFLIB_LOWERLIMIT_T_F32 members description

Type	Name	Description
tFrac32	f32LowerLimit	Value determining the lower limit threshold.

7.73 GFLIB_LOWERLIMIT_T_FLT

```
#include <GFLIB_LowerLimit.h>
```

7.73.1 Description

Structure containing the lower limit.

7.73.2 Compound Type Members

Table 7-74. GFLIB_LOWERLIMIT_T_FLT members description

Type	Name	Description
tFloat	fltLowerLimit	Value determining the lower limit threshold.

7.74 GFLIB_LUT1D_T_F16

```
#include <GFLIB_Lut1D.h>
```

7.74.1 Description

Structure containing 1D look-up table parameters.

7.74.2 Compound Type Members

Table 7-75. GFLIB_LUT1D_T_F16 members description

Type	Name	Description
tU16	u16ShamOffset	Shift amount for extracting the fractional offset within an interpolated interval.
const tFrac16 *	pf16Table	Table holding ordinate values of interpolating intervals.

7.75 GFLIB_LUT1D_T_F32

```
#include <GFLIB_Lut1D.h>
```

7.75.1 Description

Structure containing 1D look-up table parameters.

7.75.2 Compound Type Members

Table 7-76. GFLIB_LUT1D_T_F32 members description

Type	Name	Description
tU32	u32ShamOffset	Shift amount for extracting the fractional offset within an interpolated interval.
const tFrac32 *	pf32Table	Table holding ordinate values of interpolating intervals.

7.76 GFLIB_LUT1D_T_FLT

```
#include <GFLIB_Lut1D.h>
```

7.76.1 Description

Structure containing 1D look-up table parameters.

7.76.2 Compound Type Members

Table 7-77. GFLIB_LUT1D_T_FLT members description

Type	Name	Description
tU32	u32ShamOffset	Shift amount for extracting the fractional offset within an interpolated interval.
const tFloat *	pfltTable	Table holding ordinate values of interpolating intervals. The ordinate values are in full range single precision floating point format.

7.77 GFLIB_LUT2D_T_F16

```
#include <GFLIB_Lut2D.h>
```

7.77.1 Description

Structure containing 2D look-up table parameters.

7.77.2 Compound Type Members

Table 7-78. GFLIB_LUT2D_T_F16 members description

Type	Name	Description
tU16	u16ShamOffset1	Shift amount for extracting the fractional offset within an interpolated interval.
tU16	u16ShamOffset2	Shift amount for extracting the fractional offset within an interpolated interval.
const tFrac16 *	pf16Table	Table holding ordinate values of interpolating intervals.

7.78 GFLIB_LUT2D_T_F32

```
#include <GFLIB_Lut2D.h>
```

7.78.1 Description

Structure containing 2D look-up table parameters.

7.78.2 Compound Type Members

Table 7-79. GFLIB_LUT2D_T_F32 members description

Type	Name	Description
tU32	u32ShamOffset1	Shift amount for extracting the fractional offset within an interpolated interval.
tU32	u32ShamOffset2	Shift amount for extracting the fractional offset within an interpolated interval.
const tFrac32 *	pf32Table	Table holding ordinate values of interpolating intervals.

7.79 GFLIB_LUT2D_T_FLT

```
#include <GFLIB_Lut2D.h>
```

7.79.1 Description

Structure containing 2D look-up table parameters.

7.79.2 Compound Type Members

Table 7-80. GFLIB_LUT2D_T_FLT members description

Type	Name	Description
tU32	u32ShamOffset1	Shift amount for extracting the fractional offset within an interpolated interval.
tU32	u32ShamOffset2	Shift amount for extracting the fractional offset within an interpolated interval.
const tFloat *	pfltTable	Table holding ordinate values of interpolating intervals. The ordinate values are in full range single precision floating point format.

7.80 GFLIB_RAMP_T_F16

```
#include <GFLIB_Ramp.h>
```

7.80.1 Description

Structure containing increment/decrement coefficients and state value for the ramp function implemented in GFLIB_Ramp.

7.80.2 Compound Type Members

Table 7-81. GFLIB_RAMP_T_F16 members description

Type	Name	Description
tFrac16	f16State	Ramp state value.
tFrac16	f16RampUp	Ramp up increment coefficient.
tFrac16	f16RampDown	Ramp down increment (decrement) coefficient.

7.81 GFLIB_RAMP_T_F32

```
#include <GFLIB_Ramp.h>
```

7.81.1 Description

Structure containing increment/decrement coefficients and state value for the ramp function implemented in GFLIB_Ramp.

7.81.2 Compound Type Members

Table 7-82. GFLIB_RAMP_T_F32 members description

Type	Name	Description
tFrac32	f32State	Ramp state value.
tFrac32	f32RampUp	Ramp up increment coefficient.
tFrac32	f32RampDown	Ramp down increment (decrement) coefficient.

7.82 GFLIB_RAMP_T_FLT

```
#include <GFLIB_Ramp.h>
```

7.82.1 Description

Structure containing increment/decrement coefficients and state value for the ramp function implemented in GFLIB_Ramp.

7.82.2 Compound Type Members

Table 7-83. GFLIB_RAMP_T_FLT members description

Type	Name	Description
tFloat	fltState	Ramp state value.
tFloat	fltRampUp	Ramp up increment coefficient.
tFloat	fltRampDown	Ramp down increment (decrement) coefficient.

7.83 GFLIB_SIN_T_F16

```
#include <GFLIB_Sin.h>
```

7.83.1 Description

Array of four 16-bit elements for storing coefficients of the Taylor polynomial.

7.83.2 Compound Type Members

Table 7-84. GFLIB_SIN_T_F16 members description

Type	Name	Description
tFrac16	f16A	Array of four 16-bit elements for storing coefficients of the Taylor polynomial.

7.84 GFLIB_SIN_T_F32

```
#include <GFLIB_Sin.h>
```

7.84.1 Description

Array of five 32-bit elements for storing coefficients of the Taylor polynomial.

7.84.2 Compound Type Members

Table 7-85. GFLIB_SIN_T_F32 members description

Type	Name	Description
tFrac32	f32A	Array of five 32-bit elements for storing coefficients of the Taylor polynomial.

7.85 GFLIB_SIN_T_FLT

```
#include <GFLIB_Sin.h>
```

7.85.1 Description

Array of three single precision floating point elements for storing coefficients of the floating point optimized minimax approximation polynomial.

7.85.2 Compound Type Members

Table 7-86. 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.

7.86 GFLIB_SINCOS_T_F16

```
#include <GFLIB_SinCos.h>
```

7.86.1 Description

Array of four 16-bit elements for storing coefficients of the Taylor polynomial.

7.86.2 Compound Type Members

Table 7-87. GFLIB_SINCOS_T_F16 members description

Type	Name	Description
tFrac16	f16A	Array of four 16-bit elements for storing coefficients of the Taylor polynomial.

7.87 GFLIB_SINCOS_T_F32

```
#include <GFLIB_SinCos.h>
```

7.87.1 Description

Array of five 32-bit elements for storing coefficients of the Taylor polynomial.

7.87.2 Compound Type Members

Table 7-88. GFLIB_SINCOS_T_F32 members description

Type	Name	Description
tFrac32	f32A	Array of five 32-bit elements for storing coefficients of the Taylor polynomial.

7.88 GFLIB_SINCOS_T_FLT

```
#include <GFLIB_SinCos.h>
```

7.88.1 Description

Array of three single precision floating point elements for storing coefficients of the floating point optimized minimax approximation polynomial.

7.88.2 Compound Type Members

Table 7-89. 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.

7.89 GFLIB_TAN_T_F16

```
#include <GFLIB_Tan.h>
```

7.89.1 Description

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.

7.89.2 Compound Type Members

Table 7-90. 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

Table 7-90. GFLIB_TAN_T_F16 members description

Type	Name	Description
		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.

7.90 GFLIB_TAN_T_F32

```
#include <GFLIB_Tan.h>
```

7.90.1 Description

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.

7.90.2 Compound Type Members

Table 7-91. 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.

7.91 GFLIB_TAN_T_FLT

```
#include <GFLIB_Tan.h>
```

7.91.1 Description

Polynomial coefficient for fractional approximation in single precision floating point format.

7.91.2 Compound Type Members

Table 7-92. GFLIB_TAN_T_FLT members description

Type	Name	Description
tFloat	fltA	Polynomial coefficient for fractional approximation in single precision floating point format.

7.92 GFLIB_TAN_TAYLOR_COEF_T_F16

```
#include <GFLIB_Tan.h>
```

7.92.1 Description

Structure containing four polynomial coefficients for one sub-interval.

7.92.2 Compound Type Members

Table 7-93. GFLIB_TAN_TAYLOR_COEF_T_F16 members description

Type	Name	Description
const tFrac16	f16A	Structure containing four polynomial coefficients for one sub-interval.

7.93 GFLIB_TAN_TAYLOR_COEF_T_F32

```
#include <GFLIB_Tan.h>
```

7.93.1 Description

Structure containing four polynomial coefficients for one sub-interval.

7.93.2 Compound Type Members

Table 7-94. GFLIB_TAN_TAYLOR_COEF_T_F32 members description

Type	Name	Description
const tFrac32	f32A	Structure containing four polynomial coefficients for one sub-interval.

7.94 GFLIB_UPPERLIMIT_T_F16

```
#include <GFLIB_UpperLimit.h>
```

7.94.1 Description

Structure containing the upper limit.

7.94.2 Compound Type Members

Table 7-95. GFLIB_UPPERLIMIT_T_F16 members description

Type	Name	Description
tFrac16	f16UpperLimit	Value determining the upper limit threshold.

7.95 GFLIB_UPPERLIMIT_T_F32

```
#include <GFLIB_UpperLimit.h>
```

7.95.1 Description

Structure containing the upper limit.

7.95.2 Compound Type Members

Table 7-96. GFLIB_UPPERLIMIT_T_F32 members description

Type	Name	Description
tFrac32	f32UpperLimit	Value determining the upper limit threshold.

7.96 GFLIB_UPPERLIMIT_T_FLT

```
#include <GFLIB_UpperLimit.h>
```

7.96.1 Description

Structure containing the upper limit.

7.96.2 Compound Type Members

Table 7-97. GFLIB_UPPERLIMIT_T_FLT members description

Type	Name	Description
tFloat	fltUpperLimit	Value determining the upper limit threshold.

7.97 GFLIB_VECTORLIMIT_T_F16

```
#include <GFLIB_VectorLimit.h>
```

7.97.1 Description

Structure containing the limit.

7.97.2 Compound Type Members

Table 7-98. 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.

7.98 GFLIB_VECTORLIMIT_T_F32

```
#include <GFLIB_VectorLimit.h>
```

7.98.1 Description

Structure containing the limit.

7.98.2 Compound Type Members

Table 7-99. 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.

7.99 GFLIB_VECTORLIMIT_T_FLT

```
#include <GFLIB_VectorLimit.h>
```

7.99.1 Description

Structure containing the limit.

7.99.2 Compound Type Members

Table 7-100. 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.

7.100 GFLIB_VLOG10_T_FLT

```
#include <GFLIB_VLog10.h>
```

7.100.1 Description

Array of single precision floating point elements for storing the coefficients of the floating point log10 approximation polynomial.

7.100.2 Compound Type Members

Table 7-101. 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.

7.101 GMCLIB_DECOUPLINGPMSM_T_F16

```
#include <GMCLIB_DecouplingPMSM.h>
```

7.101.1 Description

Structure containing coefficients for calculation of the decoupling.

7.101.2 Compound Type Members

Table 7-102. 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 $_{q\_shift}$.

7.102 GMCLIB_DECOUPLINGPMSM_T_F32

```
#include <GMCLIB_DecouplingPMSM.h>
```

7.102.1 Description

Structure containing coefficients for calculation of the decoupling.

7.102.2 Compound Type Members

Table 7-103. 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 $_{q\_shift}$.

7.103 GMCLIB_DECOUPLINGPMSM_T_FLT

```
#include <GMCLIB_DecouplingPMSM.h>
```

7.103.1 Description

Structure containing coefficients for calculation of the decoupling.

7.103.2 Compound Type Members

Table 7-104. GMCLIB_DECOUPLINGPMSM_T_FLT members description

Type	Name	Description
tFloat	fltLD	L_D inductance [H].
tFloat	fltLQ	L_Q inductance [H].

7.104 GMCLIB_ELIMDCBUSRIP_T_F16

```
#include <GMCLIB_ElimDcBusRip.h>
```

7.104.1 Description

Structure containing the PWM modulation index and the measured value of the DC bus voltage.

7.104.2 Compound Type Members

Table 7-105. GMCLIB_ELIMDCBUSRIP_T_F16 members description

Type	Name	Description
tFrac16	f16ModIndex	Inverse Modulation Index.
tFrac16	f16ArgDcBusMsr	Measured DC bus voltage.

7.105 GMCLIB_ELIMDCBUSRIP_T_F32

```
#include <GMCLIB_ElimDcBusRip.h>
```

7.105.1 Description

Structure containing the PWM modulation index and the measured value of the DC bus voltage.

7.105.2 Compound Type Members

Table 7-106. GMCLIB_ELIMDCBUSRIP_T_F32 members description

Type	Name	Description
tFrac32	f32ModIndex	Inverse Modulation Index.
tFrac32	f32ArgDcBusMsr	Measured DC bus voltage.

7.106 GMCLIB_ELIMDCBUSRIP_T_FLT

```
#include <GMCLIB_ElimDcBusRip.h>
```

7.106.1 Description

Structure containing the PWM modulation index and the measured value of the DC bus voltage.

7.106.2 Compound Type Members

Table 7-107. GMCLIB_ELIMDCBUSRIP_T_FLT members description

Type	Name	Description
tFloat	fltModIndex	Inverse Modulation Index.
tFloat	fltArgDcBusMsr	Measured DC bus voltage.

7.107 SWLIBS_2Syst_F16

```
#include <SWLIBS_Typedefs.h>
```

7.107.1 Description

Array of two standard 16-bit fractional arguments.

7.107.2 Compound Type Members

Table 7-108. SWLIBS_2Syst_F16 members description

Type	Name	Description
tFrac16	f16Arg1	First argument
tFrac16	f16Arg2	Second argument

7.108 SWLIBS_2Syst_F32

```
#include <SWLIBS_Typedefs.h>
```

7.108.1 Description

Array of two standard 32-bit fractional arguments.

7.108.2 Compound Type Members

Table 7-109. SWLIBS_2Syst_F32 members description

Type	Name	Description
tFrac32	f32Arg1	First argument
tFrac32	f32Arg2	Second argument

7.109 SWLIBS_2Syst_FLT

```
#include <SWLIBS_Typedefs.h>
```

7.109.1 Description

Array of two standard single precision floating point arguments.

7.109.2 Compound Type Members

Table 7-110. SWLIBS_2Syst_FLT members description

Type	Name	Description
tFloat	fltArg1	First argument
tFloat	fltArg2	Second argument

7.110 SWLIBS_3Syst_F16

```
#include <SWLIBS_Typedefs.h>
```

7.110.1 Description

Array of three standard 16-bit fractional arguments.

7.110.2 Compound Type Members

Table 7-111. SWLIBS_3Syst_F16 members description

Type	Name	Description
tFrac16	f16Arg1	First argument
tFrac16	f16Arg2	Second argument
tFrac16	f16Arg3	Third argument

7.111 SWLIBS_3Syst_F32

```
#include <SWLIBS_Typedefs.h>
```

7.111.1 Description

Array of three standard 32-bit fractional arguments.

7.111.2 Compound Type Members

Table 7-112. SWLIBS_3Syst_F32 members description

Type	Name	Description
tFrac32	f32Arg1	First argument
tFrac32	f32Arg2	Second argument
tFrac32	f32Arg3	Third argument

7.112 SWLIBS_3Syst_FLT

```
#include <SWLIBS_Typedefs.h>
```

7.112.1 Description

Array of three standard single precision floating point arguments.

7.112.2 Compound Type Members

Table 7-113. SWLIBS_3Syst_FLT members description

Type	Name	Description
tFloat	fltArg1	First argument
tFloat	fltArg2	Second argument
tFloat	fltArg3	Third argument

7.113 SWLIBS_VERSION_T

```
#include <SWLIBS_Version.h>
```

7.113.1 Description

Motor Control Library Set identification structure.

7.113.2 Compound Type Members

Table 7-114. 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

Chapter 8

8.1 Macro Definitions

8.1.1 Macro Definitions Overview

```
#define AMCLIB_BEMF_OBSRV_DQ_DEFAULT_F16

#define AMCLIB_BEMF_OBSRV_DQ_DEFAULT_F32

#define AMCLIB_BEMF_OBSRV_DQ_DEFAULT_FLT

#define AMCLIB_BEMF_OBSRV_DQ_T

#define AMCLIB_BEMF_OBSRV_DQ_T

#define AMCLIB_BEMF_OBSRV_DQ_T

#define AMCLIB_BemfObsrvDQ

#define AMCLIB_BemfObsrvDQInit

#define AMCLIB_TRACK_OBSRV_DEFAULT

#define AMCLIB_TRACK_OBSRV_DEFAULT

#define AMCLIB_TRACK_OBSRV_DEFAULT

#define AMCLIB_TRACK_OBSRV_DEFAULT_F16

#define AMCLIB_TRACK_OBSRV_DEFAULT_F32

#define AMCLIB_TRACK_OBSRV_DEFAULT_FLT

#define AMCLIB_TRACK_OBSRV_T

#define AMCLIB_TRACK_OBSRV_T

#define AMCLIB_TRACK_OBSRV_T
```

```
#define AMCLIB_TrackObsrv

#define AMCLIB_TrackObsrvInit

#define F16

#define F16TOINT16

#define F16TOINT32

#define F16TOINT64

#define F16_1_DIVBY_SQRT3

#define F16_SQRT2_DIVBY_2

#define F16_SQRT3_DIVBY_2

#define F32

#define F32TOINT16

#define F32TOINT32

#define F32TOINT64

#define F32_1_DIVBY_SQRT3

#define F32_SQRT2_DIVBY_2

#define F32_SQRT3_DIVBY_2

#define F64TOINT16

#define F64TOINT32

#define F64TOINT64

#define FALSE

#define FLOAT_0_5

#define FLOAT_2_PI

#define FLOAT_4_DIVBY_PI

#define FLOAT_DIVBY_SQRT3

#define FLOAT_MAX

#define FLOAT_MIN

#define FLOAT_MINUS_0_5
```

```
#define FLOAT_MINUS_1

#define FLOAT_MIN_NORM

#define FLOAT_PI

#define FLOAT_PI_CORRECTION

#define FLOAT_PI_DIVBY_2

#define FLOAT_PI_DIVBY_4

#define FLOAT_PI_DIVBY_6

#define FLOAT_PI_SINGLE_CORRECTION

#define FLOAT_PLUS_1

#define FLOAT_SQRT3_DIVBY_2

#define FLOAT_SQRT3_DIVBY_4

#define FLOAT_SQRT3_DIVBY_4_CORRECTION

#define FLOAT_TAN_PI_DIVBY_12

#define FLOAT_TAN_PI_DIVBY_6

#define FLT

#define FRAC16

#define FRAC16_0_25

#define FRAC16_0_5

#define FRAC32

#define FRAC32_0_25

#define FRAC32_0_5

#define FRACT_MAX

#define FRACT_MIN

#define GDFLIB_FILTERFIR_PARAM_T

#define GDFLIB_FILTERFIR_PARAM_T

#define GDFLIB_FILTERFIR_PARAM_T

#define GDFLIB_FILTERFIR_STATE_T
```

```
#define GDFLIB_FILTERFIR_STATE_T

#define GDFLIB_FILTERFIR_STATE_T

#define GDFLIB_FILTER_IIR1_DEFAULT

#define GDFLIB_FILTER_IIR1_DEFAULT

#define GDFLIB_FILTER_IIR1_DEFAULT

#define GDFLIB_FILTER_IIR1_DEFAULT_F16

#define GDFLIB_FILTER_IIR1_DEFAULT_F32

#define GDFLIB_FILTER_IIR1_DEFAULT_FLT

#define GDFLIB_FILTER_IIR1_T

#define GDFLIB_FILTER_IIR1_T

#define GDFLIB_FILTER_IIR1_T

#define GDFLIB_FILTER_IIR2_DEFAULT

#define GDFLIB_FILTER_IIR2_DEFAULT

#define GDFLIB_FILTER_IIR2_DEFAULT

#define GDFLIB_FILTER_IIR2_DEFAULT_F16

#define GDFLIB_FILTER_IIR2_DEFAULT_F32

#define GDFLIB_FILTER_IIR2_DEFAULT_FLT

#define GDFLIB_FILTER_IIR2_T

#define GDFLIB_FILTER_IIR2_T

#define GDFLIB_FILTER_IIR2_T

#define GDFLIB_FILTER_MA_DEFAULT

#define GDFLIB_FILTER_MA_DEFAULT

#define GDFLIB_FILTER_MA_DEFAULT

#define GDFLIB_FILTER_MA_DEFAULT_F16

#define GDFLIB_FILTER_MA_DEFAULT_F32

#define GDFLIB_FILTER_MA_DEFAULT_FLT

#define GDFLIB_FILTER_MA_T
```

```
#define GDFLIB_FILTER_MA_T

#define GDFLIB_FILTER_MA_T

#define GDFLIB_FilterFIR

#define GDFLIB_FilterFIRInit

#define GDFLIB_FilterIIR1

#define GDFLIB_FilterIIR1Init

#define GDFLIB_FilterIIR2

#define GDFLIB_FilterIIR2Init

#define GDFLIB_FilterMA

#define GDFLIB_FilterMAInit

#define GFLIB_ACOS_DEFAULT

#define GFLIB_ACOS_DEFAULT

#define GFLIB_ACOS_DEFAULT

#define GFLIB_ACOS_DEFAULT_F16

#define GFLIB_ACOS_DEFAULT_F32

#define GFLIB_ACOS_DEFAULT_FLT

#define GFLIB_ACOS_T

#define GFLIB_ACOS_T

#define GFLIB_ACOS_T

#define GFLIB_ASIN_DEFAULT

#define GFLIB_ASIN_DEFAULT

#define GFLIB_ASIN_DEFAULT

#define GFLIB_ASIN_DEFAULT_F16

#define GFLIB_ASIN_DEFAULT_F32

#define GFLIB_ASIN_DEFAULT_FLT

#define GFLIB_ASIN_FLT_INT1

#define GFLIB_ASIN_FLT_MIN
```

```
#define GFLIB_ASIN_T

#define GFLIB_ASIN_T

#define GFLIB_ASIN_T

#define GFLIB_ATANYXSHIFTED_T

#define GFLIB_ATANYXSHIFTED_T

#define GFLIB_ATANYXSHIFTED_T

#define GFLIB_ATAN_DEFAULT

#define GFLIB_ATAN_DEFAULT

#define GFLIB_ATAN_DEFAULT

#define GFLIB_ATAN_DEFAULT_F16

#define GFLIB_ATAN_DEFAULT_F32

#define GFLIB_ATAN_DEFAULT_FLT

#define GFLIB_ATAN_T

#define GFLIB_ATAN_T

#define GFLIB_ATAN_T

#define GFLIB_Acos

#define GFLIB_Asin

#define GFLIB_Atan

#define GFLIB_AtanYX

#define GFLIB_AtanYXShifted

#define GFLIB_CONTROLLER_PIAW_P_DEFAULT

#define GFLIB_CONTROLLER_PIAW_P_DEFAULT

#define GFLIB_CONTROLLER_PIAW_P_DEFAULT

#define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16

#define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32

#define GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT

#define GFLIB_CONTROLLER_PIAW_P_T
```

```
#define GFLIB_CONTROLLER_PIAW_P_T

#define GFLIB_CONTROLLER_PIAW_P_T

#define GFLIB_CONTROLLER_PIAW_R_DEFAULT

#define GFLIB_CONTROLLER_PIAW_R_DEFAULT

#define GFLIB_CONTROLLER_PIAW_R_DEFAULT

#define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16

#define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32

#define GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT

#define GFLIB_CONTROLLER_PIAW_R_T

#define GFLIB_CONTROLLER_PIAW_R_T

#define GFLIB_CONTROLLER_PIAW_R_T

#define GFLIB_CONTROLLER_PI_P_DEFAULT

#define GFLIB_CONTROLLER_PI_P_DEFAULT

#define GFLIB_CONTROLLER_PI_P_DEFAULT

#define GFLIB_CONTROLLER_PI_P_DEFAULT_F16

#define GFLIB_CONTROLLER_PI_P_DEFAULT_F32

#define GFLIB_CONTROLLER_PI_P_DEFAULT_FLT

#define GFLIB_CONTROLLER_PI_P_T

#define GFLIB_CONTROLLER_PI_P_T

#define GFLIB_CONTROLLER_PI_P_T

#define GFLIB_CONTROLLER_PI_R_DEFAULT

#define GFLIB_CONTROLLER_PI_R_DEFAULT

#define GFLIB_CONTROLLER_PI_R_DEFAULT

#define GFLIB_CONTROLLER_PI_R_DEFAULT_F16

#define GFLIB_CONTROLLER_PI_R_DEFAULT_F32

#define GFLIB_CONTROLLER_PI_R_DEFAULT_FLT

#define GFLIB_CONTROLLER_PI_R_T
```

```
#define GFLIB_CONTROLLER_PI_R_T

#define GFLIB_CONTROLLER_PI_R_T

#define GFLIB_COS_DEFAULT

#define GFLIB_COS_DEFAULT

#define GFLIB_COS_DEFAULT

#define GFLIB_COS_DEFAULT_F16

#define GFLIB_COS_DEFAULT_F32

#define GFLIB_COS_DEFAULT_FLT

#define GFLIB_COS_T

#define GFLIB_COS_T

#define GFLIB_COS_T

#define GFLIB_ControllerPIp

#define GFLIB_ControllerPIpAW

#define GFLIB_ControllerPIr

#define GFLIB_ControllerPIrAW

#define GFLIB_Cos

#define GFLIB_HYST_DEFAULT

#define GFLIB_HYST_DEFAULT

#define GFLIB_HYST_DEFAULT

#define GFLIB_HYST_DEFAULT_F16

#define GFLIB_HYST_DEFAULT_F32

#define GFLIB_HYST_DEFAULT_FLT

#define GFLIB_HYST_T

#define GFLIB_HYST_T

#define GFLIB_HYST_T

#define GFLIB_Hyst

#define GFLIB_INTEGRATOR_TR_DEFAULT
```

#define GFLIB_INTEGRATOR_TR_DEFAULT

#define GFLIB_INTEGRATOR_TR_DEFAULT

#define GFLIB_INTEGRATOR_TR_DEFAULT_F16

#define GFLIB_INTEGRATOR_TR_DEFAULT_F32

#define GFLIB_INTEGRATOR_TR_DEFAULT_FLT

#define GFLIB_INTEGRATOR_TR_T

#define GFLIB_INTEGRATOR_TR_T

#define GFLIB_INTEGRATOR_TR_T

#define GFLIB_IntegratorTR

#define GFLIB_LIMIT_DEFAULT

#define GFLIB_LIMIT_DEFAULT

#define GFLIB_LIMIT_DEFAULT

#define GFLIB_LIMIT_DEFAULT_F16

#define GFLIB_LIMIT_DEFAULT_F32

#define GFLIB_LIMIT_DEFAULT_FLT

#define GFLIB_LIMIT_T

#define GFLIB_LIMIT_T

#define GFLIB_LIMIT_T

#define GFLIB_LOG10_DEFAULT

#define GFLIB_LOG10_DEFAULT_FLT

#define GFLIB_LOG10_GET_FLOAT_WORD

#define GFLIB_LOG10_SET_FLOAT_WORD

#define GFLIB_LOG10_T

#define GFLIB_LOWERLIMIT_DEFAULT

#define GFLIB_LOWERLIMIT_DEFAULT

#define GFLIB_LOWERLIMIT_DEFAULT

#define GFLIB_LOWERLIMIT_DEFAULT_F16

```
#define GFLIB_LOWERLIMIT_DEFAULT_F32

#define GFLIB_LOWERLIMIT_DEFAULT_FLT

#define GFLIB_LOWERLIMIT_T

#define GFLIB_LOWERLIMIT_T

#define GFLIB_LOWERLIMIT_T

#define GFLIB_LUT1D_DEFAULT

#define GFLIB_LUT1D_DEFAULT

#define GFLIB_LUT1D_DEFAULT

#define GFLIB_LUT1D_DEFAULT_F16

#define GFLIB_LUT1D_DEFAULT_F32

#define GFLIB_LUT1D_DEFAULT_FLT

#define GFLIB_LUT1D_T

#define GFLIB_LUT1D_T

#define GFLIB_LUT1D_T

#define GFLIB_LUT2D_DEFAULT

#define GFLIB_LUT2D_DEFAULT

#define GFLIB_LUT2D_DEFAULT

#define GFLIB_LUT2D_DEFAULT_F16

#define GFLIB_LUT2D_DEFAULT_F32

#define GFLIB_LUT2D_DEFAULT_FLT

#define GFLIB_LUT2D_T

#define GFLIB_LUT2D_T

#define GFLIB_LUT2D_T

#define GFLIB_Limit

#define GFLIB_Log10

#define GFLIB_LowerLimit

#define GFLIB_Lut1D
```

```
#define GFLIB_Lut2D

#define GFLIB_RAMP_DEFAULT

#define GFLIB_RAMP_DEFAULT

#define GFLIB_RAMP_DEFAULT

#define GFLIB_RAMP_DEFAULT_F16

#define GFLIB_RAMP_DEFAULT_F32

#define GFLIB_RAMP_DEFAULT_FLT

#define GFLIB_RAMP_T

#define GFLIB_RAMP_T

#define GFLIB_RAMP_T

#define GFLIB_Ramp

#define GFLIB_SINCOS_DEFAULT

#define GFLIB_SINCOS_DEFAULT

#define GFLIB_SINCOS_DEFAULT

#define GFLIB_SINCOS_DEFAULT_F16

#define GFLIB_SINCOS_DEFAULT_F32

#define GFLIB_SINCOS_DEFAULT_FLT

#define GFLIB_SINCOS_FLT_MIN

#define GFLIB_SINCOS_T

#define GFLIB_SINCOS_T

#define GFLIB_SINCOS_T

#define GFLIB_SIN_DEFAULT

#define GFLIB_SIN_DEFAULT

#define GFLIB_SIN_DEFAULT

#define GFLIB_SIN_DEFAULT_F16

#define GFLIB_SIN_DEFAULT_F32

#define GFLIB_SIN_DEFAULT_FLT
```

```
#define GFLIB_SIN_FLT_MIN

#define GFLIB_SIN_T

#define GFLIB_SIN_T

#define GFLIB_SIN_T

#define GFLIB_Sign

#define GFLIB_Sin

#define GFLIB_SinCos

#define GFLIB_Sqrt

#define GFLIB_TAN_DEFAULT

#define GFLIB_TAN_DEFAULT

#define GFLIB_TAN_DEFAULT

#define GFLIB_TAN_DEFAULT_F16

#define GFLIB_TAN_DEFAULT_F32

#define GFLIB_TAN_DEFAULT_FLT

#define GFLIB_TAN_FLT_3P4

#define GFLIB_TAN_FLT_CORR1

#define GFLIB_TAN_FLT_CORR2

#define GFLIB_TAN_FLT_MIN

#define GFLIB_TAN_FLT_PI

#define GFLIB_TAN_FLT_PI2

#define GFLIB_TAN_FLT_PI4

#define GFLIB_TAN_T

#define GFLIB_TAN_T

#define GFLIB_TAN_T

#define GFLIB_Tan

#define GFLIB_UPPERLIMIT_DEFAULT

#define GFLIB_UPPERLIMIT_DEFAULT
```

```
#define GFLIB_UPPERLIMIT_DEFAULT

#define GFLIB_UPPERLIMIT_DEFAULT_F16

#define GFLIB_UPPERLIMIT_DEFAULT_F32

#define GFLIB_UPPERLIMIT_DEFAULT_FLT

#define GFLIB_UPPERLIMIT_T

#define GFLIB_UPPERLIMIT_T

#define GFLIB_UPPERLIMIT_T

#define GFLIB_UpperLimit

#define GFLIB_VECTORLIMIT_DEFAULT

#define GFLIB_VECTORLIMIT_DEFAULT

#define GFLIB_VECTORLIMIT_DEFAULT

#define GFLIB_VECTORLIMIT_DEFAULT_F16

#define GFLIB_VECTORLIMIT_DEFAULT_F32

#define GFLIB_VECTORLIMIT_DEFAULT_FLT

#define GFLIB_VECTORLIMIT_T

#define GFLIB_VECTORLIMIT_T

#define GFLIB_VECTORLIMIT_T

#define GFLIB_VLOG10_DEFAULT

#define GFLIB_VLOG10_DEFAULT_FLT

#define GFLIB_VLOG10_GET_FLOAT_WORD

#define GFLIB_VLOG10_SET_FLOAT_WORD

#define GFLIB_VLOG10_T

#define GFLIB_VLog10

#define GFLIB_VectorLimit

#define GMCLIB_Clark

#define GMCLIB_ClarkInv

#define GMCLIB_DECOUPLINGPMSM_DEFAULT
```

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT

#define GMCLIB_DECOUPLINGPMSM_DEFAULT

#define GMCLIB_DECOUPLINGPMSM_DEFAULT_F16

#define GMCLIB_DECOUPLINGPMSM_DEFAULT_F32

#define GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT

#define GMCLIB_DECOUPLINGPMSM_T

#define GMCLIB_DECOUPLINGPMSM_T

#define GMCLIB_DECOUPLINGPMSM_T

#define GMCLIB_DecouplingPMSM

#define GMCLIB_ELIMDCBUSRIP_DEFAULT

#define GMCLIB_ELIMDCBUSRIP_DEFAULT

#define GMCLIB_ELIMDCBUSRIP_DEFAULT

#define GMCLIB_ELIMDCBUSRIP_DEFAULT_F16

#define GMCLIB_ELIMDCBUSRIP_DEFAULT_F32

#define GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT

#define GMCLIB_ELIMDCBUSRIP_FLT_DNMAX

#define GMCLIB_ELIMDCBUSRIP_T

#define GMCLIB_ELIMDCBUSRIP_T

#define GMCLIB_ELIMDCBUSRIP_T

#define GMCLIB_ElimDcBusRip

#define GMCLIB_Park

#define GMCLIB_ParkInv

#define GMCLIB_SvmStd

#define INT16TOF16

#define INT16TOF32

#define INT16TOINT32

#define INT16_MAX
```

#define INT16_MIN

#define INT32TOF16

#define INT32TOF32

#define INT32TOINT16

#define INT32TOINT64

#define INT32_MAX

#define INT32_MIN

#define INT64TOF16

#define INT64TOF32

#define INT64TOINT32

#define MLIB_Abs

#define MLIB_AbsSat

#define MLIB_Add

#define MLIB_AddSat

#define MLIB_Convert

#define MLIB_ConvertPU

#define MLIB_Div

#define MLIB_DivSat

#define MLIB_Mac

#define MLIB_MacSat

#define MLIB_Mnac

#define MLIB_Msu

#define MLIB_Mul

#define MLIB_MulSat

#define MLIB_Neg

#define MLIB_NegSat

#define MLIB_Norm

```
#define MLIB_Round

#define MLIB_ShBi

#define MLIB_ShBiSat

#define MLIB_ShL

#define MLIB_ShLSat

#define MLIB_ShR

#define MLIB_Sub

#define MLIB_SubSat

#define MLIB_VMac

#define NULL

#define SFRACT_MAX

#define SFRACT_MIN

#define SWLIBS_2Syst

#define SWLIBS_2Syst

#define SWLIBS_2Syst

#define SWLIBS_3Syst

#define SWLIBS_3Syst

#define SWLIBS_3Syst

#define SWLIBS_DEFAULT_IMPLEMENTATION

#define SWLIBS_DEFAULT_IMPLEMENTATION_F16

#define SWLIBS_DEFAULT_IMPLEMENTATION_F32

#define SWLIBS_DEFAULT_IMPLEMENTATION_FLT

#define SWLIBS_ID

#define SWLIBS_MCID_SIZE

#define SWLIBS_MCIMPLEMENTATION_SIZE

#define SWLIBS_MCVERSION_SIZE

#define SWLIBS_STD_OFF
```

```
#define SWLIBS_STD_ON

#define SWLIBS_SUPPORTED_IMPLEMENTATION

#define SWLIBS_SUPPORT_F16

#define SWLIBS_SUPPORT_F32

#define SWLIBS_SUPPORT_FLT

#define SWLIBS_VERSION

#define SWLIBS_VERSION_DEFAULT

#define TRUE

#define UINT16_MAX

#define UINT32_MAX
```

Chapter 9
Macro References

This section describes in details the macro definitions available in Automotive Math and Motor Control Library Set for NXP MPC574xP devices.

9.1 Define AMCLIB_BemfObsrvDQInit

```
#include <AMCLIB_BemfObsrvDQ.h>
```

9.1.1 Macro Definition

```
#define AMCLIB_BemfObsrvDQInit macro_dispatcher(AMCLIB_BemfObsrvDQInit, __VA_ARGS__)
(__VA_ARGS__)
```

9.1.2 Description

This function initializes the state of the BEMF observer.

9.2 Define AMCLIB_BemfObsrvDQ

```
#include <AMCLIB_BemfObsrvDQ.h>
```

9.2.1 Macro Definition

```
#define AMCLIB_BemfObsrvDQ macro_dispatcher(AMCLIB_BemfObsrvDQ, __VA_ARGS__)(__VA_ARGS__)
```

9.2.2 Description

This function calculates the algorithm of back electro-motive force observer in rotating reference frame.

9.3 Define AMCLIB_BEMF_OBSRV_DQ_T

```
#include <AMCLIB_BemfObsrvDQ.h>
```

9.3.1 Macro Definition

```
#define AMCLIB_BEMF_OBSRV_DQ_T AMCLIB_BEMF_OBSRV_DQ_T_FLT
```

9.3.2 Description

9.4 Define AMCLIB_BEMF_OBSRV_DQ_T

```
#include <AMCLIB_BemfObsrvDQ.h>
```

9.4.1 Macro Definition

```
#define AMCLIB_BEMF_OBSRV_DQ_T AMCLIB_BEMF_OBSRV_DQ_T_FLT
```

9.4.2 Description

9.5 Define AMCLIB_BEMF_OBSRV_DQ_T

```
#include <AMCLIB_BemfObsrvDQ.h>
```

9.5.1 Macro Definition

```
#define AMCLIB_BEMF_OBSRV_DQ_T AMCLIB_BEMF_OBSRV_DQ_T_FLT
```

9.5.2 Description

9.6 Define AMCLIB_BEMF_OBSRV_DQ_DEFAULT_F32

```
#include <AMCLIB_BemfObsrvDQ.h>
```

9.6.1 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}
```

9.6.2 Description

Default value for AMCLIB_BEMF_OBSRV_DQ_T_F32.

9.7 Define AMCLIB_BEMF_OBSRV_DQ_DEFAULT_F16

```
#include <AMCLIB_BemfObsrvDQ.h>
```

9.7.1 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}
```

9.7.2 Description

Default value for AMCLIB_BEMF_OBSRV_DQ_T_F16.

9.8 Define AMCLIB_BEMF_OBSRV_DQ_DEFAULT_FLT

```
#include <AMCLIB_BemfObsrvDQ.h>
```

9.8.1 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}
```

9.8.2 Description

Default value for AMCLIB_BEMF_OBSRV_DQ_T_FLT.

9.9 Define AMCLIB_TrackObsrv

```
#include <AMCLIB_TrackObsrv.h>
```

9.9.1 Macro Definition

```
#define AMCLIB_TrackObsrv macro_dispatcher(AMCLIB_TrackObsrv, __VA_ARGS__)(__VA_ARGS__)
```

9.9.2 Description

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

9.10 Define AMCLIB_TrackObsrvInit

```
#include <AMCLIB_TrackObsrv.h>
```

9.10.1 Macro Definition

```
#define AMCLIB_TrackObsrvInit macro_dispatcher(AMCLIB_TrackObsrvInit, __VA_ARGS__)
(__VA_ARGS__)
```

9.10.2 Description

This function initialize tracking observer

9.11 Define AMCLIB_TRACK_OBSRV_T

```
#include <AMCLIB_TrackObsrv.h>
```

9.11.1 Macro Definition

```
#define AMCLIB_TRACK_OBSRV_T AMCLIB_TRACK_OBSRV_T_FLT
```

9.11.2 Description

Definition of AMCLIB_TRACK_OBSRV_T as alias for AMCLIB_TRACK_OBSRV_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of AMCLIB_TRACK_OBSRV_T as alias for AMCLIB_TRACK_OBSRV_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of AMCLIB_TRACK_OBSRV_T as alias for AMCLIB_TRACK_OBSRV_T_FLT datatype in case the single precision floating point implementation is selected.

9.12 Define AMCLIB_TRACK_OBSRV_T

```
#include <AMCLIB_TrackObsrv.h>
```

9.12.1 Macro Definition

```
#define AMCLIB_TRACK_OBSRV_T AMCLIB_TRACK_OBSRV_T_FLT
```

9.12.2 Description

Definition of AMCLIB_TRACK_OBSRV_T as alias for AMCLIB_TRACK_OBSRV_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of AMCLIB_TRACK_OBSRV_T as alias for AMCLIB_TRACK_OBSRV_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of AMCLIB_TRACK_OBSRV_T as alias for AMCLIB_TRACK_OBSRV_T_FLT datatype in case the single precision floating point implementation is selected.

9.13 Define AMCLIB_TRACK_OBSRV_T

```
#include <AMCLIB_TrackObsrv.h>
```

9.13.1 Macro Definition

```
#define AMCLIB_TRACK_OBSRV_T AMCLIB_TRACK_OBSRV_T_FLT
```

9.13.2 Description

Definition of AMCLIB_TRACK_OBSRV_T as alias for AMCLIB_TRACK_OBSRV_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of AMCLIB_TRACK_OBSRV_T as alias for AMCLIB_TRACK_OBSRV_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of AMCLIB_TRACK_OBSRV_T as alias for AMCLIB_TRACK_OBSRV_T_FLT datatype in case the single precision floating point implementation is selected.

9.14 Define AMCLIB_TRACK_OBSRV_DEFAULT

```
#include <AMCLIB_TrackObsrv.h>
```

9.14.1 Macro Definition

```
#define AMCLIB_TRACK_OBSRV_DEFAULT AMCLIB_TRACK_OBSRV_DEFAULT_FLT
```

9.14.2 Description

Definition of AMCLIB_TRACK_OBSRV_DEFAULT as alias for AMCLIB_TRACK_OBSRV_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of AMCLIB_TRACK_OBSRV_DEFAULT as alias for AMCLIB_TRACK_OBSRV_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.15 Define AMCLIB_TRACK_OBSRV_DEFAULT

```
#include <AMCLIB_TrackObsrv.h>
```

9.15.1 Macro Definition

```
#define AMCLIB_TRACK_OBSRV_DEFAULT AMCLIB_TRACK_OBSRV_DEFAULT_FLT
```

9.15.2 Description

Definition of AMCLIB_TRACK_OBSRV_DEFAULT as alias for AMCLIB_TRACK_OBSRV_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of AMCLIB_TRACK_OBSRV_DEFAULT as alias for AMCLIB_TRACK_OBSRV_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.16 Define AMCLIB_TRACK_OBSRV_DEFAULT

```
#include <AMCLIB_TrackObsrv.h>
```

9.16.1 Macro Definition

```
#define AMCLIB_TRACK_OBSRV_DEFAULT AMCLIB_TRACK_OBSRV_DEFAULT_FLT
```

9.16.2 Description

Definition of AMCLIB_TRACK_OBSRV_DEFAULT as alias for AMCLIB_TRACK_OBSRV_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of AMCLIB_TRACK_OBSRV_DEFAULT as alias for AMCLIB_TRACK_OBSRV_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.17 Define AMCLIB_TRACK_OBSRV_DEFAULT_F32

```
#include <AMCLIB_TrackObsrv.h>
```

9.17.1 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}
```

9.17.2 Description

Default value for AMCLIB_TRACK_OBSRV_T_F32.

9.18 Define AMCLIB_TRACK_OBSRV_DEFAULT_F16

```
#include <AMCLIB_TrackObsrv.h>
```

9.18.1 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}
```

9.18.2 Description

Default value for AMCLIB_TRACK_OBSRV_T_F16.

9.19 Define AMCLIB_TRACK_OBSRV_DEFAULT_FLT

```
#include <AMCLIB_TrackObsrv.h>
```

9.19.1 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}
```

9.19.2 Description

Default value for AMCLIB_TRACK_OBSRV_T_FLT.

9.20 Define GDFLIB_FilterFIRInit

```
#include <GDFLIB_FilterFIR.h>
```

9.20.1 Macro Definition

```
#define GDFLIB_FilterFIRInit macro_dispatcher(GDFLIB_FilterFIRInit, __VA_ARGS__)(__VA_ARGS__)
```

9.20.2 Description

This function initializes the FIR filter buffers.

9.21 Define GDFLIB_FilterFIR

```
#include <GDFLIB_FilterFIR.h>
```

9.21.1 Macro Definition

```
#define GDFLIB_FilterFIR macro_dispatcher(GDFLIB_FilterFIR, __VA_ARGS__)(__VA_ARGS__)
```

9.21.2 Description

The function performs a single iteration of an FIR filter.

9.22 Define GDFLIB_FILTERFIR_PARAM_T

```
#include <GDFLIB_FilterFIR.h>
```

9.22.1 Macro Definition

```
#define GDFLIB_FILTERFIR_PARAM_T GDFLIB_FILTERFIR_PARAM_T_FLT
```

9.22.2 Description

Definition of alias for GDFLIB_FILTERFIR_PARAM_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_PARAM_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_PARAM_T_FLT datatype in case the single precision floating point implementation is selected.

9.23 Define GDFLIB_FILTERFIR_PARAM_T

```
#include <GDFLIB_FilterFIR.h>
```

9.23.1 Macro Definition

```
#define GDFLIB_FILTERFIR_PARAM_T GDFLIB_FILTERFIR_PARAM_T_FLT
```

9.23.2 Description

Definition of alias for GDFLIB_FILTERFIR_PARAM_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_PARAM_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_PARAM_T_FLT datatype in case the single precision floating point implementation is selected.

9.24 Define GDFLIB_FILTERFIR_PARAM_T

```
#include <GDFLIB_FilterFIR.h>
```

9.24.1 Macro Definition

```
#define GDFLIB_FILTERFIR_PARAM_T GDFLIB_FILTERFIR_PARAM_T_FLT
```

9.24.2 Description

Definition of alias for GDFLIB_FILTERFIR_PARAM_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_PARAM_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_PARAM_T_FLT datatype in case the single precision floating point implementation is selected.

9.25 Define GDFLIB_FILTERFIR_STATE_T

```
#include <GDFLIB_FilterFIR.h>
```

9.25.1 Macro Definition

```
#define GDFLIB_FILTERFIR_STATE_T GDFLIB_FILTERFIR_STATE_T_FLT
```

9.25.2 Description

Definition of alias for GDFLIB_FILTERFIR_STATE_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_STATE_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_STATE_T_FLT datatype in case the single precision floating point implementation is selected.

9.26 Define GDFLIB_FILTERFIR_STATE_T

```
#include <GDFLIB_FilterFIR.h>
```

9.26.1 Macro Definition

```
#define GDFLIB_FILTERFIR_STATE_T GDFLIB_FILTERFIR_STATE_T_FLT
```

9.26.2 Description

Definition of alias for GDFLIB_FILTERFIR_STATE_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_STATE_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_STATE_T_FLT datatype in case the single precision floating point implementation is selected.

9.27 Define GDFLIB_FILTERFIR_STATE_T

```
#include <GDFLIB_FilterFIR.h>
```

9.27.1 Macro Definition

```
#define GDFLIB_FILTERFIR_STATE_T GDFLIB_FILTERFIR_STATE_T_FLT
```

9.27.2 Description

Definition of alias for GDFLIB_FILTERFIR_STATE_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_STATE_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_STATE_T_FLT datatype in case the single precision floating point implementation is selected.

9.28 Define GDFLIB_FilterIIR1Init

```
#include <GDFLIB_FilterIIR1.h>
```

9.28.1 Macro Definition

```
#define GDFLIB_FilterIIR1Init macro_dispatcher(GDFLIB_FilterIIR1Init, __VA_ARGS__)
(__VA_ARGS__)
```

9.28.2 Description

This function initializes the first order IIR filter buffers.

9.29 Define GDFLIB_FilterIIR1

```
#include <GDFLIB_FilterIIR1.h>
```

9.29.1 Macro Definition

```
#define GDFLIB_FilterIIR1 macro_dispatcher(GDFLIB_FilterIIR1, __VA_ARGS__)(__VA_ARGS__)
```

9.29.2 Description

This function implements the first order IIR filter.

9.30 Define GDFLIB_FILTER_IIR1_T

```
#include <GDFLIB_FilterIIR1.h>
```

9.30.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_T GDFLIB_FILTER_IIR1_T_FLT
```

9.30.2 Description

Definition of GDFLIB_FILTER_IIR1_T as alias for GDFLIB_FILTER_IIR1_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR1_T as alias for GDFLIB_FILTER_IIR1_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR1_T as alias for GDFLIB_FILTER_IIR1_T_FLT datatype in case the single precision floating point implementation is selected.

9.31 Define GDFLIB_FILTER_IIR1_T

```
#include <GDFLIB_FilterIIR1.h>
```

9.31.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_T GDFLIB_FILTER_IIR1_T_FLT
```

9.31.2 Description

Definition of GDFLIB_FILTER_IIR1_T as alias for GDFLIB_FILTER_IIR1_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR1_T as alias for GDFLIB_FILTER_IIR1_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR1_T as alias for GDFLIB_FILTER_IIR1_T_FLT datatype in case the single precision floating point implementation is selected.

9.32 Define GDFLIB_FILTER_IIR1_T

```
#include <GDFLIB_FilterIIR1.h>
```

9.32.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_T GDFLIB_FILTER_IIR1_T_FLT
```

9.32.2 Description

Definition of GDFLIB_FILTER_IIR1_T as alias for GDFLIB_FILTER_IIR1_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR1_T as alias for GDFLIB_FILTER_IIR1_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR1_T as alias for GDFLIB_FILTER_IIR1_T_FLT datatype in case the single precision floating point implementation is selected.

9.33 Define GDFLIB_FILTER_IIR1_DEFAULT

```
#include <GDFLIB_FilterIIR1.h>
```

9.33.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_DEFAULT GDFLIB_FILTER_IIR1_DEFAULT_FLT
```

9.33.2 Description

Definition of GDFLIB_FILTER_IIR1_DEFAULT as alias for GDFLIB_FILTER_IIR1_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GDFLIB_FILTER_IIR1_DEFAULT as alias for GDFLIB_FILTER_IIR1_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.34 Define GDFLIB_FILTER_IIR1_DEFAULT

```
#include <GDFLIB_FilterIIR1.h>
```

9.34.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_DEFAULT GDFLIB_FILTER_IIR1_DEFAULT_FLT
```

9.34.2 Description

Definition of GDFLIB_FILTER_IIR1_DEFAULT as alias for GDFLIB_FILTER_IIR1_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GDFLIB_FILTER_IIR1_DEFAULT as alias for GDFLIB_FILTER_IIR1_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.35 Define GDFLIB_FILTER_IIR1_DEFAULT

```
#include <GDFLIB_FilterIIR1.h>
```

9.35.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_DEFAULT GDFLIB_FILTER_IIR1_DEFAULT_FLT
```

9.35.2 Description

Definition of GDFLIB_FILTER_IIR1_DEFAULT as alias for GDFLIB_FILTER_IIR1_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GDFLIB_FILTER_IIR1_DEFAULT as alias for GDFLIB_FILTER_IIR1_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.36 Define GDFLIB_FILTER_IIR1_DEFAULT_F32

```
#include <GDFLIB_FilterIIR1.h>
```

9.36.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_DEFAULT_F32 {{(tFrac32)0,(tFrac32)0,(tFrac32)0},{(tFrac32)0},
{(tFrac32)0}}
```

9.36.2 Description

Default value for GDFLIB_FILTER_IIR1_T_F32.

9.37 Define GDFLIB_FILTER_IIR1_DEFAULT_F16

```
#include <GDFLIB_FilterIIR1.h>
```

9.37.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_DEFAULT_F16 {{(tFrac16)0,(tFrac16)0,(tFrac16)0},{(tFrac16)0},
{(tFrac16)0}}
```

9.37.2 Description

Default value for GDFLIB_FILTER_IIR1_T_F16.

9.38 Define GDFLIB_FILTER_IIR1_DEFAULT_FLT

```
#include <GDFLIB_FilterIIR1.h>
```

9.38.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_DEFAULT_FLT {{(tFloat)0,(tFloat)0,(tFloat)0},{(tFloat)0},
{(tFloat)0}}
```

9.38.2 Description

Default value for GDFLIB_FILTER_IIR1_T_FLT.

9.39 Define GDFLIB_FilterIIR2Init

```
#include <GDFLIB_FilterIIR2.h>
```

9.39.1 Macro Definition

```
#define GDFLIB_FilterIIR2Init macro_dispatcher(GDFLIB_FilterIIR2Init, __VA_ARGS__)
(__VA_ARGS__)
```

9.39.2 Description

This function initializes the second order IIR filter buffers.

9.40 Define GDFLIB_FilterIIR2

```
#include <GDFLIB_FilterIIR2.h>
```

9.40.1 Macro Definition

```
#define GDFLIB_FilterIIR2 macro_dispatcher(GDFLIB_FilterIIR2, __VA_ARGS__)(__VA_ARGS__)
```

9.40.2 Description

This function implements the second order IIR filter.

9.41 Define GDFLIB_FILTER_IIR2_T

```
#include <GDFLIB_FilterIIR2.h>
```

9.41.1 Macro Definition

```
#define GDFLIB_FILTER_IIR2_T GDFLIB_FILTER_IIR2_T_FLT
```

9.41.2 Description

Definition of GDFLIB_FILTER_IIR2_T as alias for GDFLIB_FILTER_IIR2_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_T as alias for GDFLIB_FILTER_IIR2_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_T as alias for GDFLIB_FILTER_IIR2_T_FLT datatype in case the single precision floating point implementation is selected.

9.42 Define GDFLIB_FILTER_IIR2_T

```
#include <GDFLIB_FilterIIR2.h>
```

9.42.1 Macro Definition

```
#define GDFLIB_FILTER_IIR2_T GDFLIB_FILTER_IIR2_T_FLT
```

9.42.2 Description

Definition of GDFLIB_FILTER_IIR2_T as alias for GDFLIB_FILTER_IIR2_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_T as alias for GDFLIB_FILTER_IIR2_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_T as alias for GDFLIB_FILTER_IIR2_T_FLT datatype in case the single precision floating point implementation is selected.

9.43 Define GDFLIB_FILTER_IIR2_T

```
#include <GDFLIB_FilterIIR2.h>
```

9.43.1 Macro Definition

```
#define GDFLIB_FILTER_IIR2_T GDFLIB_FILTER_IIR2_T_FLT
```

9.43.2 Description

Definition of GDFLIB_FILTER_IIR2_T as alias for GDFLIB_FILTER_IIR2_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_T as alias for GDFLIB_FILTER_IIR2_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_T as alias for GDFLIB_FILTER_IIR2_T_FLT datatype in case the single precision floating point implementation is selected.

9.44 Define GDFLIB_FILTER_IIR2_DEFAULT

```
#include <GDFLIB_FilterIIR2.h>
```

9.44.1 Macro Definition

```
#define GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_FLT
```

9.44.2 Description

Definition of GDFLIB_FILTER_IIR2_DEFAULT as alias for GDFLIB_FILTER_IIR2_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.45 Define GDFLIB_FILTER_IIR2_DEFAULT

```
#include <GDFLIB_FilterIIR2.h>
```

9.45.1 Macro Definition

```
#define GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_FLT
```

9.45.2 Description

Definition of GDFLIB_FILTER_IIR2_DEFAULT as alias for GDFLIB_FILTER_IIR2_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.46 Define GDFLIB_FILTER_IIR2_DEFAULT

```
#include <GDFLIB_FilterIIR2.h>
```

9.46.1 Macro Definition

```
#define GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_FLT
```

9.46.2 Description

Definition of GDFLIB_FILTER_IIR2_DEFAULT as alias for GDFLIB_FILTER_IIR2_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.47 Define GDFLIB_FILTER_IIR2_DEFAULT_F32

```
#include <GDFLIB_FilterIIR2.h>
```

9.47.1 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}}
```

9.47.2 Description

Default value for GDFLIB_FILTER_IIR2_T_F32.

9.48 Define GDFLIB_FILTER_IIR2_DEFAULT_F16

```
#include <GDFLIB_FilterIIR2.h>
```

9.48.1 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}}
```

9.48.2 Description

Default value for GDFLIB_FILTER_IIR2_T_F16.

9.49 Define GDFLIB_FILTER_IIR2_DEFAULT_FLT

```
#include <GDFLIB_FilterIIR2.h>
```

9.49.1 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}}
```

9.49.2 Description

Default value for GDFLIB_FILTER_IIR2_T_FLT.

9.50 Define GDFLIB_FilterMAInit

```
#include <GDFLIB_FilterMA.h>
```

9.50.1 Macro Definition

```
#define GDFLIB_FilterMAInit macro_dispatcher(GDFLIB_FilterMAInit, __VA_ARGS__)(__VA_ARGS__)
```

9.50.2 Description

This function clears the internal filter accumulator.

9.51 Define GDFLIB_FilterMA

```
#include <GDFLIB_FilterMA.h>
```

9.51.1 Macro Definition

```
#define GDFLIB_FilterMA macro_dispatcher(GDFLIB_FilterMA, __VA_ARGS__)(__VA_ARGS__)
```

9.51.2 Description

This function implements a moving average recursive filter.

9.52 Define GDFLIB_FILTER_MA_T

```
#include <GDFLIB_FilterMA.h>
```

9.52.1 Macro Definition

```
#define GDFLIB_FILTER_MA_T GDFLIB_FILTER_MA_T_FLT
```

9.52.2 Description

Definition of GDFLIB_FILTER_MA_T as alias for GDFLIB_FILTER_MA_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_MA_T as alias for GDFLIB_FILTER_MA_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_MA_T as alias for GDFLIB_FILTER_MA_T_FLT datatype in case the single precision floating point implementation is selected.

9.53 Define GDFLIB_FILTER_MA_T

```
#include <GDFLIB_FilterMA.h>
```

9.53.1 Macro Definition

```
#define GDFLIB_FILTER_MA_T GDFLIB_FILTER_MA_T_FLT
```

9.53.2 Description

Definition of GDFLIB_FILTER_MA_T as alias for GDFLIB_FILTER_MA_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_MA_T as alias for GDFLIB_FILTER_MA_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_MA_T as alias for GDFLIB_FILTER_MA_T_FLT datatype in case the single precision floating point implementation is selected.

9.54 Define GDFLIB_FILTER_MA_T

```
#include <GDFLIB_FilterMA.h>
```

9.54.1 Macro Definition

```
#define GDFLIB_FILTER_MA_T GDFLIB_FILTER_MA_T_FLT
```

9.54.2 Description

Definition of GDFLIB_FILTER_MA_T as alias for GDFLIB_FILTER_MA_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_MA_T as alias for GDFLIB_FILTER_MA_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_MA_T as alias for GDFLIB_FILTER_MA_T_FLT datatype in case the single precision floating point implementation is selected.

9.55 Define GDFLIB_FILTER_MA_DEFAULT

```
#include <GDFLIB_FilterMA.h>
```

9.55.1 Macro Definition

```
#define GDFLIB_FILTER_MA_DEFAULT GDFLIB_FILTER_MA_DEFAULT_FLT
```

9.55.2 Description

Definition of GDFLIB_FILTER_MA_DEFAULT as alias for GDFLIB_FILTER_MA_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GDFLIB_FILTER_MA_DEFAULT as alias for GDFLIB_FILTER_MA_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.56 Define GDFLIB_FILTER_MA_DEFAULT

```
#include <GDFLIB_FilterMA.h>
```

9.56.1 Macro Definition

```
#define GDFLIB_FILTER_MA_DEFAULT GDFLIB_FILTER_MA_DEFAULT_FLT
```

9.56.2 Description

Definition of GDFLIB_FILTER_MA_DEFAULT as alias for GDFLIB_FILTER_MA_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GDFLIB_FILTER_MA_DEFAULT as alias for GDFLIB_FILTER_MA_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.57 Define GDFLIB_FILTER_MA_DEFAULT

```
#include <GDFLIB_FilterMA.h>
```

9.57.1 Macro Definition

```
#define GDFLIB_FILTER_MA_DEFAULT GDFLIB_FILTER_MA_DEFAULT_FLT
```

9.57.2 Description

Definition of GDFLIB_FILTER_MA_DEFAULT as alias for GDFLIB_FILTER_MA_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GDFLIB_FILTER_MA_DEFAULT as alias for GDFLIB_FILTER_MA_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.58 Define GDFLIB_FILTER_MA_DEFAULT_F32

```
#include <GDFLIB_FilterMA.h>
```

9.58.1 Macro Definition

```
#define GDFLIB_FILTER_MA_DEFAULT_F32 {0,0}
```

9.58.2 Description

Default value for GDFLIB_FILTER_MA_T_F32.

9.59 Define GDFLIB_FILTER_MA_DEFAULT_F16

```
#include <GDFLIB_FilterMA.h>
```

9.59.1 Macro Definition

```
#define GDFLIB_FILTER_MA_DEFAULT_F16 {0,0}
```

9.59.2 Description

Default value for GDFLIB_FILTER_MA_T_F16.

9.60 Define GDFLIB_FILTER_MA_DEFAULT_FLT

```
#include <GDFLIB_FilterMA.h>
```

9.60.1 Macro Definition

```
#define GDFLIB_FILTER_MA_DEFAULT_FLT {0,0}
```

9.60.2 Description

Default value for GDFLIB_FILTER_MA_T_FLT.

9.61 Define GFLIB_Acos

```
#include <GFLIB_Acos.h>
```

9.61.1 Macro Definition

```
#define GFLIB_Acos macro_dispatcher(GFLIB_Acos, __VA_ARGS__)(__VA_ARGS__)
```

9.61.2 Description

This function implements polynomial approximation of arccosine function.

9.62 Define GFLIB_ACOS_T

```
#include <GFLIB_Acos.h>
```

9.62.1 Macro Definition

```
#define GFLIB_ACOS_T GFLIB_ACOS_T_FLT
```

9.62.2 Description

Definition of GFLIB_ACOS_T as alias for GFLIB_ACOS_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ACOS_T as alias for GFLIB_ACOS_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ACOS_T as alias for GFLIB_ACOS_T_FLT datatype in case the Single precision floating point implementation is selected.

9.63 Define GFLIB_ACOS_T

```
#include <GFLIB_Acos.h>
```

9.63.1 Macro Definition

```
#define GFLIB_ACOS_T GFLIB_ACOS_T_FLT
```

9.63.2 Description

Definition of GFLIB_ACOS_T as alias for GFLIB_ACOS_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ACOS_T as alias for GFLIB_ACOS_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ACOS_T as alias for GFLIB_ACOS_T_FLT datatype in case the Single precision floating point implementation is selected.

9.64 Define GFLIB_ACOS_T

```
#include <GFLIB_Acos.h>
```

9.64.1 Macro Definition

```
#define GFLIB_ACOS_T GFLIB_ACOS_T_FLT
```

9.64.2 Description

Definition of GFLIB_ACOS_T as alias for GFLIB_ACOS_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ACOS_T as alias for GFLIB_ACOS_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ACOS_T as alias for GFLIB_ACOS_T_FLT datatype in case the Single precision floating point implementation is selected.

9.65 Define GFLIB_ACOS_DEFAULT

```
#include <GFLIB_Acos.h>
```

9.65.1 Macro Definition

```
#define GFLIB_ACOS_DEFAULT GFLIB_ACOS_DEFAULT_FLT
```

9.65.2 Description

Definition of GFLIB_ACOS_DEFAULT as alias for GFLIB_ACOS_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ACOS_DEFAULT as alias for GFLIB_ACOS_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ACOS_DEFAULT as alias for GFLIB_ACOS_DEFAULT_FLT default value in case the Single precision floating point implementation is selected.

9.66 Define GFLIB_ACOS_DEFAULT

```
#include <GFLIB_Acos.h>
```

9.66.1 Macro Definition

```
#define GFLIB_ACOS_DEFAULT GFLIB_ACOS_DEFAULT_FLT
```

9.66.2 Description

Definition of GFLIB_ACOS_DEFAULT as alias for GFLIB_ACOS_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ACOS_DEFAULT as alias for GFLIB_ACOS_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ACOS_DEFAULT as alias for GFLIB_ACOS_DEFAULT_FLT default value in case the Single precision floating point implementation is selected.

9.67 Define GFLIB_ACOS_DEFAULT

```
#include <GFLIB_Acos.h>
```

9.67.1 Macro Definition

```
#define GFLIB_ACOS_DEFAULT GFLIB_ACOS_DEFAULT_FLT
```

9.67.2 Description

Definition of GFLIB_ACOS_DEFAULT as alias for GFLIB_ACOS_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ACOS_DEFAULT as alias for GFLIB_ACOS_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ACOS_DEFAULT as alias for GFLIB_ACOS_DEFAULT_FLT default value in case the Single precision floating point implementation is selected.

9.68 Define GFLIB_ACOS_DEFAULT_F32

```
#include <GFLIB_Acos.h>
```

9.68.1 Macro Definition

```
#define GFLIB_ACOS_DEFAULT_F32 &f32gflibAcosCoef
```

9.68.2 Description

Default approximation coefficients for GFLIB_Acos_F32 function.

9.69 Define GFLIB_ACOS_DEFAULT_F16

```
#include <GFLIB_Acos.h>
```

9.69.1 Macro Definition

```
#define GFLIB_ACOS_DEFAULT_F16 &f16gflibAcosCoef
```

9.69.2 Description

Default approximation coefficients for GFLIB_Acos_F16 function.

9.70 Define GFLIB_ACOS_DEFAULT_FLT

```
#include <GFLIB_Acos.h>
```

9.70.1 Macro Definition

```
#define GFLIB_ACOS_DEFAULT_FLT &fltgflibAcosCoef
```

9.70.2 Description

Default approximation coefficients for GFLIB_Acos_FLT function.

9.71 Define GFLIB_ASIN_FLT_MIN

#include <GFLIB_Asin.c>

9.71.1 Macro Definition

```
#define GFLIB_ASIN_FLT_MIN ((tFloat)0.005)
```

9.71.2 Description

Floating-point min. input value for computation.

9.72 Define GFLIB_ASIN_FLT_INT1

#include <GFLIB_Asin.c>

9.72.1 Macro Definition

```
#define GFLIB_ASIN_FLT_INT1 ((tFloat)0.41)
```

9.72.2 Description

Floating-point min. input value for computation in interval 1.

9.73 Define GFLIB_Asin

#include <GFLIB_Asin.h>

9.73.1 Macro Definition

```
#define GFLIB_Asin macro_dispatcher(GFLIB_Asin, __VA_ARGS__)(__VA_ARGS__)
```

9.73.2 Description

This function implements polynomial approximation of arcsine function.

9.74 Define GFLIB_ASIN_T

```
#include <GFLIB_Asin.h>
```

9.74.1 Macro Definition

```
#define GFLIB_ASIN_T GFLIB_ASIN_T_FLT
```

9.74.2 Description

Definition of GFLIB_ASIN_T as alias for GFLIB_ASIN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ASIN_T as alias for GFLIB_ASIN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ASIN_T as alias for GFLIB_ASIN_T_FLT datatype in case the Single precision floating point implementation is selected.

9.75 Define GFLIB_ASIN_T

```
#include <GFLIB_Asin.h>
```

9.75.1 Macro Definition

```
#define GFLIB_ASIN_T GFLIB_ASIN_T_FLT
```

9.75.2 Description

Definition of GFLIB_ASIN_T as alias for GFLIB_ASIN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ASIN_T as alias for GFLIB_ASIN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ASIN_T as alias for GFLIB_ASIN_T_FLT datatype in case the Single precision floating point implementation is selected.

9.76 Define GFLIB_ASIN_T

```
#include <GFLIB_Asin.h>
```

9.76.1 Macro Definition

```
#define GFLIB_ASIN_T GFLIB_ASIN_T_FLT
```

9.76.2 Description

Definition of GFLIB_ASIN_T as alias for GFLIB_ASIN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ASIN_T as alias for GFLIB_ASIN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ASIN_T as alias for GFLIB_ASIN_T_FLT datatype in case the Single precision floating point implementation is selected.

9.77 Define GFLIB_ASIN_DEFAULT

```
#include <GFLIB_Asin.h>
```

9.77.1 Macro Definition

```
#define GFLIB_ASIN_DEFAULT GFLIB_ASIN_DEFAULT_FLT
```

9.77.2 Description

Definition of GFLIB_ASIN_DEFAULT as alias for GFLIB_ASIN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ASIN_DEFAULT as alias for GFLIB_ASIN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ASIN_DEFAULT as alias for GFLIB_ASIN_DEFAULT_FLT default value in case the Single precision floating point implementation is selected.

9.78 Define GFLIB_ASIN_DEFAULT

```
#include <GFLIB_Asin.h>
```

9.78.1 Macro Definition

```
#define GFLIB_ASIN_DEFAULT GFLIB_ASIN_DEFAULT_FLT
```

9.78.2 Description

Definition of GFLIB_ASIN_DEFAULT as alias for GFLIB_ASIN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ASIN_DEFAULT as alias for GFLIB_ASIN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ASIN_DEFAULT as alias for GFLIB_ASIN_DEFAULT_FLT default value in case the Single precision floating point implementation is selected.

9.79 Define GFLIB_ASIN_DEFAULT

```
#include <GFLIB_Asin.h>
```

9.79.1 Macro Definition

```
#define GFLIB_ASIN_DEFAULT GFLIB_ASIN_DEFAULT_FLT
```

9.79.2 Description

Definition of GFLIB_ASIN_DEFAULT as alias for GFLIB_ASIN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ASIN_DEFAULT as alias for GFLIB_ASIN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ASIN_DEFAULT as alias for GFLIB_ASIN_DEFAULT_FLT default value in case the Single precision floating point implementation is selected.

9.80 Define GFLIB_ASIN_DEFAULT_F32

```
#include <GFLIB_Asin.h>
```

9.80.1 Macro Definition

```
#define GFLIB_ASIN_DEFAULT_F32 &f32gflibAsinCoef
```

9.80.2 Description

Default approximation coefficients for GFLIB_Asin_F32 function.

9.81 Define GFLIB_ASIN_DEFAULT_F16

```
#include <GFLIB_Asin.h>
```

9.81.1 Macro Definition

```
#define GFLIB_ASIN_DEFAULT_F16 &f16gflibAsinCoef
```

9.81.2 Description

Default approximation coefficients for GFLIB_Asin_F16 function.

9.82 Define GFLIB_ASIN_DEFAULT_FLT

```
#include <GFLIB_Asin.h>
```

9.82.1 Macro Definition

```
#define GFLIB_ASIN_DEFAULT_FLT &fltgflibAsinCoef
```

9.82.2 Description

Default approximation coefficients for GFLIB_Asin_FLT function.

9.83 Define GFLIB_Atan

```
#include <GFLIB_Atan.h>
```

9.83.1 Macro Definition

```
#define GFLIB_Atan macro_dispatcher(GFLIB_Atan, __VA_ARGS__)(__VA_ARGS__)
```

9.83.2 Description

This function implements polynomial approximation of arctangent function.

9.84 Define GFLIB_ATAN_T

```
#include <GFLIB_Atan.h>
```

9.84.1 Macro Definition

```
#define GFLIB_ATAN_T GFLIB_ATAN_T_FLT
```

9.84.2 Description

Definition of GFLIB_ATAN_T as alias for GFLIB_ATAN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ATAN_T as alias for GFLIB_ATAN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ATAN_T as alias for GFLIB_ATAN_T_FLT datatype in case the Single precision floating point implementation is selected.

9.85 Define GFLIB_ATAN_T

```
#include <GFLIB_Atan.h>
```

9.85.1 Macro Definition

```
#define GFLIB_ATAN_T GFLIB_ATAN_T_FLT
```

9.85.2 Description

Definition of GFLIB_ATAN_T as alias for GFLIB_ATAN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ATAN_T as alias for GFLIB_ATAN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ATAN_T as alias for GFLIB_ATAN_T_FLT datatype in case the Single precision floating point implementation is selected.

9.86 Define GFLIB_ATAN_T

```
#include <GFLIB_Atan.h>
```

9.86.1 Macro Definition

```
#define GFLIB_ATAN_T GFLIB_ATAN_T_FLT
```

9.86.2 Description

Definition of GFLIB_ATAN_T as alias for GFLIB_ATAN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ATAN_T as alias for GFLIB_ATAN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ATAN_T as alias for GFLIB_ATAN_T_FLT datatype in case the Single precision floating point implementation is selected.

9.87 Define GFLIB_ATAN_DEFAULT

```
#include <GFLIB_Atan.h>
```

9.87.1 Macro Definition

```
#define GFLIB_ATAN_DEFAULT GFLIB_ATAN_DEFAULT_FLT
```

9.87.2 Description

Definition of GFLIB_ATAN_DEFAULT as alias for GFLIB_ATAN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ATAN_DEFAULT as alias for GFLIB_ATAN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ATAN_DEFAULT as alias for GFLIB_ATAN_DEFAULT_FLT default value in case the Single precision floating point implementation is selected.

9.88 Define GFLIB_ATAN_DEFAULT

```
#include <GFLIB_Atan.h>
```

9.88.1 Macro Definition

```
#define GFLIB_ATAN_DEFAULT GFLIB_ATAN_DEFAULT_FLT
```

9.88.2 Description

Definition of GFLIB_ATAN_DEFAULT as alias for GFLIB_ATAN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ATAN_DEFAULT as alias for GFLIB_ATAN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ATAN_DEFAULT as alias for GFLIB_ATAN_DEFAULT_FLT default value in case the Single precision floating point implementation is selected.

9.89 Define GFLIB_ATAN_DEFAULT

```
#include <GFLIB_Atan.h>
```

9.89.1 Macro Definition

```
#define GFLIB_ATAN_DEFAULT GFLIB_ATAN_DEFAULT_FLT
```

9.89.2 Description

Definition of GFLIB_ATAN_DEFAULT as alias for GFLIB_ATAN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ATAN_DEFAULT as alias for GFLIB_ATAN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ATAN_DEFAULT as alias for GFLIB_ATAN_DEFAULT_FLT default value in case the Single precision floating point implementation is selected.

9.90 Define GFLIB_ATAN_DEFAULT_F32

```
#include <GFLIB_Atan.h>
```

9.90.1 Macro Definition

```
#define GFLIB_ATAN_DEFAULT_F32 &f32gflibAtanCoef
```

9.90.2 Description

Default approximation coefficients for GFLIB_Atan_F32 function.

9.91 Define GFLIB_ATAN_DEFAULT_F16

```
#include <GFLIB_Atan.h>
```

9.91.1 Macro Definition

```
#define GFLIB_ATAN_DEFAULT_F16 &f16gflibAtanCoef
```

9.91.2 Description

Default approximation coefficients for GFLIB_Atan_F16 function.

9.92 Define GFLIB_ATAN_DEFAULT_FLT

```
#include <GFLIB_Atan.h>
```

9.92.1 Macro Definition

```
#define GFLIB_ATAN_DEFAULT_FLT &fltgflibAtanCoef
```

9.92.2 Description

Default approximation coefficients for GFLIB_Atan_FLT function.

9.93 Define GFLIB_AtanYX

```
#include <GFLIB_AtanYX.h>
```

9.93.1 Macro Definition

```
#define GFLIB_AtanYX macro_dispatcher(GFLIB_AtanYX, __VA_ARGS__)(__VA_ARGS__)
```

9.93.2 Description

This function calculate the angle between the positive x-axis and the direction of a vector given by the (x, y) coordinates.

9.94 Define GFLIB_AtanYXShifted

```
#include <GFLIB_AtanYXShifted.h>
```

9.94.1 Macro Definition

```
#define GFLIB_AtanYXShifted macro_dispatcher(GFLIB_AtanYXShifted, __VA_ARGS__)(__VA_ARGS__)
```

9.94.2 Description

This function calculates the angle of two sine waves shifted in phase to each other.

9.95 Define GFLIB_ATANYXSHIFTED_T

```
#include <GFLIB_AtanYXShifted.h>
```

9.95.1 Macro Definition

```
#define GFLIB_ATANYXSHIFTED_T GFLIB_ATANYXSHIFTED_T_FLT
```

9.95.2 Description

Definition of GFLIB_ATANYXSHIFTED_T as alias for GFLIB_ATANYXSHIFTED_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ATANYXSHIFTED_T as alias for GFLIB_ATANYXSHIFTED_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ATANYXSHIFTED_T as alias for GFLIB_ATANYXSHIFTED_T_FLT datatype in case the single precision floating point implementation is selected.

9.96 Define GFLIB_ATANYXSHIFTED_T

```
#include <GFLIB_AtanYXShifted.h>
```

9.96.1 Macro Definition

```
#define GFLIB_ATANYXSHIFTED_T GFLIB_ATANYXSHIFTED_T_FLT
```

9.96.2 Description

Definition of GFLIB_ATANYXSHIFTED_T as alias for GFLIB_ATANYXSHIFTED_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ATANYXSHIFTED_T as alias for GFLIB_ATANYXSHIFTED_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ATANYXSHIFTED_T as alias for GFLIB_ATANYXSHIFTED_T_FLT datatype in case the single precision floating point implementation is selected.

9.97 Define GFLIB_ATANYXSHIFTED_T

```
#include <GFLIB_AtanYXShifted.h>
```

9.97.1 Macro Definition

```
#define GFLIB_ATANYXSHIFTED_T GFLIB_ATANYXSHIFTED_T_FLT
```

9.97.2 Description

Definition of GFLIB_ATANYXSHIFTED_T as alias for GFLIB_ATANYXSHIFTED_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ATANYXSHIFTED_T as alias for GFLIB_ATANYXSHIFTED_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ATANYXSHIFTED_T as alias for GFLIB_ATANYXSHIFTED_T_FLT datatype in case the single precision floating point implementation is selected.

9.98 Define GFLIB_ControllerPIp

```
#include <GFLIB_ControllerPIp.h>
```

9.98.1 Macro Definition

```
#define GFLIB_ControllerPIp macro_dispatcher(GFLIB_ControllerPIp, __VA_ARGS__)(__VA_ARGS__)
```

9.98.2 Description

This function calculates a parallel form of the Proportional- Integral controller, without integral anti-windup.

9.99 Define GFLIB_CONTROLLER_PI_P_T

```
#include <GFLIB_ControllerPIp.h>
```

9.99.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_T GFLIB_CONTROLLER_PI_P_T_FLT
```

9.99.2 Description

Definition of GFLIB_CONTROLLER_PI_P_T as alias for GFLIB_CONTROLLER_PI_P_T_F32 datatype in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PI_P_T as alias for GFLIB_CONTROLLER_PI_P_T_FLT datatype in case the single precision floating point implementation is selected.

9.100 Define GFLIB_CONTROLLER_PI_P_T

```
#include <GFLIB_ControllerPIp.h>
```

9.100.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_T GFLIB_CONTROLLER_PI_P_T_FLT
```

9.100.2 Description

Definition of GFLIB_CONTROLLER_PI_P_T as alias for GFLIB_CONTROLLER_PI_P_T_F32 datatype in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PI_P_T as alias for GFLIB_CONTROLLER_PI_P_T_FLT datatype in case the single precision floating point implementation is selected.

9.101 Define GFLIB_CONTROLLER_PI_P_T

```
#include <GFLIB_ControllerPIp.h>
```

9.101.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_T GFLIB_CONTROLLER_PI_P_T_FLT
```

9.101.2 Description

Definition of GFLIB_CONTROLLER_PI_P_T as alias for GFLIB_CONTROLLER_PI_P_T_F32 datatype in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PI_P_T as alias for GFLIB_CONTROLLER_PI_P_T_FLT datatype in case the single precision floating point implementation is selected.

9.102 Define GFLIB_CONTROLLER_PI_P_DEFAULT

```
#include <GFLIB_ControllerPIp.h>
```

9.102.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_DEFAULT GFLIB_CONTROLLER_PI_P_DEFAULT_FLT
```

9.102.2 Description

Definition of GFLIB_CONTROLLER_PI_P_DEFAULT as alias for GFLIB_CONTROLLER_PI_P_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PI_P_DEFAULT as alias for GFLIB_CONTROLLER_PI_P_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.103 Define GFLIB_CONTROLLER_PI_P_DEFAULT

```
#include <GFLIB_ControllerPIp.h>
```

9.103.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_DEFAULT GFLIB_CONTROLLER_PI_P_DEFAULT_FLT
```

9.103.2 Description

Definition of GFLIB_CONTROLLER_PI_P_DEFAULT as alias for GFLIB_CONTROLLER_PI_P_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PI_P_DEFAULT as alias for GFLIB_CONTROLLER_PI_P_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.104 Define GFLIB_CONTROLLER_PI_P_DEFAULT

```
#include <GFLIB_ControllerPIp.h>
```

9.104.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_DEFAULT GFLIB_CONTROLLER_PI_P_DEFAULT_FLT
```

9.104.2 Description

Definition of GFLIB_CONTROLLER_PI_P_DEFAULT as alias for GFLIB_CONTROLLER_PI_P_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PI_P_DEFAULT as alias for GFLIB_CONTROLLER_PI_P_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.105 Define GFLIB_CONTROLLER_PI_P_DEFAULT_F32

```
#include <GFLIB_ControllerPIp.h>
```

9.105.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_DEFAULT_F32 {(tFrac32)0,(tFrac32)0,(tS16)0,(tS16)0,(tFrac32)0,
(tFrac32)0}
```

9.105.2 Description

Default value for GFLIB_CONTROLLER_PI_P_T_F32.

9.106 Define GFLIB_CONTROLLER_PI_P_DEFAULT_F16

```
#include <GFLIB_ControllerPIp.h>
```

9.106.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_DEFAULT_F16 {(tFrac16)0,(tFrac16)0,(tS16)0,(tS16)0,(tFrac32)0,
(tFrac16)0}
```

9.106.2 Description

Default value for GFLIB_CONTROLLER_PI_P_T_F16.

9.107 Define GFLIB_CONTROLLER_PI_P_DEFAULT_FLT

```
#include <GFLIB_ControllerPIp.h>
```

9.107.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_DEFAULT_FLT {(tFloat)0,(tFloat)0,(tFloat)0,(tFloat)0}
```

9.107.2 Description

Default value for GFLIB_CONTROLLER_PI_P_T_FLT.

9.108 Define GFLIB_ControllerPIpAW

```
#include <GFLIB_ControllerPIpAW.h>
```

9.108.1 Macro Definition

```
#define GFLIB_ControllerPIpAW macro_dispatcher(GFLIB_ControllerPIpAW, __VA_ARGS__)
(__VA_ARGS__)
```

9.108.2 Description

This function calculates a standard recurrent form of the Proportional- Integral controller, with integral anti-windup.

9.109 Define GFLIB_CONTROLLER_PIAW_P_T

```
#include <GFLIB_ControllerPIpAW.h>
```

9.109.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_T GFLIB_CONTROLLER_PIAW_P_T_FLT
```

9.109.2 Description

Definition of GFLIB_CONTROLLER_PIAW_P_T as alias for GFLIB_CONTROLLER_PIAW_P_T_F32 datatype in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PIAW_P_T as alias for GFLIB_CONTROLLER_PIAW_P_T_FLT datatype in case the single precision floating point implementation is selected.

9.110 Define GFLIB_CONTROLLER_PIAW_P_T

```
#include <GFLIB_ControllerPIpAW.h>
```

9.110.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_T GFLIB_CONTROLLER_PIAW_P_T_FLT
```

9.110.2 Description

Definition of GFLIB_CONTROLLER_PIAW_P_T as alias for GFLIB_CONTROLLER_PIAW_P_T_F32 datatype in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PIAW_P_T as alias for GFLIB_CONTROLLER_PIAW_P_T_FLT datatype in case the single precision floating point implementation is selected.

9.111 Define GFLIB_CONTROLLER_PIAW_P_T

```
#include <GFLIB_ControllerPIpAW.h>
```

9.111.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_T GFLIB_CONTROLLER_PIAW_P_T_FLT
```

9.111.2 Description

Definition of GFLIB_CONTROLLER_PIAW_P_T as alias for GFLIB_CONTROLLER_PIAW_P_T_F32 datatype in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PIAW_P_T as alias for GFLIB_CONTROLLER_PIAW_P_T_FLT datatype in case the single precision floating point implementation is selected.

9.112 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT

```
#include <GFLIB_ControllerPIpAW.h>
```

9.112.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_DEFAULT GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT
```

9.112.2 Description

Definition of GFLIB_CONTROLLER_PIAW_P_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PIAW_P_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.113 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT

```
#include <GFLIB_ControllerPIpAW.h>
```

9.113.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_DEFAULT GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT
```

9.113.2 Description

Definition of GFLIB_CONTROLLER_PIAW_P_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PIAW_P_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.114 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT

```
#include <GFLIB_ControllerPIpAW.h>
```

9.114.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_DEFAULT GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT
```

9.114.2 Description

Definition of GFLIB_CONTROLLER_PIAW_P_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PIAW_P_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.115 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32

```
#include <GFLIB_ControllerPIpAW.h>
```

9.115.1 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}
```

9.115.2 Description

Default value for GFLIB_CONTROLLER_PIAW_P_T_F32.

9.116 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16

```
#include <GFLIB_ControllerPIpAW.h>
```

9.116.1 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}
```

9.116.2 Description

Default value for GFLIB_CONTROLLER_PIAW_P_T_F16.

9.117 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT

```
#include <GFLIB_ControllerPIpAW.h>
```

9.117.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT {(tFloat)0,(tFloat)0,FLOAT_MIN,FLOAT_MAX,
(tFloat)0,(tFloat)0,(tU16)0}
```

9.117.2 Description

Default value for GFLIB_CONTROLLER_PIAW_P_T_FLT.

9.118 Define GFLIB_ControllerPIr

```
#include <GFLIB_ControllerPIr.h>
```

9.118.1 Macro Definition

```
#define GFLIB_ControllerPIr macro_dispatcher(GFLIB_ControllerPIr, __VA_ARGS__)(__VA_ARGS__)
```

9.118.2 Description

This function calculates a standard recurrent form of the Proportional- Integral controller, without integral anti-windup.

9.119 Define GFLIB_CONTROLLER_PI_R_T

```
#include <GFLIB_ControllerPIr.h>
```

9.119.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_T GFLIB_CONTROLLER_PI_R_T_FLT
```

9.119.2 Description

Definition of GFLIB_CONTROLLER_PI_R_T as alias for GFLIB_CONTROLLER_PI_R_T_F32 datatype in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PI_R_T as alias for GFLIB_CONTROLLER_PI_R_T_FLT datatype in case the single precision floating point implementation is selected.

9.120 Define GFLIB_CONTROLLER_PI_R_T

```
#include <GFLIB_ControllerPIr.h>
```

9.120.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_T GFLIB_CONTROLLER_PI_R_T_FLT
```

9.120.2 Description

Definition of GFLIB_CONTROLLER_PI_R_T as alias for GFLIB_CONTROLLER_PI_R_T_F32 datatype in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PI_R_T as alias for GFLIB_CONTROLLER_PI_R_T_FLT datatype in case the single precision floating point implementation is selected.

9.121 Define GFLIB_CONTROLLER_PI_R_T

```
#include <GFLIB_ControllerPIr.h>
```

9.121.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_T GFLIB_CONTROLLER_PI_R_T_FLT
```

9.121.2 Description

Definition of GFLIB_CONTROLLER_PI_R_T as alias for GFLIB_CONTROLLER_PI_R_T_F32 datatype in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PI_R_T as alias for GFLIB_CONTROLLER_PI_R_T_FLT datatype in case the single precision floating point implementation is selected.

9.122 Define GFLIB_CONTROLLER_PI_R_DEFAULT

```
#include <GFLIB_ControllerPIr.h>
```

9.122.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_DEFAULT GFLIB_CONTROLLER_PI_R_DEFAULT_FLT
```

9.122.2 Description

Definition of GFLIB_CONTROLLER_PI_R_DEFAULT as alias for GFLIB_CONTROLLER_PI_R_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PI_R_DEFAULT as alias for GFLIB_CONTROLLER_PI_R_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.123 Define GFLIB_CONTROLLER_PI_R_DEFAULT

```
#include <GFLIB_ControllerPIr.h>
```

9.123.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_DEFAULT GFLIB_CONTROLLER_PI_R_DEFAULT_FLT
```

9.123.2 Description

Definition of GFLIB_CONTROLLER_PI_R_DEFAULT as alias for GFLIB_CONTROLLER_PI_R_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PI_R_DEFAULT as alias for GFLIB_CONTROLLER_PI_R_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.124 Define GFLIB_CONTROLLER_PI_R_DEFAULT

```
#include <GFLIB_ControllerPIr.h>
```

9.124.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_DEFAULT GFLIB_CONTROLLER_PI_R_DEFAULT_FLT
```

9.124.2 Description

Definition of GFLIB_CONTROLLER_PI_R_DEFAULT as alias for GFLIB_CONTROLLER_PI_R_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PI_R_DEFAULT as alias for GFLIB_CONTROLLER_PI_R_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.125 Define GFLIB_CONTROLLER_PI_R_DEFAULT_F32

```
#include <GFLIB_ControllerPIr.h>
```

9.125.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_DEFAULT_F32 {(tFrac32)0,(tFrac32)0,(tFrac32)0,(tFrac32)0,
(tU16)0}
```

9.125.2 Description

Default value for GFLIB_CONTROLLER_PI_R_T_F32.

9.126 Define GFLIB_CONTROLLER_PI_R_DEFAULT_F16

```
#include <GFLIB_ControllerPIr.h>
```

9.126.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_DEFAULT_F16 {(tFrac16)0,(tFrac16)0,(tFrac32)0,(tFrac16)0,
(tU16)0}
```

9.126.2 Description

Default value for GFLIB_CONTROLLER_PI_R_T_F16.

9.127 Define GFLIB_CONTROLLER_PI_R_DEFAULT_FLT

```
#include <GFLIB_ControllerPIr.h>
```

9.127.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_DEFAULT_FLT {(tFloat)0,(tFloat)0,(tFloat)0,(tFloat)0}
```

9.127.2 Description

Default value for GFLIB_CONTROLLER_PI_R_T_FLT.

9.128 Define GFLIB_ControllerPIrAW

```
#include <GFLIB_ControllerPIrAW.h>
```

9.128.1 Macro Definition

```
#define GFLIB_ControllerPIrAW macro_dispatcher(GFLIB_ControllerPIrAW, __VA_ARGS__)
(__VA_ARGS__)
```

9.128.2 Description

This function calculates a standard recurrent form of the Proportional-Integral controller, with integral anti-windup.

9.129 Define GFLIB_CONTROLLER_PIAW_R_T

```
#include <GFLIB_ControllerPIrAW.h>
```

9.129.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_T GFLIB_CONTROLLER_PIAW_R_T_FLT
```

9.129.2 Description

Definition of GFLIB_CONTROLLER_PIAW_R_T as alias for GFLIB_CONTROLLER_PIAW_R_T_F32 datatype in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PIAW_R_T as alias for GFLIB_CONTROLLER_PIAW_R_T_FLT datatype in case the single precision floating point implementation is selected.

9.130 Define GFLIB_CONTROLLER_PIAW_R_T

```
#include <GFLIB_ControllerPIrAW.h>
```

9.130.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_T GFLIB_CONTROLLER_PIAW_R_T_FLT
```

9.130.2 Description

Definition of GFLIB_CONTROLLER_PIAW_R_T as alias for GFLIB_CONTROLLER_PIAW_R_T_F32 datatype in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PIAW_R_T as alias for GFLIB_CONTROLLER_PIAW_R_T_FLT datatype in case the single precision floating point implementation is selected.

9.131 Define GFLIB_CONTROLLER_PIAW_R_T

```
#include <GFLIB_ControllerPIrAW.h>
```

9.131.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_T GFLIB_CONTROLLER_PIAW_R_T_FLT
```

9.131.2 Description

Definition of GFLIB_CONTROLLER_PIAW_R_T as alias for GFLIB_CONTROLLER_PIAW_R_T_F32 datatype in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PIAW_R_T as alias for GFLIB_CONTROLLER_PIAW_R_T_FLT datatype in case the single precision floating point implementation is selected.

9.132 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT

```
#include <GFLIB_ControllerPIrAW.h>
```

9.132.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_DEFAULT GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT
```

9.132.2 Description

Definition of GFLIB_CONTROLLER_PIAW_R_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PIAW_R_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.133 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT

```
#include <GFLIB_ControllerPIrAW.h>
```

9.133.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_DEFAULT GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT
```

9.133.2 Description

Definition of GFLIB_CONTROLLER_PIAW_R_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PIAW_R_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.134 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT

```
#include <GFLIB_ControllerPIrAW.h>
```

9.134.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_DEFAULT GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT
```

9.134.2 Description

Definition of GFLIB_CONTROLLER_PIAW_R_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_CONTROLLER_PIAW_R_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.135 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32

```
#include <GFLIB_ControllerPIrAW.h>
```

9.135.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32 {(tFrac32)0,(tFrac32)0,(tFrac32)0,
(tFrac32)0,INT32_MIN,INT32_MAX,(tU16)0}
```

9.135.2 Description

Default value for GFLIB_CONTROLLER_PIAW_R_T_F32.

9.136 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16

```
#include <GFLIB_ControllerPIrAW.h>
```

9.136.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16 {(tFrac16)0,(tFrac16)0,(tFrac32)0,
(tFrac16)0,INT16_MIN,INT16_MAX,(tU16)0}
```

9.136.2 Description

Default value for GFLIB_CONTROLLER_PIAW_R_T_F16.

9.137 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT

```
#include <GFLIB_ControllerPIrAW.h>
```

9.137.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT {(tFloat)0,(tFloat)0,(tFloat)0,
(tFloat)0,FLOAT_MIN,FLOAT_MAX }
```

9.137.2 Description

Default value for GFLIB_CONTROLLER_PIAW_R_T_FLT.

9.138 Define GFLIB_Cos

```
#include <GFLIB_Cos.h>
```

9.138.1 Macro Definition

```
#define GFLIB_Cos macro_dispatcher(GFLIB_Cos, __VA_ARGS__)(__VA_ARGS__)
```

9.138.2 Description

This function implements polynomial approximation of cosine function.

9.139 Define GFLIB_COS_T

```
#include <GFLIB_Cos.h>
```

9.139.1 Macro Definition

```
#define GFLIB_COS_T GFLIB_COS_T_FLT
```

9.139.2 Description

Definition of GFLIB_COS_T as alias for GFLIB_COS_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_COS_T as alias for GFLIB_COS_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_COS_T as alias for GFLIB_COS_T_FLT datatype in case the single precision floating point implementation is selected.

9.140 Define GFLIB_COS_T

```
#include <GFLIB_Cos.h>
```

9.140.1 Macro Definition

```
#define GFLIB_COS_T GFLIB_COS_T_FLT
```

9.140.2 Description

Definition of GFLIB_COS_T as alias for GFLIB_COS_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_COS_T as alias for GFLIB_COS_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_COS_T as alias for GFLIB_COS_T_FLT datatype in case the single precision floating point implementation is selected.

9.141 Define GFLIB_COS_T

```
#include <GFLIB_Cos.h>
```

9.141.1 Macro Definition

```
#define GFLIB_COS_T GFLIB_COS_T_FLT
```

9.141.2 Description

Definition of GFLIB_COS_T as alias for GFLIB_COS_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_COS_T as alias for GFLIB_COS_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_COS_T as alias for GFLIB_COS_T_FLT datatype in case the single precision floating point implementation is selected.

9.142 Define GFLIB_COS_DEFAULT

```
#include <GFLIB_Cos.h>
```

9.142.1 Macro Definition

```
#define GFLIB_COS_DEFAULT GFLIB_COS_DEFAULT_FLT
```

9.142.2 Description

Definition of GFLIB_COS_DEFAULT as alias for GFLIB_COS_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_COS_DEFAULT as alias for GFLIB_COS_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_COS_DEFAULT as alias for GFLIB_COS_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.143 Define GFLIB_COS_DEFAULT

```
#include <GFLIB_Cos.h>
```

9.143.1 Macro Definition

```
#define GFLIB_COS_DEFAULT GFLIB_COS_DEFAULT_FLT
```

9.143.2 Description

Definition of GFLIB_COS_DEFAULT as alias for GFLIB_COS_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_COS_DEFAULT as alias for GFLIB_COS_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_COS_DEFAULT as alias for GFLIB_COS_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.144 Define GFLIB_COS_DEFAULT

```
#include <GFLIB_Cos.h>
```

9.144.1 Macro Definition

```
#define GFLIB_COS_DEFAULT GFLIB_COS_DEFAULT_FLT
```

9.144.2 Description

Definition of GFLIB_COS_DEFAULT as alias for GFLIB_COS_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_COS_DEFAULT as alias for GFLIB_COS_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_COS_DEFAULT as alias for GFLIB_COS_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.145 Define GFLIB_COS_DEFAULT_F32

```
#include <GFLIB_Cos.h>
```

9.145.1 Macro Definition

```
#define GFLIB_COS_DEFAULT_F32 &f32gflibCosCoef
```

9.145.2 Description

Default approximation coefficients for GFLIB_Cos_F32 function.

9.146 Define GFLIB_COS_DEFAULT_F16

```
#include <GFLIB_Cos.h>
```

9.146.1 Macro Definition

```
#define GFLIB_COS_DEFAULT_F16 &f16gflibCosCoef
```

9.146.2 Description

Default approximation coefficients for GFLIB_Cos_F32 function.

9.147 Define GFLIB_COS_DEFAULT_FLT

```
#include <GFLIB_Cos.h>
```

9.147.1 Macro Definition

```
#define GFLIB_COS_DEFAULT_FLT &fltgflibCosCoef
```

9.147.2 Description

Default approximation coefficients for GFLIB_Cos_FLT function.

9.148 Define GFLIB_Hyst

```
#include <GFLIB_Hyst.h>
```

9.148.1 Macro Definition

```
#define GFLIB_Hyst macro_dispatcher(GFLIB_Hyst, __VA_ARGS__)(__VA_ARGS__)
```

9.148.2 Description

The function implements the hysteresis functionality.

9.149 Define GFLIB_HYST_T

```
#include <GFLIB_Hyst.h>
```

9.149.1 Macro Definition

```
#define GFLIB_HYST_T GFLIB_HYST_T_FLT
```

9.149.2 Description

Definition of GFLIB_HYST_T as alias for GFLIB_HYST_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_HYST_T as alias for GFLIB_HYST_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_HYST_T as alias for GFLIB_HYST_T_FLT datatype in case the single precision floating point implementation is selected.

9.150 Define GFLIB_HYST_T

```
#include <GFLIB_Hyst.h>
```

9.150.1 Macro Definition

```
#define GFLIB_HYST_T GFLIB_HYST_T_FLT
```

9.150.2 Description

Definition of GFLIB_HYST_T as alias for GFLIB_HYST_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_HYST_T as alias for GFLIB_HYST_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_HYST_T as alias for GFLIB_HYST_T_FLT datatype in case the single precision floating point implementation is selected.

9.151 Define GFLIB_HYST_T

```
#include <GFLIB_Hyst.h>
```

9.151.1 Macro Definition

```
#define GFLIB_HYST_T GFLIB_HYST_T_FLT
```

9.151.2 Description

Definition of GFLIB_HYST_T as alias for GFLIB_HYST_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_HYST_T as alias for GFLIB_HYST_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_HYST_T as alias for GFLIB_HYST_T_FLT datatype in case the single precision floating point implementation is selected.

9.152 Define GFLIB_HYST_DEFAULT

```
#include <GFLIB_Hyst.h>
```

9.152.1 Macro Definition

```
#define GFLIB_HYST_DEFAULT GFLIB_HYST_DEFAULT_FLT
```

9.152.2 Description

Definition of GFLIB_HYST_DEFAULT as alias for GFLIB_HYST_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_HYST_DEFAULT as alias for GFLIB_HYST_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_HYST_DEFAULT as alias for GFLIB_HYST_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.153 Define GFLIB_HYST_DEFAULT

```
#include <GFLIB_Hyst.h>
```

9.153.1 Macro Definition

```
#define GFLIB_HYST_DEFAULT GFLIB_HYST_DEFAULT_FLT
```

9.153.2 Description

Definition of GFLIB_HYST_DEFAULT as alias for GFLIB_HYST_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_HYST_DEFAULT as alias for GFLIB_HYST_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_HYST_DEFAULT as alias for GFLIB_HYST_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.154 Define GFLIB_HYST_DEFAULT

```
#include <GFLIB_Hyst.h>
```

9.154.1 Macro Definition

```
#define GFLIB_HYST_DEFAULT GFLIB_HYST_DEFAULT_FLT
```

9.154.2 Description

Definition of GFLIB_HYST_DEFAULT as alias for GFLIB_HYST_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_HYST_DEFAULT as alias for GFLIB_HYST_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_HYST_DEFAULT as alias for GFLIB_HYST_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.155 Define GFLIB_HYST_DEFAULT_F32

```
#include <GFLIB_Hyst.h>
```

9.155.1 Macro Definition

```
#define GFLIB_HYST_DEFAULT_F32 {(tFrac32)0,(tFrac32)0,(tFrac32)0,(tFrac32)0,(tFrac32)0}
```

9.155.2 Description

Default value for GFLIB_HYST_T_F32.

9.156 Define GFLIB_HYST_DEFAULT_F16

```
#include <GFLIB_Hyst.h>
```

9.156.1 Macro Definition

```
#define GFLIB_HYST_DEFAULT_F16 {(tFrac16)0,(tFrac16)0,(tFrac16)0,(tFrac16)0,(tFrac16)0}
```

9.156.2 Description

Default value for GFLIB_HYST_T_F16.

9.157 Define GFLIB_HYST_DEFAULT_FLT

```
#include <GFLIB_Hyst.h>
```

9.157.1 Macro Definition

```
#define GFLIB_HYST_DEFAULT_FLT {(tFloat)0,(tFloat)0,(tFloat)0,(tFloat)0,(tFloat)0}
```

9.157.2 Description

Default value for GFLIB_HYST_T_FLT.

9.158 Define GFLIB_IntegratorTR

```
#include <GFLIB_IntegratorTR.h>
```

9.158.1 Macro Definition

```
#define GFLIB_IntegratorTR macro_dispatcher(GFLIB_IntegratorTR, __VA_ARGS__)(__VA_ARGS__)
```

9.158.2 Description

The function calculates a discrete implementation of the integrator (sum), discretized using a trapezoidal (Bilinear) transformation.

9.159 Define GFLIB_INTEGRATOR_TR_T

```
#include <GFLIB_IntegratorTR.h>
```

9.159.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_T GFLIB_INTEGRATOR_TR_T_FLT
```

9.159.2 Description

Definition of GFLIB_INTEGRATOR_TR_T as alias for GFLIB_INTEGRATOR_TR_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_INTEGRATOR_TR_T as alias for GFLIB_INTEGRATOR_TR_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_INTEGRATOR_TR_T as alias for GFLIB_INTEGRATOR_TR_T_FLT datatype in case the single precision floating point implementation is selected.

9.160 Define GFLIB_INTEGRATOR_TR_T

```
#include <GFLIB_IntegratorTR.h>
```

9.160.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_T GFLIB_INTEGRATOR_TR_T_FLT
```

9.160.2 Description

Definition of GFLIB_INTEGRATOR_TR_T as alias for GFLIB_INTEGRATOR_TR_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_INTEGRATOR_TR_T as alias for GFLIB_INTEGRATOR_TR_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_INTEGRATOR_TR_T as alias for GFLIB_INTEGRATOR_TR_T_FLT datatype in case the single precision floating point implementation is selected.

9.161 Define GFLIB_INTEGRATOR_TR_T

```
#include <GFLIB_IntegratorTR.h>
```

9.161.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_T GFLIB_INTEGRATOR_TR_T_FLT
```

9.161.2 Description

Definition of GFLIB_INTEGRATOR_TR_T as alias for GFLIB_INTEGRATOR_TR_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_INTEGRATOR_TR_T as alias for GFLIB_INTEGRATOR_TR_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_INTEGRATOR_TR_T as alias for GFLIB_INTEGRATOR_TR_T_FLT datatype in case the single precision floating point implementation is selected.

9.162 Define GFLIB_INTEGRATOR_TR_DEFAULT

```
#include <GFLIB_IntegratorTR.h>
```

9.162.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_DEFAULT GFLIB_INTEGRATOR_TR_DEFAULT_FLT
```

9.162.2 Description

Definition of GFLIB_INTEGRATOR_TR_DEFAULT as alias for GFLIB_INTEGRATOR_TR_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_INTEGRATOR_TR_DEFAULT as alias for GFLIB_INTEGRATOR_TR_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.163 Define GFLIB_INTEGRATOR_TR_DEFAULT

```
#include <GFLIB_IntegratorTR.h>
```

9.163.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_DEFAULT GFLIB_INTEGRATOR_TR_DEFAULT_FLT
```

9.163.2 Description

Definition of GFLIB_INTEGRATOR_TR_DEFAULT as alias for GFLIB_INTEGRATOR_TR_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_INTEGRATOR_TR_DEFAULT as alias for GFLIB_INTEGRATOR_TR_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.164 Define GFLIB_INTEGRATOR_TR_DEFAULT

```
#include <GFLIB_IntegratorTR.h>
```

9.164.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_DEFAULT GFLIB_INTEGRATOR_TR_DEFAULT_FLT
```

9.164.2 Description

Definition of GFLIB_INTEGRATOR_TR_DEFAULT as alias for GFLIB_INTEGRATOR_TR_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

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.

Definition of GFLIB_INTEGRATOR_TR_DEFAULT as alias for GFLIB_INTEGRATOR_TR_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.165 Define GFLIB_INTEGRATOR_TR_DEFAULT_F32

```
#include <GFLIB_IntegratorTR.h>
```

9.165.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_DEFAULT_F32 {(tFrac32)0,(tFrac32)0,(tFrac32)0,(tU16)0}
```

9.165.2 Description

Default value for GFLIB_INTEGRATOR_TR_T_F32.

9.166 Define GFLIB_INTEGRATOR_TR_DEFAULT_F16

```
#include <GFLIB_IntegratorTR.h>
```

9.166.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_DEFAULT_F16 {(tFrac32)0,(tFrac16)0,(tFrac16)0,(tU16)0}
```

9.166.2 Description

Default value for GFLIB_INTEGRATOR_TR_T_F16.

9.167 Define GFLIB_INTEGRATOR_TR_DEFAULT_FLT

```
#include <GFLIB_IntegratorTR.h>
```

9.167.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_DEFAULT_FLT {(tFloat)0,(tFloat)0,(tFloat)0}
```

9.167.2 Description

Default value for GFLIB_INTEGRATOR_TR_T_FLT.

9.168 Define GFLIB_Limit

```
#include <GFLIB_Limit.h>
```

9.168.1 Macro Definition

```
#define GFLIB_Limit macro_dispatcher(GFLIB_Limit, __VA_ARGS__)(__VA_ARGS__)
```

9.168.2 Description

This function tests whether the input value is within the upper and lower limits.

9.169 Define GFLIB_LIMIT_T

```
#include <GFLIB_Limit.h>
```

9.169.1 Macro Definition

```
#define GFLIB_LIMIT_T GFLIB_LIMIT_T_FLT
```

9.169.2 Description

Definition of GFLIB_LIMIT_T as alias for GFLIB_LIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_T as alias for GFLIB_LIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_T as alias for GFLIB_LIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

9.170 Define GFLIB_LIMIT_T

```
#include <GFLIB_Limit.h>
```

9.170.1 Macro Definition

```
#define GFLIB_LIMIT_T GFLIB_LIMIT_T_FLT
```

9.170.2 Description

Definition of GFLIB_LIMIT_T as alias for GFLIB_LIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_T as alias for GFLIB_LIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_T as alias for GFLIB_LIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

9.171 Define GFLIB_LIMIT_T

```
#include <GFLIB_Limit.h>
```

9.171.1 Macro Definition

```
#define GFLIB_LIMIT_T GFLIB_LIMIT_T_FLT
```

9.171.2 Description

Definition of GFLIB_LIMIT_T as alias for GFLIB_LIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_T as alias for GFLIB_LIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_T as alias for GFLIB_LIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

9.172 Define GFLIB_LIMIT_DEFAULT

```
#include <GFLIB_Limit.h>
```

9.172.1 Macro Definition

```
#define GFLIB_LIMIT_DEFAULT GFLIB_LIMIT_DEFAULT_FLT
```

9.172.2 Description

Definition of GFLIB_LIMIT_DEFAULT as alias for GFLIB_LIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_DEFAULT as alias for GFLIB_LIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_DEFAULT as alias for GFLIB_LIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.173 Define GFLIB_LIMIT_DEFAULT

```
#include <GFLIB_Limit.h>
```

9.173.1 Macro Definition

```
#define GFLIB_LIMIT_DEFAULT GFLIB_LIMIT_DEFAULT_FLT
```

9.173.2 Description

Definition of GFLIB_LIMIT_DEFAULT as alias for GFLIB_LIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_DEFAULT as alias for GFLIB_LIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_DEFAULT as alias for GFLIB_LIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.174 Define GFLIB_LIMIT_DEFAULT

```
#include <GFLIB_Limit.h>
```

9.174.1 Macro Definition

```
#define GFLIB_LIMIT_DEFAULT GFLIB_LIMIT_DEFAULT_FLT
```

9.174.2 Description

Definition of GFLIB_LIMIT_DEFAULT as alias for GFLIB_LIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_DEFAULT as alias for GFLIB_LIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_DEFAULT as alias for GFLIB_LIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.175 Define GFLIB_LIMIT_DEFAULT_F32

```
#include <GFLIB_Limit.h>
```

9.175.1 Macro Definition

```
#define GFLIB_LIMIT_DEFAULT_F32 {INT32_MIN,INT32_MAX }
```

9.175.2 Description

Default value for GFLIB_LIMIT_T_F32.

9.176 Define GFLIB_LIMIT_DEFAULT_F16

```
#include <GFLIB_Limit.h>
```

9.176.1 Macro Definition

```
#define GFLIB_LIMIT_DEFAULT_F16 {INT16_MIN,INT16_MAX }
```

9.176.2 Description

Default value for GFLIB_LIMIT_T_F16.

9.177 Define GFLIB_LIMIT_DEFAULT_FLT

```
#include <GFLIB_Limit.h>
```

9.177.1 Macro Definition

```
#define GFLIB_LIMIT_DEFAULT_FLT {FLOAT_MIN,FLOAT_MAX }
```

9.177.2 Description

Default value for GFLIB_LIMIT_T_FLT.

9.178 Define GFLIB_LOG10_GET_FLOAT_WORD

```
#include <GFLIB_Log10.c>
```

9.178.1 Macro Definition

```
#define GFLIB_LOG10_GET_FLOAT_WORD (lx) = ((gflib_log10_float2word_ptr)&(x))->u32
```

9.178.2 Description

9.179 Define GFLIB_LOG10_SET_FLOAT_WORD

```
#include <GFLIB_Log10.c>
```

9.179.1 Macro Definition

```
#define GFLIB_LOG10_SET_FLOAT_WORD ((gflib_log10_float2word_ptr)&(x))->u32 = (lx)
```

9.179.2 Description

9.180 Define GFLIB_Log10

```
#include <GFLIB_Log10.h>
```

9.180.1 Macro Definition

```
#define GFLIB_Log10 macro_dispatcher(GFLIB_Log10, __VA_ARGS__)(__VA_ARGS__)
```

9.180.2 Description

This function implements polynomial approximation of arcsine function.

9.181 Define GFLIB_LOG10_T

```
#include <GFLIB_Log10.h>
```

9.181.1 Macro Definition

```
#define GFLIB_LOG10_T GFLIB_LOG10_T_FLT
```

9.181.2 Description

Definition of GFLIB_LOG10_T as alias for GFLIB_LOG10_T_FLT datatype in case the single precision floating point implementation is selected.

9.182 Define GFLIB_LOG10_DEFAULT

```
#include <GFLIB_Log10.h>
```

9.182.1 Macro Definition

```
#define GFLIB_LOG10_DEFAULT GFLIB_LOG10_DEFAULT_FLT
```

9.182.2 Description

Definition of GFLIB_LOG10_DEFAULT as alias for GFLIB_LOG10_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.183 Define GFLIB_LOG10_DEFAULT_FLT

```
#include <GFLIB_Log10.h>
```

9.183.1 Macro Definition

```
#define GFLIB_LOG10_DEFAULT_FLT &fltgflibLog10Coef
```

9.183.2 Description

Default approximation coefficients for GFLIB_Log10_FLT function.

9.184 Define GFLIB_LowerLimit

```
#include <GFLIB_LowerLimit.h>
```

9.184.1 Macro Definition

```
#define GFLIB_LowerLimit macro_dispatcher(GFLIB_LowerLimit, __VA_ARGS__)(__VA_ARGS__)
```

9.184.2 Description

This function tests whether the input value is above the lower limit.

9.185 Define GFLIB_LOWERLIMIT_T

```
#include <GFLIB_LowerLimit.h>
```

9.185.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_T GFLIB_LOWERLIMIT_T_FLT
```

9.185.2 Description

Definition of GFLIB_LOWERLIMIT_T as alias for GFLIB_LOWERLIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_T as alias for GFLIB_LOWERLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_T as alias for GFLIB_LOWERLIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

9.186 Define GFLIB_LOWERLIMIT_T

```
#include <GFLIB_LowerLimit.h>
```

9.186.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_T GFLIB_LOWERLIMIT_T_FLT
```

9.186.2 Description

Definition of GFLIB_LOWERLIMIT_T as alias for GFLIB_LOWERLIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_T as alias for GFLIB_LOWERLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_T as alias for GFLIB_LOWERLIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

9.187 Define GFLIB_LOWERLIMIT_T

```
#include <GFLIB_LowerLimit.h>
```

9.187.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_T GFLIB_LOWERLIMIT_T_FLT
```

9.187.2 Description

Definition of GFLIB_LOWERLIMIT_T as alias for GFLIB_LOWERLIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_T as alias for GFLIB_LOWERLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_T as alias for GFLIB_LOWERLIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

9.188 Define GFLIB_LOWERLIMIT_DEFAULT

```
#include <GFLIB_LowerLimit.h>
```

9.188.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_DEFAULT GFLIB_LOWERLIMIT_DEFAULT_FLT
```

9.188.2 Description

Definition of GFLIB_LOWERLIMIT_DEFAULT as alias for GFLIB_LOWERLIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_DEFAULT as alias for GFLIB_LOWERLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_DEFAULT as alias for GFLIB_LOWERLIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.189 Define GFLIB_LOWERLIMIT_DEFAULT

```
#include <GFLIB_LowerLimit.h>
```

9.189.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_DEFAULT GFLIB_LOWERLIMIT_DEFAULT_FLT
```

9.189.2 Description

Definition of GFLIB_LOWERLIMIT_DEFAULT as alias for GFLIB_LOWERLIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_DEFAULT as alias for GFLIB_LOWERLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_DEFAULT as alias for GFLIB_LOWERLIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.190 Define GFLIB_LOWERLIMIT_DEFAULT

```
#include <GFLIB_LowerLimit.h>
```

9.190.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_DEFAULT GFLIB_LOWERLIMIT_DEFAULT_FLT
```

9.190.2 Description

Definition of GFLIB_LOWERLIMIT_DEFAULT as alias for GFLIB_LOWERLIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_DEFAULT as alias for GFLIB_LOWERLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_DEFAULT as alias for GFLIB_LOWERLIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.191 Define GFLIB_LOWERLIMIT_DEFAULT_F32

```
#include <GFLIB_LowerLimit.h>
```

9.191.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_DEFAULT_F32 {INT32_MIN }
```

9.191.2 Description

Default value for GFLIB_LOWERLIMIT_T_F32.

9.192 Define GFLIB_LOWERLIMIT_DEFAULT_F16

```
#include <GFLIB_LowerLimit.h>
```

9.192.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_DEFAULT_F16 {INT16_MIN }
```

9.192.2 Description

Default value for GFLIB_LOWERLIMIT_T_F16.

9.193 Define GFLIB_LOWERLIMIT_DEFAULT_FLT

```
#include <GFLIB_LowerLimit.h>
```

9.193.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_DEFAULT_FLT {FLOAT_MIN }
```

9.193.2 Description

Default value for GFLIB_LOWERLIMIT_T_FLT.

9.194 Define GFLIB_Lut1D

```
#include <GFLIB_Lut1D.h>
```

9.194.1 Macro Definition

```
#define GFLIB_Lut1D macro_dispatcher(GFLIB_Lut1D, __VA_ARGS__)(__VA_ARGS__)
```

9.194.2 Description

This function implements the one-dimensional look-up table.

9.195 Define GFLIB_LUT1D_T

```
#include <GFLIB_Lut1D.h>
```

9.195.1 Macro Definition

```
#define GFLIB_LUT1D_T GFLIB_LUT1D_T_FLT
```

9.195.2 Description

Definition of GFLIB_LUT1D_T as alias for GFLIB_LUT1D_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_T as alias for GFLIB_LUT1D_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_T as alias for GFLIB_LUT1D_T_FLT datatype in case the single precision floating point implementation is selected.

9.196 Define GFLIB_LUT1D_T

```
#include <GFLIB_Lut1D.h>
```

9.196.1 Macro Definition

```
#define GFLIB_LUT1D_T GFLIB_LUT1D_T_FLT
```

9.196.2 Description

Definition of GFLIB_LUT1D_T as alias for GFLIB_LUT1D_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_T as alias for GFLIB_LUT1D_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_T as alias for GFLIB_LUT1D_T_FLT datatype in case the single precision floating point implementation is selected.

9.197 Define GFLIB_LUT1D_T

```
#include <GFLIB_Lut1D.h>
```

9.197.1 Macro Definition

```
#define GFLIB_LUT1D_T GFLIB_LUT1D_T_FLT
```

9.197.2 Description

Definition of GFLIB_LUT1D_T as alias for GFLIB_LUT1D_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_T as alias for GFLIB_LUT1D_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_T as alias for GFLIB_LUT1D_T_FLT datatype in case the single precision floating point implementation is selected.

9.198 Define GFLIB_LUT1D_DEFAULT

```
#include <GFLIB_Lut1D.h>
```

9.198.1 Macro Definition

```
#define GFLIB_LUT1D_DEFAULT GFLIB_LUT1D_DEFAULT_FLT
```

9.198.2 Description

Definition of GFLIB_LUT1D_DEFAULT as alias for GFLIB_LUT1D_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_DEFAULT as alias for GFLIB_LUT1D_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_DEFAULT as alias for GFLIB_LUT1D_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.199 Define GFLIB_LUT1D_DEFAULT

```
#include <GFLIB_Lut1D.h>
```

9.199.1 Macro Definition

```
#define GFLIB_LUT1D_DEFAULT GFLIB_LUT1D_DEFAULT_FLT
```

9.199.2 Description

Definition of GFLIB_LUT1D_DEFAULT as alias for GFLIB_LUT1D_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_DEFAULT as alias for GFLIB_LUT1D_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_DEFAULT as alias for GFLIB_LUT1D_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.200 Define GFLIB_LUT1D_DEFAULT

```
#include <GFLIB_Lut1D.h>
```

9.200.1 Macro Definition

```
#define GFLIB_LUT1D_DEFAULT GFLIB_LUT1D_DEFAULT_FLT
```

9.200.2 Description

Definition of GFLIB_LUT1D_DEFAULT as alias for GFLIB_LUT1D_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_DEFAULT as alias for GFLIB_LUT1D_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_DEFAULT as alias for GFLIB_LUT1D_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.201 Define GFLIB_LUT1D_DEFAULT_F32

```
#include <GFLIB_Lut1D.h>
```

9.201.1 Macro Definition

```
#define GFLIB_LUT1D_DEFAULT_F32 {(tU32)0,(tFrac32 *)0}
```

9.201.2 Description

Default value for GFLIB_LUT1D_T_F32.

9.202 Define GFLIB_LUT1D_DEFAULT_F16

```
#include <GFLIB_Lut1D.h>
```

9.202.1 Macro Definition

```
#define GFLIB_LUT1D_DEFAULT_F16 {(tU16)0,(tFrac16 *)0}
```

9.202.2 Description

Default value for GFLIB_LUT1D_T_F16.

9.203 Define GFLIB_LUT1D_DEFAULT_FLT

```
#include <GFLIB_Lut1D.h>
```

9.203.1 Macro Definition

```
#define GFLIB_LUT1D_DEFAULT_FLT {(tU32)0,(tFloat *)0}
```

9.203.2 Description

Default value for GFLIB_LUT1D_T_FLT.

9.204 Define GFLIB_Lut2D

```
#include <GFLIB_Lut2D.h>
```

9.204.1 Macro Definition

```
#define GFLIB_Lut2D macro_dispatcher(GFLIB_Lut2D, __VA_ARGS__)(__VA_ARGS__)
```

9.204.2 Description

This function implements the two-dimensional look-up table.

9.205 Define GFLIB_LUT2D_T

```
#include <GFLIB_Lut2D.h>
```

9.205.1 Macro Definition

```
#define GFLIB_LUT2D_T GFLIB_LUT2D_T_FLT
```

9.205.2 Description

Definition of GFLIB_LUT2D_T as alias for GFLIB_LUT2D_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_T as alias for GFLIB_LUT2D_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_T as alias for GFLIB_LUT2D_T_FLT datatype in case the single precision floating point implementation is selected.

9.206 Define GFLIB_LUT2D_T

```
#include <GFLIB_Lut2D.h>
```

9.206.1 Macro Definition

```
#define GFLIB_LUT2D_T GFLIB_LUT2D_T_FLT
```

9.206.2 Description

Definition of GFLIB_LUT2D_T as alias for GFLIB_LUT2D_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_T as alias for GFLIB_LUT2D_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_T as alias for GFLIB_LUT2D_T_FLT datatype in case the single precision floating point implementation is selected.

9.207 Define GFLIB_LUT2D_T

```
#include <GFLIB_Lut2D.h>
```

9.207.1 Macro Definition

```
#define GFLIB_LUT2D_T GFLIB_LUT2D_T_FLT
```

9.207.2 Description

Definition of GFLIB_LUT2D_T as alias for GFLIB_LUT2D_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_T as alias for GFLIB_LUT2D_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_T as alias for GFLIB_LUT2D_T_FLT datatype in case the single precision floating point implementation is selected.

9.208 Define GFLIB_LUT2D_DEFAULT

```
#include <GFLIB_Lut2D.h>
```

9.208.1 Macro Definition

```
#define GFLIB_LUT2D_DEFAULT GFLIB_LUT2D_DEFAULT_FLT
```

9.208.2 Description

Definition of GFLIB_LUT2D_DEFAULT as alias for GFLIB_LUT2D_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_DEFAULT as alias for GFLIB_LUT2D_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_DEFAULT as alias for GFLIB_LUT2D_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.209 Define GFLIB_LUT2D_DEFAULT

```
#include <GFLIB_Lut2D.h>
```

9.209.1 Macro Definition

```
#define GFLIB_LUT2D_DEFAULT GFLIB_LUT2D_DEFAULT_FLT
```

9.209.2 Description

Definition of GFLIB_LUT2D_DEFAULT as alias for GFLIB_LUT2D_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_DEFAULT as alias for GFLIB_LUT2D_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_DEFAULT as alias for GFLIB_LUT2D_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.210 Define GFLIB_LUT2D_DEFAULT

```
#include <GFLIB_Lut2D.h>
```

9.210.1 Macro Definition

```
#define GFLIB_LUT2D_DEFAULT GFLIB_LUT2D_DEFAULT_FLT
```

9.210.2 Description

Definition of GFLIB_LUT2D_DEFAULT as alias for GFLIB_LUT2D_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_DEFAULT as alias for GFLIB_LUT2D_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_DEFAULT as alias for GFLIB_LUT2D_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.211 Define GFLIB_LUT2D_DEFAULT_F32

```
#include <GFLIB_Lut2D.h>
```

9.211.1 Macro Definition

```
#define GFLIB_LUT2D_DEFAULT_F32 {(tU32)0,(tU32)0,(tFrac32 *)0}
```

9.211.2 Description

Default value for GFLIB_LUT2D_T_F32.

9.212 Define GFLIB_LUT2D_DEFAULT_F16

```
#include <GFLIB_Lut2D.h>
```

9.212.1 Macro Definition

```
#define GFLIB_LUT2D_DEFAULT_F16 {(tU16)0,(tU16)0,(tFrac16 *)0}
```

9.212.2 Description

Default value for GFLIB_LUT2D_T_F16.

9.213 Define GFLIB_LUT2D_DEFAULT_FLT

```
#include <GFLIB_Lut2D.h>
```

9.213.1 Macro Definition

```
#define GFLIB_LUT2D_DEFAULT_FLT {(tU32)0,(tU32)0,(tFloat *)0}
```

9.213.2 Description

Default value for GFLIB_LUT2D_T_FLT.

9.214 Define GFLIB_Ramp

```
#include <GFLIB_Ramp.h>
```

9.214.1 Macro Definition

```
#define GFLIB_Ramp macro_dispatcher(GFLIB_Ramp, __VA_ARGS__)(__VA_ARGS__)
```

9.214.2 Description

The function calculates the up/down ramp with the step increment/decrement defined in the pParam structure.

9.215 Define GFLIB_RAMP_T

```
#include <GFLIB_Ramp.h>
```

9.215.1 Macro Definition

```
#define GFLIB_RAMP_T GFLIB_RAMP_T_FLT
```

9.215.2 Description

Definition of GFLIB_RAMP_T as alias for GFLIB_RAMP_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_RAMP_T as alias for GFLIB_RAMP_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_RAMP_T as alias for GFLIB_RAMP_T_FLT datatype in case the single precision floating point implementation is selected.

9.216 Define GFLIB_RAMP_T

```
#include <GFLIB_Ramp.h>
```

9.216.1 Macro Definition

```
#define GFLIB_RAMP_T GFLIB_RAMP_T_FLT
```

9.216.2 Description

Definition of GFLIB_RAMP_T as alias for GFLIB_RAMP_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_RAMP_T as alias for GFLIB_RAMP_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_RAMP_T as alias for GFLIB_RAMP_T_FLT datatype in case the single precision floating point implementation is selected.

9.217 Define GFLIB_RAMP_T

```
#include <GFLIB_Ramp.h>
```

9.217.1 Macro Definition

```
#define GFLIB_RAMP_T GFLIB_RAMP_T_FLT
```

9.217.2 Description

Definition of GFLIB_RAMP_T as alias for GFLIB_RAMP_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_RAMP_T as alias for GFLIB_RAMP_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_RAMP_T as alias for GFLIB_RAMP_T_FLT datatype in case the single precision floating point implementation is selected.

9.218 Define GFLIB_RAMP_DEFAULT

```
#include <GFLIB_Ramp.h>
```

9.218.1 Macro Definition

```
#define GFLIB_RAMP_DEFAULT GFLIB_RAMP_DEFAULT_FLT
```

9.218.2 Description

Definition of GFLIB_RAMP_DEFAULT as alias for GFLIB_RAMP_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_RAMP_DEFAULT as alias for GFLIB_RAMP_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_RAMP_DEFAULT as alias for GFLIB_RAMP_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.219 Define GFLIB_RAMP_DEFAULT

```
#include <GFLIB_Ramp.h>
```

9.219.1 Macro Definition

```
#define GFLIB_RAMP_DEFAULT GFLIB_RAMP_DEFAULT_FLT
```

9.219.2 Description

Definition of GFLIB_RAMP_DEFAULT as alias for GFLIB_RAMP_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_RAMP_DEFAULT as alias for GFLIB_RAMP_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_RAMP_DEFAULT as alias for GFLIB_RAMP_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.220 Define GFLIB_RAMP_DEFAULT

```
#include <GFLIB_Ramp.h>
```

9.220.1 Macro Definition

```
#define GFLIB_RAMP_DEFAULT GFLIB_RAMP_DEFAULT_FLT
```

9.220.2 Description

Definition of GFLIB_RAMP_DEFAULT as alias for GFLIB_RAMP_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_RAMP_DEFAULT as alias for GFLIB_RAMP_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_RAMP_DEFAULT as alias for GFLIB_RAMP_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.221 Define GFLIB_RAMP_DEFAULT_F32

```
#include <GFLIB_Ramp.h>
```

9.221.1 Macro Definition

```
#define GFLIB_RAMP_DEFAULT_F32 {(tFrac32)0,(tFrac32)0,(tFrac32)0}
```

9.221.2 Description

Default value for GFLIB_RAMP_T_F32.

9.222 Define GFLIB_RAMP_DEFAULT_F16

```
#include <GFLIB_Ramp.h>
```

9.222.1 Macro Definition

```
#define GFLIB_RAMP_DEFAULT_F16 {(tFrac16)0,(tFrac16)0,(tFrac16)0}
```

9.222.2 Description

Default value for GFLIB_RAMP_T_F16.

9.223 Define GFLIB_RAMP_DEFAULT_FLT

```
#include <GFLIB_Ramp.h>
```

9.223.1 Macro Definition

```
#define GFLIB_RAMP_DEFAULT_FLT {(tFloat)0,(tFloat)0,(tFloat)0}
```

9.223.2 Description

Default value for GFLIB_RAMP_T_FLT.

9.224 Define GFLIB_Sign

```
#include <GFLIB_Sign.h>
```

9.224.1 Macro Definition

```
#define GFLIB_Sign macro_dispatcher(GFLIB_Sign, __VA_ARGS__)(__VA_ARGS__)
```

9.224.2 Description

This function returns the signum of input value.

9.225 Define GFLIB_SIN_FLT_MIN

```
#include <GFLIB_Sin.c>
```

9.225.1 Macro Definition

```
#define GFLIB_SIN_FLT_MIN ((tFloat)1.220703125000000e-04)
```

9.225.2 Description

Floating-point min. input value for computation.

9.226 Define GFLIB_Sin

```
#include <GFLIB_Sin.h>
```

9.226.1 Macro Definition

```
#define GFLIB_Sin macro_dispatcher(GFLIB_Sin, __VA_ARGS__)(__VA_ARGS__)
```

9.226.2 Description

This function implements polynomial approximation of sine function.

9.227 Define GFLIB_SIN_T

```
#include <GFLIB_Sin.h>
```

9.227.1 Macro Definition

```
#define GFLIB_SIN_T GFLIB_SIN_T_FLT
```

9.227.2 Description

Definition of GFLIB_SIN_T as alias for GFLIB_SIN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_SIN_T as alias for GFLIB_SIN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_SIN_T as alias for GFLIB_SIN_T_FLT datatype in case the single precision floating point implementation is selected.

9.228 Define GFLIB_SIN_T

```
#include <GFLIB_Sin.h>
```

9.228.1 Macro Definition

```
#define GFLIB_SIN_T GFLIB_SIN_T_FLT
```

9.228.2 Description

Definition of GFLIB_SIN_T as alias for GFLIB_SIN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_SIN_T as alias for GFLIB_SIN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_SIN_T as alias for GFLIB_SIN_T_FLT datatype in case the single precision floating point implementation is selected.

9.229 Define GFLIB_SIN_T

```
#include <GFLIB_Sin.h>
```

9.229.1 Macro Definition

```
#define GFLIB_SIN_T GFLIB_SIN_T_FLT
```

9.229.2 Description

Definition of GFLIB_SIN_T as alias for GFLIB_SIN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_SIN_T as alias for GFLIB_SIN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_SIN_T as alias for GFLIB_SIN_T_FLT datatype in case the single precision floating point implementation is selected.

9.230 Define GFLIB_SIN_DEFAULT

```
#include <GFLIB_Sin.h>
```

9.230.1 Macro Definition

```
#define GFLIB_SIN_DEFAULT GFLIB_SIN_DEFAULT_FLT
```

9.230.2 Description

Definition of GFLIB_SIN_DEFAULT as alias for GFLIB_SIN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_SIN_DEFAULT as alias for GFLIB_SIN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_SIN_DEFAULT as alias for GFLIB_SIN_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.231 Define GFLIB_SIN_DEFAULT

```
#include <GFLIB_Sin.h>
```

9.231.1 Macro Definition

```
#define GFLIB_SIN_DEFAULT GFLIB_SIN_DEFAULT_FLT
```

9.231.2 Description

Definition of GFLIB_SIN_DEFAULT as alias for GFLIB_SIN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_SIN_DEFAULT as alias for GFLIB_SIN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_SIN_DEFAULT as alias for GFLIB_SIN_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.232 Define GFLIB_SIN_DEFAULT

```
#include <GFLIB_Sin.h>
```

9.232.1 Macro Definition

```
#define GFLIB_SIN_DEFAULT GFLIB_SIN_DEFAULT_FLT
```

9.232.2 Description

Definition of GFLIB_SIN_DEFAULT as alias for GFLIB_SIN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_SIN_DEFAULT as alias for GFLIB_SIN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_SIN_DEFAULT as alias for GFLIB_SIN_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.233 Define GFLIB_SIN_DEFAULT_F32

```
#include <GFLIB_Sin.h>
```

9.233.1 Macro Definition

```
#define GFLIB_SIN_DEFAULT_F32 &f32gflibSinCoef
```

9.233.2 Description

Default approximation coefficients for GFLIB_Sin_F32 function.

9.234 Define GFLIB_SIN_DEFAULT_F16

```
#include <GFLIB_Sin.h>
```

9.234.1 Macro Definition

```
#define GFLIB_SIN_DEFAULT_F16 &f16gflibSinCoef
```

9.234.2 Description

Default approximation coefficients for GFLIB_Sin_F16 function.

9.235 Define GFLIB_SIN_DEFAULT_FLT

```
#include <GFLIB_Sin.h>
```

9.235.1 Macro Definition

```
#define GFLIB_SIN_DEFAULT_FLT &fltgflibSinCoef
```

9.235.2 Description

Default approximation coefficients for GFLIB_Sin_FLT function.

9.236 Define GFLIB_SINCOS_FLT_MIN

```
#include <GFLIB_SinCos.c>
```

9.236.1 Macro Definition

```
#define GFLIB_SINCOS_FLT_MIN ((tFloat)1.220703125000000e-04)
```

9.236.2 Description

Floating-point min. input value for computation.

9.237 Define GFLIB_SinCos

```
#include <GFLIB_SinCos.h>
```

9.237.1 Macro Definition

```
#define GFLIB_SinCos macro_dispatcher(GFLIB_SinCos, __VA_ARGS__)(__VA_ARGS__)
```

9.237.2 Description

This function implements polynomial approximation of sine function.

9.238 Define GFLIB_SINCOS_T

```
#include <GFLIB_SinCos.h>
```

9.238.1 Macro Definition

```
#define GFLIB_SINCOS_T GFLIB_SINCOS_T_FLT
```

9.238.2 Description

Definition of GFLIB_SINCOS_T as alias for GFLIB_SINCOS_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_SINCOS_T as alias for GFLIB_SINCOS_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_SINCOS_T as alias for GFLIB_SINCOS_T_FLT datatype in case the single precision floating point implementation is selected.

9.239 Define GFLIB_SINCOS_T

```
#include <GFLIB_SinCos.h>
```

9.239.1 Macro Definition

```
#define GFLIB_SINCOS_T GFLIB_SINCOS_T_FLT
```

9.239.2 Description

Definition of GFLIB_SINCOS_T as alias for GFLIB_SINCOS_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_SINCOS_T as alias for GFLIB_SINCOS_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_SINCOS_T as alias for GFLIB_SINCOS_T_FLT datatype in case the single precision floating point implementation is selected.

9.240 Define GFLIB_SINCOS_T

```
#include <GFLIB_SinCos.h>
```

9.240.1 Macro Definition

```
#define GFLIB_SINCOS_T GFLIB_SINCOS_T_FLT
```

9.240.2 Description

Definition of GFLIB_SINCOS_T as alias for GFLIB_SINCOS_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_SINCOS_T as alias for GFLIB_SINCOS_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_SINCOS_T as alias for GFLIB_SINCOS_T_FLT datatype in case the single precision floating point implementation is selected.

9.241 Define GFLIB_SINCOS_DEFAULT

```
#include <GFLIB_SinCos.h>
```

9.241.1 Macro Definition

```
#define GFLIB_SINCOS_DEFAULT GFLIB_SINCOS_DEFAULT_FLT
```

9.241.2 Description

Definition of GFLIB_SINCOS_DEFAULT as alias for GFLIB_SINCOS_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_SINCOS_DEFAULT as alias for GFLIB_SINCOS_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_SINCOS_DEFAULT as alias for GFLIB_SINCOS_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.242 Define GFLIB_SINCOS_DEFAULT

```
#include <GFLIB_SinCos.h>
```

9.242.1 Macro Definition

```
#define GFLIB_SINCOS_DEFAULT GFLIB_SINCOS_DEFAULT_FLT
```

9.242.2 Description

Definition of GFLIB_SINCOS_DEFAULT as alias for GFLIB_SINCOS_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_SINCOS_DEFAULT as alias for GFLIB_SINCOS_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_SINCOS_DEFAULT as alias for GFLIB_SINCOS_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.243 Define GFLIB_SINCOS_DEFAULT

```
#include <GFLIB_SinCos.h>
```

9.243.1 Macro Definition

```
#define GFLIB_SINCOS_DEFAULT GFLIB_SINCOS_DEFAULT_FLT
```

9.243.2 Description

Definition of GFLIB_SINCOS_DEFAULT as alias for GFLIB_SINCOS_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_SINCOS_DEFAULT as alias for GFLIB_SINCOS_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_SINCOS_DEFAULT as alias for GFLIB_SINCOS_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.244 Define GFLIB_SINCOS_DEFAULT_F32

```
#include <GFLIB_SinCos.h>
```

9.244.1 Macro Definition

```
#define GFLIB_SINCOS_DEFAULT_F32 &f32gflibSinCosCoef
```

9.244.2 Description

Default approximation coefficients for GFLIB_SinCos_F32 function.

9.245 Define GFLIB_SINCOS_DEFAULT_F16

```
#include <GFLIB_SinCos.h>
```

9.245.1 Macro Definition

```
#define GFLIB_SINCOS_DEFAULT_F16 &f16gflibSinCosCoef
```

9.245.2 Description

Default approximation coefficients for GFLIB_SinCos_F16 function.

9.246 Define GFLIB_SINCOS_DEFAULT_FLT

```
#include <GFLIB_SinCos.h>
```

9.246.1 Macro Definition

```
#define GFLIB_SINCOS_DEFAULT_FLT &fltgflibSinCosCoef
```

9.246.2 Description

Default approximation coefficients for GFLIB_SinCos_FLT function.

9.247 Define GFLIB_Sqrt

```
#include <GFLIB_Sqrt.h>
```

9.247.1 Macro Definition

```
#define GFLIB_Sqrt macro_dispatcher(GFLIB_Sqrt, __VA_ARGS__)(__VA_ARGS__)
```

9.247.2 Description

This function returns the square root of input value.

9.248 Define GFLIB_TAN_FLT_MIN

```
#include <GFLIB_Tan.c>
```

9.248.1 Macro Definition

```
#define GFLIB_TAN_FLT_MIN ((tFloat)2.441406250000000e-04)
```

9.248.2 Description

Floating-point min. input value for computation.

9.249 Define GFLIB_TAN_FLT_3P4

```
#include <GFLIB_Tan.c>
```

9.249.1 Macro Definition

```
#define GFLIB_TAN_FLT_3P4 ((tFloat)2.356194490192345e+00)
```

9.249.2 Description

Floating-point constant 3*PI/4.

9.250 Define GFLIB_TAN_FLT_PI

```
#include <GFLIB_Tan.c>
```

9.250.1 Macro Definition

```
#define GFLIB_TAN_FLT_PI ((tFloat)3.141592653589793e+00)
```

9.250.2 Description

Floating-point constant PI.

9.251 Define GFLIB_TAN_FLT_CORR1

```
#include <GFLIB_Tan.c>
```

9.251.1 Macro Definition

```
#define GFLIB_TAN_FLT_CORR1 ((tFloat)-8.742278012618954e-08)
```

9.251.2 Description

Floating-point correction constant PI_double - PI_single.

9.252 Define GFLIB_TAN_FLT_PI4

```
#include <GFLIB_Tan.c>
```

9.252.1 Macro Definition

```
#define GFLIB_TAN_FLT_PI4 ((tFloat)7.853981633974483e-01)
```

9.252.2 Description

Floating-point constant PI/4.

9.253 Define GFLIB_TAN_FLT_PI2

```
#include <GFLIB_Tan.c>
```

9.253.1 Macro Definition

```
#define GFLIB_TAN_FLT_PI2 ((tFloat)1.570796326794897e+00)
```

9.253.2 Description

Floating-point constant PI/2.

9.254 Define GFLIB_TAN_FLT_CORR2

```
#include <GFLIB_Tan.c>
```

9.254.1 Macro Definition

```
#define GFLIB_TAN_FLT_CORR2 ((tFloat)-4.371139006309477e-08)
```

9.254.2 Description

Floating-point correction constant PI_double/2 - PI_single/2.

9.255 Define GFLIB_Tan

```
#include <GFLIB_Tan.h>
```

9.255.1 Macro Definition

```
#define GFLIB_Tan macro_dispatcher(GFLIB_Tan, __VA_ARGS__)(__VA_ARGS__)
```

9.255.2 Description

This function implements polynomial approximation of tangent function.

9.256 Define GFLIB_TAN_T

```
#include <GFLIB_Tan.h>
```

9.256.1 Macro Definition

```
#define GFLIB_TAN_T GFLIB_TAN_T_FLT
```

9.256.2 Description

Definition of GFLIB_TAN_T as alias for GFLIB_TAN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_TAN_T as alias for GFLIB_TAN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_TAN_T as alias for GFLIB_TAN_T_FLT datatype in case the single precision floating point implementation is selected.

9.257 Define GFLIB_TAN_T

```
#include <GFLIB_Tan.h>
```

9.257.1 Macro Definition

```
#define GFLIB_TAN_T GFLIB_TAN_T_FLT
```

9.257.2 Description

Definition of GFLIB_TAN_T as alias for GFLIB_TAN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_TAN_T as alias for GFLIB_TAN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_TAN_T as alias for GFLIB_TAN_T_FLT datatype in case the single precision floating point implementation is selected.

9.258 Define GFLIB_TAN_T

```
#include <GFLIB_Tan.h>
```

9.258.1 Macro Definition

```
#define GFLIB_TAN_T GFLIB_TAN_T_FLT
```

9.258.2 Description

Definition of GFLIB_TAN_T as alias for GFLIB_TAN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_TAN_T as alias for GFLIB_TAN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_TAN_T as alias for GFLIB_TAN_T_FLT datatype in case the single precision floating point implementation is selected.

9.259 Define GFLIB_TAN_DEFAULT

```
#include <GFLIB_Tan.h>
```

9.259.1 Macro Definition

```
#define GFLIB_TAN_DEFAULT GFLIB_TAN_DEFAULT_FLT
```

9.259.2 Description

Definition of GFLIB_TAN_DEFAULT as alias for GFLIB_TAN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_TAN_DEFAULT as alias for GFLIB_TAN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_TAN_DEFAULT as alias for GFLIB_TAN_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.260 Define GFLIB_TAN_DEFAULT

```
#include <GFLIB_Tan.h>
```

9.260.1 Macro Definition

```
#define GFLIB_TAN_DEFAULT GFLIB_TAN_DEFAULT_FLT
```

9.260.2 Description

Definition of GFLIB_TAN_DEFAULT as alias for GFLIB_TAN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_TAN_DEFAULT as alias for GFLIB_TAN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_TAN_DEFAULT as alias for GFLIB_TAN_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.261 Define GFLIB_TAN_DEFAULT

```
#include <GFLIB_Tan.h>
```

9.261.1 Macro Definition

```
#define GFLIB_TAN_DEFAULT GFLIB_TAN_DEFAULT_FLT
```

9.261.2 Description

Definition of GFLIB_TAN_DEFAULT as alias for GFLIB_TAN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_TAN_DEFAULT as alias for GFLIB_TAN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_TAN_DEFAULT as alias for GFLIB_TAN_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.262 Define GFLIB_TAN_DEFAULT_F32

```
#include <GFLIB_Tan.h>
```

9.262.1 Macro Definition

```
#define GFLIB_TAN_DEFAULT_F32 &f32gflibTanCoef
```

9.262.2 Description

Default approximation coefficients for GFLIB_Tan_F32 function.

9.263 Define GFLIB_TAN_DEFAULT_F16

```
#include <GFLIB_Tan.h>
```

9.263.1 Macro Definition

```
#define GFLIB_TAN_DEFAULT_F16 &f16gflibTanCoef
```

9.263.2 Description

Default approximation coefficients for GFLIB_Tan_F16 function.

9.264 Define GFLIB_TAN_DEFAULT_FLT

```
#include <GFLIB_Tan.h>
```

9.264.1 Macro Definition

```
#define GFLIB_TAN_DEFAULT_FLT &fltgflibTanCoef
```

9.264.2 Description

Default approximation coefficients for GFLIB_Tan_FLT function.

9.265 Define GFLIB_UpperLimit

```
#include <GFLIB_UpperLimit.h>
```

9.265.1 Macro Definition

```
#define GFLIB_UpperLimit macro_dispatcher(GFLIB_UpperLimit, __VA_ARGS__)(__VA_ARGS__)
```

9.265.2 Description

This function tests whether the input value is below the upper limit.

9.266 Define GFLIB_UPPERLIMIT_T

```
#include <GFLIB_UpperLimit.h>
```

9.266.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_T GFLIB_UPPERLIMIT_T_FLT
```

9.266.2 Description

Definition of GFLIB_UPPERLIMIT_T as alias for GFLIB_UPPERLIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_T as alias for GFLIB_UPPERLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_T as alias for GFLIB_UPPERLIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

9.267 Define GFLIB_UPPERLIMIT_T

```
#include <GFLIB_UpperLimit.h>
```

9.267.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_T GFLIB_UPPERLIMIT_T_FLT
```

9.267.2 Description

Definition of GFLIB_UPPERLIMIT_T as alias for GFLIB_UPPERLIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_T as alias for GFLIB_UPPERLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_T as alias for GFLIB_UPPERLIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

9.268 Define GFLIB_UPPERLIMIT_T

```
#include <GFLIB_UpperLimit.h>
```

9.268.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_T GFLIB_UPPERLIMIT_T_FLT
```

9.268.2 Description

Definition of GFLIB_UPPERLIMIT_T as alias for GFLIB_UPPERLIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_T as alias for GFLIB_UPPERLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_T as alias for GFLIB_UPPERLIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

9.269 Define GFLIB_UPPERLIMIT_DEFAULT

```
#include <GFLIB_UpperLimit.h>
```

9.269.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_DEFAULT GFLIB_UPPERLIMIT_DEFAULT_FLT
```

9.269.2 Description

Definition of GFLIB_UPPERLIMIT_DEFAULT as alias for GFLIB_UPPERLIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_DEFAULT as alias for GFLIB_UPPERLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_DEFAULT as alias for GFLIB_UPPERLIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.270 Define GFLIB_UPPERLIMIT_DEFAULT

```
#include <GFLIB_UpperLimit.h>
```

9.270.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_DEFAULT GFLIB_UPPERLIMIT_DEFAULT_FLT
```

9.270.2 Description

Definition of GFLIB_UPPERLIMIT_DEFAULT as alias for GFLIB_UPPERLIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_DEFAULT as alias for GFLIB_UPPERLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_DEFAULT as alias for GFLIB_UPPERLIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.271 Define GFLIB_UPPERLIMIT_DEFAULT

```
#include <GFLIB_UpperLimit.h>
```

9.271.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_DEFAULT GFLIB_UPPERLIMIT_DEFAULT_FLT
```

9.271.2 Description

Definition of GFLIB_UPPERLIMIT_DEFAULT as alias for GFLIB_UPPERLIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_DEFAULT as alias for GFLIB_UPPERLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_DEFAULT as alias for GFLIB_UPPERLIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.272 Define GFLIB_UPPERLIMIT_DEFAULT_F32

```
#include <GFLIB_UpperLimit.h>
```

9.272.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_DEFAULT_F32 {INT32_MAX }
```

9.272.2 Description

Default value for GFLIB_UPPERLIMIT_T_F32.

9.273 Define GFLIB_UPPERLIMIT_DEFAULT_F16

```
#include <GFLIB_UpperLimit.h>
```

9.273.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_DEFAULT_F16 {INT16_MAX }
```

9.273.2 Description

Default value for GFLIB_UPPERLIMIT_T_F16.

9.274 Define GFLIB_UPPERLIMIT_DEFAULT_FLT

```
#include <GFLIB_UpperLimit.h>
```

9.274.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_DEFAULT_FLT {FLOAT_MAX }
```

9.274.2 Description

Default value for GFLIB_UPPERLIMIT_T_FLT.

9.275 Define GFLIB_VectorLimit

```
#include <GFLIB_VectorLimit.h>
```

9.275.1 Macro Definition

```
#define GFLIB_VectorLimit macro_dispatcher(GFLIB_VectorLimit, __VA_ARGS__)(__VA_ARGS__)
```

9.275.2 Description

This function limits the magnitude of the input vector.

9.276 Define GFLIB_VECTORLIMIT_T

```
#include <GFLIB_VectorLimit.h>
```

9.276.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_T GFLIB_VECTORLIMIT_T_FLT
```

9.276.2 Description

Definition of GFLIB_VECTORLIMIT_T as alias for GFLIB_VECTORLIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_T as alias for GFLIB_VECTORLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_T as alias for GFLIB_VECTORLIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

9.277 Define GFLIB_VECTORLIMIT_T

```
#include <GFLIB_VectorLimit.h>
```

9.277.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_T GFLIB_VECTORLIMIT_T_FLT
```

9.277.2 Description

Definition of GFLIB_VECTORLIMIT_T as alias for GFLIB_VECTORLIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_T as alias for GFLIB_VECTORLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_T as alias for GFLIB_VECTORLIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

9.278 Define GFLIB_VECTORLIMIT_T

```
#include <GFLIB_VectorLimit.h>
```

9.278.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_T GFLIB_VECTORLIMIT_T_FLT
```

9.278.2 Description

Definition of GFLIB_VECTORLIMIT_T as alias for GFLIB_VECTORLIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_T as alias for GFLIB_VECTORLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_T as alias for GFLIB_VECTORLIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

9.279 Define GFLIB_VECTORLIMIT_DEFAULT

```
#include <GFLIB_VectorLimit.h>
```

9.279.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_DEFAULT GFLIB_VECTORLIMIT_DEFAULT_FLT
```

9.279.2 Description

Definition of GFLIB_VECTORLIMIT_DEFAULT as alias for GFLIB_VECTORLIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_DEFAULT as alias for GFLIB_VECTORLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_DEFAULT as alias for GFLIB_VECTORLIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.280 Define GFLIB_VECTORLIMIT_DEFAULT

```
#include <GFLIB_VectorLimit.h>
```

9.280.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_DEFAULT GFLIB_VECTORLIMIT_DEFAULT_FLT
```

9.280.2 Description

Definition of GFLIB_VECTORLIMIT_DEFAULT as alias for GFLIB_VECTORLIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_DEFAULT as alias for GFLIB_VECTORLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_DEFAULT as alias for GFLIB_VECTORLIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.281 Define GFLIB_VECTORLIMIT_DEFAULT

```
#include <GFLIB_VectorLimit.h>
```

9.281.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_DEFAULT GFLIB_VECTORLIMIT_DEFAULT_FLT
```

9.281.2 Description

Definition of GFLIB_VECTORLIMIT_DEFAULT as alias for GFLIB_VECTORLIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_DEFAULT as alias for GFLIB_VECTORLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_DEFAULT as alias for GFLIB_VECTORLIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.282 Define GFLIB_VECTORLIMIT_DEFAULT_F32

```
#include <GFLIB_VectorLimit.h>
```

9.282.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_DEFAULT_F32 {(tFrac32)0}
```

9.282.2 Description

Default value for GFLIB_VECTORLIMIT_T_F32.

9.283 Define GFLIB_VECTORLIMIT_DEFAULT_F16

```
#include <GFLIB_VectorLimit.h>
```

9.283.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_DEFAULT_F16 {(tFrac16)0}
```

9.283.2 Description

Default value for GFLIBVECTORLIMIT_T_F16.

9.284 Define GFLIB_VECTORLIMIT_DEFAULT_FLT

```
#include <GFLIB_VectorLimit.h>
```

9.284.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_DEFAULT_FLT {(tFloat)0}
```

9.284.2 Description

Default value for GFLIB_VECTORLIMIT_T_FLT.

9.285 Define GFLIB_VLOG10_GET_FLOAT_WORD

```
#include <GFLIB_VLog10.c>
```

9.285.1 Macro Definition

```
#define GFLIB_VLOG10_GET_FLOAT_WORD (lx) = ((gflib_vlog10_float2word_ptr)&(x))->u32
```

9.285.2 Description

9.286 Define GFLIB_VLOG10_SET_FLOAT_WORD

```
#include <GFLIB_VLog10.c>
```

9.286.1 Macro Definition

```
#define GFLIB_VLOG10_SET_FLOAT_WORD ((gflib_vlog10_float2word_ptr)&(x))->u32 = (lx)
```

9.286.2 Description

9.287 Define GFLIB_VLog10

```
#include <GFLIB_VLog10.h>
```

9.287.1 Macro Definition

```
#define GFLIB_VLog10 macro_dispatcher(GFLIB_VLog10, __VA_ARGS__)(__VA_ARGS__)
```

9.287.2 Description

This function implements polynomial approximation of arcsine function.

9.288 Define GFLIB_VLOG10_T

```
#include <GFLIB_VLog10.h>
```

9.288.1 Macro Definition

```
#define GFLIB_VLOG10_T GFLIB_VLOG10_T_FLT
```

9.288.2 Description

Definition of GFLIB_VLOG10_T as alias for GFLIB_VLOG10_T_FLT datatype in case the single precision floating point implementation is selected.

9.289 Define GFLIB_VLOG10_DEFAULT

```
#include <GFLIB_VLog10.h>
```

9.289.1 Macro Definition

```
#define GFLIB_VLOG10_DEFAULT GFLIB_VLOG10_DEFAULT_FLT
```

9.289.2 Description

Definition of GFLIB_VLOG10_DEFAULT as alias for GFLIB_VLOG10_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.290 Define GFLIB_VLOG10_DEFAULT_FLT

```
#include <GFLIB_VLog10.h>
```

9.290.1 Macro Definition

```
#define GFLIB_VLOG10_DEFAULT_FLT &fltgflibVLog10Coef
```

9.290.2 Description

Default approximation coefficients for GFLIB_VLog10_FLT function.

9.291 Define GMCLIB_Clark

```
#include <GMCLIB_Clark.h>
```

9.291.1 Macro Definition

```
#define GMCLIB_Clark macro_dispatcher(GMCLIB_Clark, __VA_ARGS__)(__VA_ARGS__)
```

9.291.2 Description

This function implements the Clarke transformation.

9.292 Define GMCLIB_ClarkInv

```
#include <GMCLIB_ClarkInv.h>
```

9.292.1 Macro Definition

```
#define GMCLIB_ClarkInv macro_dispatcher(GMCLIB_ClarkInv, __VA_ARGS__)(__VA_ARGS__)
```

9.292.2 Description

This function implements the inverse Clarke transformation.

9.293 Define GMCLIB_DecouplingPMSM

```
#include <GMCLIB_DecouplingPMSM.h>
```

9.293.1 Macro Definition

```
#define GMCLIB_DecouplingPMSM macro_dispatcher(GMCLIB_DecouplingPMSM, __VA_ARGS__)
(__VA_ARGS__)
```

9.293.2 Description

This function calculates the cross-coupling voltages to eliminate the dq axis coupling causing non-linearity of the field oriented control.

9.294 Define GMCLIB_DECOUPLINGPMSM_T

```
#include <GMCLIB_DecouplingPMSM.h>
```

9.294.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_T GMCLIB_DECOUPLINGPMSM_T_FLT
```

9.294.2 Description

Definition of GMCLIB_DECOUPLINGPMSM_T as alias for GMCLIB_DECOUPLINGPMSM_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_T as alias for GMCLIB_DECOUPLINGPMSM_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_T as alias for GMCLIB_DECOUPLINGPMSM_T_FLT datatype in case the single precision floating point implementation is selected.

9.295 Define GMCLIB_DECOUPLINGPMSM_T

```
#include <GMCLIB_DecouplingPMSM.h>
```

9.295.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_T GMCLIB_DECOUPLINGPMSM_T_FLT
```

9.295.2 Description

Definition of GMCLIB_DECOUPLINGPMSM_T as alias for GMCLIB_DECOUPLINGPMSM_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_T as alias for GMCLIB_DECOUPLINGPMSM_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_T as alias for GMCLIB_DECOUPLINGPMSM_T_FLT datatype in case the single precision floating point implementation is selected.

9.296 Define GMCLIB_DECOUPLINGPMSM_T

```
#include <GMCLIB_DecouplingPMSM.h>
```

9.296.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_T GMCLIB_DECOUPLINGPMSM_T_FLT
```

9.296.2 Description

Definition of GMCLIB_DECOUPLINGPMSM_T as alias for GMCLIB_DECOUPLINGPMSM_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_T as alias for GMCLIB_DECOUPLINGPMSM_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_T as alias for GMCLIB_DECOUPLINGPMSM_T_FLT datatype in case the single precision floating point implementation is selected.

9.297 Define GMCLIB_DECOUPLINGPMSM_DEFAULT

```
#include <GMCLIB_DecouplingPMSM.h>
```

9.297.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT
```

9.297.2 Description

Definition of GMCLIB_DECOUPLINGPMSM_DEFAULT as alias for GMCLIB_DECOUPLINGPMSM_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_DEFAULT as alias for GMCLIB_DECOUPLINGPMSM_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_DEFAULT as alias for GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.298 Define GMCLIB_DECOUPLINGPMSM_DEFAULT

```
#include <GMCLIB_DecouplingPMSM.h>
```

9.298.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT
```

9.298.2 Description

Definition of GMCLIB_DECOUPLINGPMSM_DEFAULT as alias for GMCLIB_DECOUPLINGPMSM_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_DEFAULT as alias for GMCLIB_DECOUPLINGPMSM_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_DEFAULT as alias for GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.299 Define GMCLIB_DECOUPLINGPMSM_DEFAULT

```
#include <GMCLIB_DecouplingPMSM.h>
```

9.299.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT
```

9.299.2 Description

Definition of GMCLIB_DECOUPLINGPMSM_DEFAULT as alias for GMCLIB_DECOUPLINGPMSM_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_DEFAULT as alias for GMCLIB_DECOUPLINGPMSM_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_DEFAULT as alias for GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.300 Define GMCLIB_DECOUPLINGPMSM_DEFAULT_F32

```
#include <GMCLIB_DecouplingPMSM.h>
```

9.300.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT_F32 {(tFrac32)0,(tS16)0,(tFrac32)0,(tS16)0}
```

9.300.2 Description

Default value for GMCLIB_DECOUPLINGPMSM_T_F32.

9.301 Define GMCLIB_DECOUPLINGPMSM_DEFAULT_F16

```
#include <GMCLIB_DecouplingPMSM.h>
```

9.301.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT_F16 {(tFrac16)0,(tS16)0,(tFrac16)0,(tS16)0}
```

9.301.2 Description

Default value for GMCLIB_DECOUPLINGPMSM_T_F16.

9.302 Define GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT

```
#include <GMCLIB_DecouplingPMSM.h>
```

9.302.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT {(tFloat)0,(tFloat)0}
```

9.302.2 Description

Default value for GMCLIB_DECOUPLINGPMSM_T_FLT.

9.303 Define GMCLIB_ELIMDCBUSRIP_FLT_DNMAX

```
#include <GMCLIB_ElimDcBusRip.c>
```

9.303.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_FLT_DNMAX ((tFloat)1.1754942106924411e-38)
```

9.303.2 Description

Largest denormalized floating-point value.

9.304 Define GMCLIB_ElimDcBusRip

```
#include <GMCLIB_ElimDcBusRip.h>
```

9.304.1 Macro Definition

```
#define GMCLIB_ElimDcBusRip macro_dispatcher(GMCLIB_ElimDcBusRip, __VA_ARGS__)(__VA_ARGS__)
```

9.304.2 Description

This function implements the DC Bus voltage ripple elimination.

9.305 Define GMCLIB_ELIMDCBUSRIP_T

```
#include <GMCLIB_ElimDcBusRip.h>
```

9.305.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_T GMCLIB_ELIMDCBUSRIP_T_FLT
```

9.305.2 Description

Definition of GMCLIB_ELIMDCBUSRIP_T as alias for GMCLIB_ELIMDCBUSRIP_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_T as alias for GMCLIB_ELIMDCBUSRIP_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_T as alias for GMCLIB_ELIMDCBUSRIP_T_FLT datatype in case the single precision floating point implementation is selected.

9.306 Define GMCLIB_ELIMDCBUSRIP_T

```
#include <GMCLIB_ElimDcBusRip.h>
```

9.306.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_T GMCLIB_ELIMDCBUSRIP_T_FLT
```

9.306.2 Description

Definition of GMCLIB_ELIMDCBUSRIP_T as alias for GMCLIB_ELIMDCBUSRIP_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_T as alias for GMCLIB_ELIMDCBUSRIP_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_T as alias for GMCLIB_ELIMDCBUSRIP_T_FLT datatype in case the single precision floating point implementation is selected.

9.307 Define GMCLIB_ELIMDCBUSRIP_T

```
#include <GMCLIB_ElimDcBusRip.h>
```

9.307.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_T GMCLIB_ELIMDCBUSRIP_T_FLT
```

9.307.2 Description

Definition of GMCLIB_ELIMDCBUSRIP_T as alias for GMCLIB_ELIMDCBUSRIP_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_T as alias for GMCLIB_ELIMDCBUSRIP_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_T as alias for GMCLIB_ELIMDCBUSRIP_T_FLT datatype in case the single precision floating point implementation is selected.

9.308 Define GMCLIB_ELIMDCBUSRIP_DEFAULT

```
#include <GMCLIB_ElimDcBusRip.h>
```

9.308.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_DEFAULT GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT
```

9.308.2 Description

Definition of GMCLIB_ELIMDCBUSRIP_DEFAULT as alias for GMCLIB_ELIMDCBUSRIP_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_DEFAULT as alias for GMCLIB_ELIMDCBUSRIP_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_DEFAULT as alias for GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.309 Define GMCLIB_ELIMDCBUSRIP_DEFAULT

```
#include <GMCLIB_ElimDcBusRip.h>
```

9.309.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_DEFAULT GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT
```

9.309.2 Description

Definition of GMCLIB_ELIMDCBUSRIP_DEFAULT as alias for GMCLIB_ELIMDCBUSRIP_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_DEFAULT as alias for GMCLIB_ELIMDCBUSRIP_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_DEFAULT as alias for GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.310 Define GMCLIB_ELIMDCBUSRIP_DEFAULT

```
#include <GMCLIB_ElimDcBusRip.h>
```

9.310.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_DEFAULT GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT
```

9.310.2 Description

Definition of GMCLIB_ELIMDCBUSRIP_DEFAULT as alias for GMCLIB_ELIMDCBUSRIP_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_DEFAULT as alias for GMCLIB_ELIMDCBUSRIP_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_DEFAULT as alias for GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

9.311 Define GMCLIB_ELIMDCBUSRIP_DEFAULT_F32

```
#include <GMCLIB_ElimDcBusRip.h>
```

9.311.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_DEFAULT_F32 {(tFrac32)0,(tFrac32)0}
```

9.311.2 Description

Default value for GMCLIB_ELIMDCBUSRIP_T_F32.

9.312 Define GMCLIB_ELIMDCBUSRIP_DEFAULT_F16

```
#include <GMCLIB_ElimDcBusRip.h>
```

9.312.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_DEFAULT_F16 {(tFrac16)0,(tFrac16)0}
```

9.312.2 Description

Default value for GMCLIB_ELIMDCBUSRIP_T_F16.

9.313 Define GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT

```
#include <GMCLIB_ElimDcBusRip.h>
```

9.313.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT {(tFloat)0,(tFloat)0}
```

9.313.2 Description

Default value for GMCLIB_ELIMDCBUSRIP_T_FLT.

9.314 Define GMCLIB_Park

```
#include <GMCLIB_Park.h>
```

9.314.1 Macro Definition

```
#define GMCLIB_Park macro_dispatcher(GMCLIB_Park, __VA_ARGS__)(__VA_ARGS__)
```

9.314.2 Description

This function implements the calculates Park transformation.

9.315 Define GMCLIB_ParkInv

```
#include <GMCLIB_ParkInv.h>
```

9.315.1 Macro Definition

```
#define GMCLIB_ParkInv macro_dispatcher(GMCLIB_ParkInv, __VA_ARGS__)(__VA_ARGS__)
```

9.315.2 Description

This function implements the inverse Park transformation.

9.316 Define GMCLIB_SvmStd

```
#include <GMCLIB_SvmStd.h>
```

9.316.1 Macro Definition

```
#define GMCLIB_SvmStd macro_dispatcher(GMCLIB_SvmStd, __VA_ARGS__)(__VA_ARGS__)
```

9.316.2 Description

This function calculates the duty-cycle ratios using the Standard Space Vector Modulation technique.

9.317 Define MLIB_Abs

```
#include <MLIB_Abs.h>
```

9.317.1 Macro Definition

```
#define MLIB_Abs macro_dispatcher(MLIB_Abs, __VA_ARGS__)(__VA_ARGS__)
```

9.317.2 Description

This function returns absolute value of input parameter.

9.318 Define MLIB_AbsSat

```
#include <MLIB_AbsSat.h>
```

9.318.1 Macro Definition

```
#define MLIB_AbsSat macro_dispatcher(MLIB_AbsSat, __VA_ARGS__)(__VA_ARGS__)
```

9.318.2 Description

This function returns absolute value of input parameter and saturate if necessary.

9.319 Define MLIB_Add

```
#include <MLIB_Add.h>
```

9.319.1 Macro Definition

```
#define MLIB_Add macro_dispatcher(MLIB_Add, __VA_ARGS__)(__VA_ARGS__)
```

9.319.2 Description

This function returns sum of two input parameters.

9.320 Define MLIB_AddSat

```
#include <MLIB_AddSat.h>
```

9.320.1 Macro Definition

```
#define MLIB_AddSat macro_dispatcher(MLIB_AddSat, __VA_ARGS__)(__VA_ARGS__)
```

9.320.2 Description

This function returns sum of two input parameters and saturate if necessary.

9.321 Define MLIB_Convert

```
#include <MLIB_Convert.h>
```

9.321.1 Macro Definition

```
#define MLIB_Convert macro_dispatcher(MLIB_Convert, __VA_ARGS__)(__VA_ARGS__)
```

9.321.2 Description

This function converts the input value to different representation.

9.322 Define MLIB_ConvertPU

```
#include <MLIB_ConvertPU.h>
```

9.322.1 Macro Definition

```
#define MLIB_ConvertPU macro_dispatcher(MLIB_ConvertPU, __VA_ARGS__)(__VA_ARGS__)
```

9.322.2 Description

This function converts the input value to different representation with scale.

9.323 Define MLIB_Div

```
#include <MLIB_Div.h>
```

9.323.1 Macro Definition

```
#define MLIB_Div macro_dispatcher(MLIB_Div,__VA_ARGS__)(__VA_ARGS__)
```

9.323.2 Description

This function divides the first parameter by the second one.

9.324 Define MLIB_DivSat

```
#include <MLIB_DivSat.h>
```

9.324.1 Macro Definition

```
#define MLIB_DivSat macro_dispatcher(MLIB_DivSat, __VA_ARGS__)(__VA_ARGS__)
```

9.324.2 Description

This function divides the first parameter by the second one as saturate.

9.325 Define MLIB_Mac

```
#include <MLIB_Mac.h>
```

9.325.1 Macro Definition

```
#define MLIB_Mac macro_dispatcher(MLIB_Mac, __VA_ARGS__)(__VA_ARGS__)
```

9.325.2 Description

This function implements the multiply accumulate function.

9.326 Define MLIB_MacSat

```
#include <MLIB_MacSat.h>
```

9.326.1 Macro Definition

```
#define MLIB_MacSat macro_dispatcher(MLIB_MacSat, __VA_ARGS__)(__VA_ARGS__)
```

9.326.2 Description

This function implements the multiply accumulate function saturated if necessary.

9.327 Define MLIB_Mnac

```
#include <MLIB_Mnac.h>
```

9.327.1 Macro Definition

```
#define MLIB_Mnac macro_dispatcher(MLIB_Mnac, __VA_ARGS__)(__VA_ARGS__)
```

9.327.2 Description

This function implements the multiply-subtract function.

9.328 Define MLIB_Msu

```
#include <MLIB_Msu.h>
```

9.328.1 Macro Definition

```
#define MLIB_Msu macro_dispatcher(MLIB_Msu, __VA_ARGS__)(__VA_ARGS__)
```

9.328.2 Description

This function implements the multiply accumulate function.

9.329 Define MLIB_Mul

```
#include <MLIB_Mul.h>
```

9.329.1 Macro Definition

```
#define MLIB_Mul macro_dispatcher(MLIB_Mul, __VA_ARGS__)(__VA_ARGS__)
```

9.329.2 Description

This function multiply two input parameters.

9.330 Define MLIB_MulSat

```
#include <MLIB_MulSat.h>
```

9.330.1 Macro Definition

```
#define MLIB_MulSat macro_dispatcher(MLIB_MulSat, __VA_ARGS__)(__VA_ARGS__)
```

9.330.2 Description

This function multiply two input parameters and saturate if necessary.

9.331 Define MLIB_Neg

```
#include <MLIB_Neg.h>
```

9.331.1 Macro Definition

```
#define MLIB_Neg macro_dispatcher(MLIB_Neg, __VA_ARGS__)(__VA_ARGS__)
```

9.331.2 Description

This function returns negative value of input parameter.

9.332 Define MLIB_NegSat

```
#include <MLIB_NegSat.h>
```

9.332.1 Macro Definition

```
#define MLIB_NegSat macro_dispatcher(MLIB_NegSat, __VA_ARGS__)(__VA_ARGS__)
```

9.332.2 Description

This function returns negative value of input parameter and saturate if necessary.

9.333 Define MLIB_Norm

```
#include <MLIB_Norm.h>
```

9.333.1 Macro Definition

```
#define MLIB_Norm macro_dispatcher(MLIB_Norm, __VA_ARGS__)(__VA_ARGS__)
```

9.333.2 Description

This function returns the number of left shifts needed to normalize the input parameter.

9.334 Define MLIB_Round

```
#include <MLIB_Round.h>
```

9.334.1 Macro Definition

```
#define MLIB_Round macro_dispatcher(MLIB_Round, __VA_ARGS__)(__VA_ARGS__)
```

9.334.2 Description

This function rounds the first input value for number of digits defined by second parameter and saturate automatically.

9.335 Define MLIB_ShBi

```
#include <MLIB_ShBi.h>
```

9.335.1 Macro Definition

```
#define MLIB_ShBi macro_dispatcher(MLIB_ShBi, __VA_ARGS__)(__VA_ARGS__)
```

9.335.2 Description

Based on sign of second parameter this function shifts the first parameter to right or left. If the sign of second parameter is negative, shift to right.

9.336 Define MLIB_ShBiSat

```
#include <MLIB_ShBiSat.h>
```

9.336.1 Macro Definition

```
#define MLIB_ShBiSat macro_dispatcher(MLIB_ShBiSat, __VA_ARGS__)(__VA_ARGS__)
```

9.336.2 Description

Based on sign of second parameter this function shifts the first parameter to right or left and saturate if necessary. If the sign of second parameter is negative, shift to right.

9.337 Define MLIB_ShL

```
#include <MLIB_ShL.h>
```

9.337.1 Macro Definition

```
#define MLIB_ShL macro_dispatcher(MLIB_ShL, __VA_ARGS__)(__VA_ARGS__)
```

9.337.2 Description

This function shifts the first parameter to left by number defined by second parameter.

9.338 Define MLIB_ShLSat

```
#include <MLIB_ShLSat.h>
```

9.338.1 Macro Definition

```
#define MLIB_ShLSat macro_dispatcher(MLIB_ShLSat, __VA_ARGS__)(__VA_ARGS__)
```

9.338.2 Description

This function shifts the first parameter to left by number defined by second parameter and saturate if necessary.

9.339 Define MLIB_ShR

```
#include <MLIB_ShR.h>
```

9.339.1 Macro Definition

```
#define MLIB_ShR macro_dispatcher(MLIB_ShR, __VA_ARGS__)(__VA_ARGS__)
```

9.339.2 Description

This function shifts the first parameter to right by number defined by second parameter.

9.340 Define MLIB_Sub

```
#include <MLIB_Sub.h>
```

9.340.1 Macro Definition

```
#define MLIB_Sub macro_dispatcher(MLIB_Sub, __VA_ARGS__)(__VA_ARGS__)
```

9.340.2 Description

This function subtracts the second parameter from the first one.

9.341 Define MLIB_SubSat

```
#include <MLIB_SubSat.h>
```

9.341.1 Macro Definition

```
#define MLIB_SubSat macro_dispatcher(MLIB_SubSat, __VA_ARGS__)(__VA_ARGS__)
```

9.341.2 Description

This function subtracts the second parameter from the first one and saturate if necessary.

9.342 Define MLIB_VMac

```
#include <MLIB_VMac.h>
```

9.342.1 Macro Definition

```
#define MLIB_VMac macro_dispatcher(MLIB_VMac, __VA_ARGS__)(__VA_ARGS__)
```

9.342.2 Description

This function implements the vector multiply accumulate function.

9.343 Define SWLIBS_VERSION

```
#include <SWLIBS_Config.h>
```

9.343.1 Macro Definition

```
#define SWLIBS_VERSION {(unsigned char)1U,(unsigned char)1U,(unsigned char)9U}
```

9.343.2 Description

9.344 Define SWLIBS_STD_ON

```
#include <SWLIBS_Config.h>
```

9.344.1 Macro Definition

```
#define SWLIBS_STD_ON 0x01U
```

9.344.2 Description

9.345 Define SWLIBS_STD_OFF

```
#include <SWLIBS_Config.h>
```

9.345.1 Macro Definition

```
#define SWLIBS_STD_OFF 0x00U
```

9.345.2 Description

9.346 Define F32

```
#include <SWLIBS_Config.h>
```

9.346.1 Macro Definition

```
#define F32 F32
```

9.346.2 Description

9.347 Define F16

```
#include <SWLIBS_Config.h>
```

9.347.1 Macro Definition

```
#define F16 F16
```

9.347.2 Description

9.348 Define FLT

```
#include <SWLIBS_Config.h>
```

9.348.1 Macro Definition

```
#define FLT FLT
```

9.348.2 Description

9.349 Define SWLIBS_DEFAULT_IMPLEMENTATION_F32

```
#include <SWLIBS_Config.h>
```

9.349.1 Macro Definition

```
#define SWLIBS_DEFAULT_IMPLEMENTATION_F32 (1U)
```

9.349.2 Description

9.350 Define SWLIBS_DEFAULT_IMPLEMENTATION_F16

```
#include <SWLIBS_Config.h>
```

9.350.1 Macro Definition

```
#define SWLIBS_DEFAULT_IMPLEMENTATION_F16 (2U)
```

9.350.2 Description

9.351 Define SWLIBS_DEFAULT_IMPLEMENTATION_FLT

```
#include <SWLIBS_Config.h>
```

9.351.1 Macro Definition

```
#define SWLIBS_DEFAULT_IMPLEMENTATION_FLT (3U)
```

9.351.2 Description

9.352 Define SWLIBS_SUPPORT_F32

```
#include <SWLIBS_Config.h>
```

9.352.1 Macro Definition

```
#define SWLIBS_SUPPORT_F32 SWLIBS_STD_ON
```

9.352.2 Description

Enables/disables support of 32-bit fractional implementation.

9.353 Define SWLIBS_SUPPORT_F16

```
#include <SWLIBS_Config.h>
```

9.353.1 Macro Definition

```
#define SWLIBS_SUPPORT_F16 SWLIBS_STD_ON
```

9.353.2 Description

Enables/disables support of 16-bit fractional implementation.

9.354 Define SWLIBS_SUPPORT_FLT

```
#include <SWLIBS_Config.h>
```

9.354.1 Macro Definition

```
#define SWLIBS_SUPPORT_FLT SWLIBS_STD_ON
```

9.354.2 Description

Enables/disables support of single precision floating point implementation.

9.355 Define SWLIBS_SUPPORTED_IMPLEMENTATION

```
#include <SWLIBS_Config.h>
```

9.355.1 Macro Definition

```
#define SWLIBS_SUPPORTED_IMPLEMENTATION {SWLIBS_SUPPORT_F32,\ SWLIBS_SUPPORT_F16,\
SWLIBS_SUPPORT_FLT,\ 0,0,0,0,0,0}
```

9.355.2 Description

Array of supported implementations.

9.356 Define SWLIBS_DEFAULT_IMPLEMENTATION

```
#include <SWLIBS_Config.h>
```

9.356.1 Macro Definition

```
#define SWLIBS_DEFAULT_IMPLEMENTATION
```

9.356.2 Description

Selection of default implementation.

9.357 Define SFRACT_MIN

```
#include <SWLIBS_Defines.h>
```

9.357.1 Macro Definition

```
#define SFRACT_MIN (-1.0)
```

9.357.2 Description

Constant representing the maximal negative value of a signed 16-bit fixed point fractional number, floating point representation.

9.358 Define SFRACT_MAX

```
#include <SWLIBS_Defines.h>
```

9.358.1 Macro Definition

```
#define SFRACT_MAX (0.999969482421875)
```

9.358.2 Description

Constant representing the maximal positive value of a signed 16-bit fixed point fractional number, floating point representation.

9.359 Define FRACT_MIN

```
#include <SWLIBS_Defines.h>
```

9.359.1 Macro Definition

```
#define FRACT_MIN (-1.0)
```

9.359.2 Description

Constant representing the maximal negative value of signed 32-bit fixed point fractional number, floating point representation.

9.360 Define FRACT_MAX

```
#include <SWLIBS_Defines.h>
```

9.360.1 Macro Definition

```
#define FRACT_MAX (0.999999999534338712692260742187 5)
```

9.360.2 Description

Constant representing the maximal positive value of a signed 32-bit fixed point fractional number, floating point representation.

9.361 Define FRAC32_0_5

```
#include <SWLIBS_Defines.h>
```

9.361.1 Macro Definition

```
#define FRAC32_0_5 ((tFrac32) 0x40000000)
```

9.361.2 Description

Value 0.5 in 32-bit fixed point fractional format.

9.362 Define FRAC16_0_5

```
#include <SWLIBS_Defines.h>
```

9.362.1 Macro Definition

```
#define FRAC16_0_5 ((tFrac16) 0x4000)
```

9.362.2 Description

Value 0.5 in 16-bit fixed point fractional format.

9.363 Define FRAC32_0_25

```
#include <SWLIBS_Defines.h>
```

9.363.1 Macro Definition

```
#define FRAC32_0_25 ((tFrac32) 0x20000000)
```

9.363.2 Description

Value 0.25 in 32-bit fixed point fractional format.

9.364 Define FRAC16_0_25

```
#include <SWLIBS_Defines.h>
```

9.364.1 Macro Definition

```
#define FRAC16_0_25 ((tFrac16) 0x2000)
```

9.364.2 Description

Value 0.25 in 16-bit fixed point fractional format.

9.365 Define UINT16_MAX

```
#include <SWLIBS_Defines.h>
```

9.365.1 Macro Definition

```
#define UINT16_MAX ((tU16) 0x8000)
```

9.365.2 Description

Constant representing the maximal positive value of a unsigned 16-bit fixed point integer number, equal to 2^{15} = 0x8000.

9.366 Define INT16_MAX

```
#include <SWLIBS_Defines.h>
```

9.366.1 Macro Definition

```
#define INT16_MAX ((tS16) 0x7fff)
```

9.366.2 Description

Constant representing the maximal positive value of a signed 16-bit fixed point integer number, equal to 2^{15}-1 = 0x7fff.

9.367 Define INT16_MIN

```
#include <SWLIBS_Defines.h>
```

9.367.1 Macro Definition

```
#define INT16_MIN ((tS16) 0x8000)
```

9.367.2 Description

Constant representing the maximal negative value of a signed 16-bit fixed point integer number, equal to -2^{15} = 0x8000.

9.368 Define UINT32_MAX

```
#include <SWLIBS_Defines.h>
```

9.368.1 Macro Definition

```
#define UINT32_MAX ((tU32) 0x80000000U)
```

9.368.2 Description

Constant representing the maximal positive value of a unsigned 32-bit fixed point integer number, equal to 2^{31} = 0x80000000.

9.369 Define INT32_MAX

```
#include <SWLIBS_Defines.h>
```

9.369.1 Macro Definition

```
#define INT32_MAX ((tS32) 0x7fffffff)
```

9.369.2 Description

Constant representing the maximal positive value of a signed 32-bit fixed point integer number, equal to 2^{31}-1 = 0x7fff ffff.

9.370 Define INT32_MIN

```
#include <SWLIBS_Defines.h>
```

9.370.1 Macro Definition

```
#define INT32_MIN ((tS32) 0x80000000U)
```

9.370.2 Description

Constant representing the maximal negative value of a signed 32-bit fixed point integer number, equal to -2^{31} = 0x8000 0000.

9.371 Define FLOAT_MIN

```
#include <SWLIBS_Defines.h>
```

9.371.1 Macro Definition

```
#define FLOAT_MIN ((tFloat)(-3.4028234e+38F))
```

9.371.2 Description

Constant representing the maximal negative value of the 32-bit float type.

9.372 Define FLOAT_MAX

```
#include <SWLIBS_Defines.h>
```

9.372.1 Macro Definition

```
#define FLOAT_MAX ((tFloat)(3.4028234e+38F))
```

9.372.2 Description

Constant representing the maximal positive value of the 32-bit float type.

9.373 Define INT16TOINT32

```
#include <SWLIBS_Defines.h>
```

9.373.1 Macro Definition

```
#define INT16TOINT32 ((tS32) (x))
```

9.373.2 Description

Type casting - signed 16-bit integer value cast to a signed 32-bit integer.

9.374 Define INT32TOINT16

```
#include <SWLIBS_Defines.h>
```

9.374.1 Macro Definition

```
#define INT32TOINT16 ((tS16) (x))
```

9.374.2 Description

Type casting - signed 32-bit integer value cast to a signed 16-bit integer.

9.375 Define INT32TOINT64

```
#include <SWLIBS_Defines.h>
```

9.375.1 Macro Definition

```
#define INT32TOINT64 ((tS64) (x))
```

9.375.2 Description

Type casting - signed 32-bit integer value cast to a signed 64-bit integer.

9.376 Define INT64TOINT32

```
#include <SWLIBS_Defines.h>
```

9.376.1 Macro Definition

```
#define INT64TOINT32 ((tS32) (x))
```

9.376.2 Description

Type casting - signed 64-bit integer value cast to a signed 32-bit integer.

9.377 Define F16TOINT16

```
#include <SWLIBS_Defines.h>
```

9.377.1 Macro Definition

```
#define F16TOINT16 ((tS16) (x))
```

9.377.2 Description

Type casting - signed 16-bit fractional value cast to a signed 16-bit integer.

9.378 Define F32TOINT16

```
#include <SWLIBS_Defines.h>
```

9.378.1 Macro Definition

```
#define F32TOINT16 ((tS16) (x))
```

9.378.2 Description

Type casting - lower 16 bits of a signed 32-bit fractional value cast to a signed 16-bit integer.

9.379 Define F64TOINT16

```
#include <SWLIBS_Defines.h>
```

9.379.1 Macro Definition

```
#define F64TOINT16 ((tS16) (x))
```

9.379.2 Description

Type casting - lower 16 bits of a signed 64-bit fractional value cast to a signed 16-bit integer.

9.380 Define F16TOINT32

```
#include <SWLIBS_Defines.h>
```

9.380.1 Macro Definition

```
#define F16TOINT32 ((tS32) (x))
```

9.380.2 Description

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.

9.381 Define F32TOINT32

```
#include <SWLIBS_Defines.h>
```

9.381.1 Macro Definition

```
#define F32TOINT32 ((tS32) (x))
```

9.381.2 Description

Type casting - signed 32-bit fractional value cast to a signed 32-bit integer.

9.382 Define F64TOINT32

```
#include <SWLIBS_Defines.h>
```

9.382.1 Macro Definition

```
#define F64TOINT32 ((tS32) (x))
```

9.382.2 Description

Type casting - lower 32 bits of a signed 64-bit fractional value cast to a signed 32-bit integer.

9.383 Define F16TOINT64

```
#include <SWLIBS_Defines.h>
```

9.383.1 Macro Definition

```
#define F16TOINT64 ((tS64) (x))
```

9.383.2 Description

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.

9.384 Define F32TOINT64

```
#include <SWLIBS_Defines.h>
```

9.384.1 Macro Definition

```
#define F32TOINT64 ((tS64) (x))
```

9.384.2 Description

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.

9.385 Define F64TOINT64

```
#include <SWLIBS_Defines.h>
```

9.385.1 Macro Definition

```
#define F64TOINT64 ((tS64) (x))
```

9.385.2 Description

Type casting - signed 64-bit fractional value cast to a signed 64-bit integer.

9.386 Define INT16TOF16

```
#include <SWLIBS_Defines.h>
```

9.386.1 Macro Definition

```
#define INT16TOF16 ((tFrac16) (x))
```

9.386.2 Description

Type casting - signed 16-bit integer value cast to a signed 16-bit fractional.

9.387 Define INT16TOF32

```
#include <SWLIBS_Defines.h>
```

9.387.1 Macro Definition

```
#define INT16TOF32 ((tFrac32) (x))
```

9.387.2 Description

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.

9.388 Define INT32TOF16

```
#include <SWLIBS_Defines.h>
```

9.388.1 Macro Definition

```
#define INT32TOF16 ((tFrac16) (x))
```

9.388.2 Description

Type casting - lower 16-bits of a signed 32-bit integer value cast to a signed 16-bit fractional.

9.389 Define INT32TOF32

```
#include <SWLIBS_Defines.h>
```

9.389.1 Macro Definition

```
#define INT32TOF32 ((tFrac32) (x))
```

9.389.2 Description

Type casting - signed 32-bit integer value cast to a signed 32-bit fractional.

9.390 Define INT64TOF16

```
#include <SWLIBS_Defines.h>
```

9.390.1 Macro Definition

```
#define INT64TOF16 ((tFrac16) (x))
```

9.390.2 Description

Type casting - lower 16-bits of a signed 64-bit integer value cast to a signed 16-bit fractional.

9.391 Define INT64TOF32

```
#include <SWLIBS_Defines.h>
```

9.391.1 Macro Definition

```
#define INT64TOF32 ((tFrac32) (x))
```

9.391.2 Description

Type casting - lower 32-bits of a signed 64-bit integer value cast to a signed 32-bit fractional.

9.392 Define F16_1_DIVBY_SQRT3

```
#include <SWLIBS_Defines.h>
```

9.392.1 Macro Definition

```
#define F16_1_DIVBY_SQRT3 ((tFrac16) 0x49E7)
```

9.392.2 Description

One over sqrt(3) with a 16-bit result, the result rounded for a better precision, i.e. round(1/ $\sqrt{(3)}$*2 ^{15}).

9.393 Define F32_1_DIVBY_SQRT3

```
#include <SWLIBS_Defines.h>
```

9.393.1 Macro Definition

```
#define F32_1_DIVBY_SQRT3 ((tFrac32) 0x49E69D16)
```

9.393.2 Description

One over sqrt(3) with a 32-bit result, the result rounded for a better precision, i.e. round(1/ $\sqrt{(3)}*2^{31}$).

9.394 Define F16_SQRT3_DIVBY_2

```
#include <SWLIBS_Defines.h>
```

9.394.1 Macro Definition

```
#define F16_SQRT3_DIVBY_2 ((tFrac16) 0x6EDA)
```

9.394.2 Description

Sqrt(3) divided by two with a 16-bit result, the result rounded for a better precision, i.e. round($\sqrt{(3)}/2*2^{15}$).

9.395 Define F32_SQRT3_DIVBY_2

```
#include <SWLIBS_Defines.h>
```

9.395.1 Macro Definition

```
#define F32_SQRT3_DIVBY_2 ((tFrac32) 0x6ED9EBA1)
```

9.395.2 Description

Sqrt(3) divided by two with a 32-bit result, the result rounded for a better precision, i.e. round($\sqrt{(3)}/2*2^{31}$).

9.396 Define F16_SQRT2_DIVBY_2

```
#include <SWLIBS_Defines.h>
```

9.396.1 Macro Definition

```
#define F16_SQRT2_DIVBY_2 ((tFrac16) 0x5A82)
```

9.396.2 Description

Sqrt(2) divided by two with a 16-bit result, the result rounded for a better precision, i.e. round($\sqrt{(2)}/2*2^{15}$).

9.397 Define F32_SQRT2_DIVBY_2

```
#include <SWLIBS_Defines.h>
```

9.397.1 Macro Definition

```
#define F32_SQRT2_DIVBY_2 ((tFrac32) 0x5A82799A)
```

9.397.2 Description

Sqrt(2) divided by two with a 32-bit result, the result rounded for a better precision, i.e. round($\sqrt{(2)}/2*2^{31}$).

9.398 Define FRAC16

```
#include <SWLIBS_Defines.h>
```

9.398.1 Macro Definition

```
#define FRAC16 ((tFrac16) (((x) < SFRACT_MAX) ? (((x) >= SFRACT_MIN) ? ((x)*32768.0) :
INT16_MIN) : INT16_MAX))
```

9.398.2 Description

Macro converting a signed fractional [-1,1) number into a 16-bit fixed point number in format Q1.15.

9.399 Define FRAC32

```
#include <SWLIBS_Defines.h>
```

9.399.1 Macro Definition

```
#define FRAC32 ((tFrac32) (((x) < FRACT_MAX) ? (((x) >= FRACT_MIN) ? ((x)*2147483648.0) :
INT32_MIN) : INT32_MAX))
```

9.399.2 Description

Macro converting a signed fractional [-1,1) number into a 32-bit fixed point number in format Q1.31.

9.400 Define FLOAT_DIVBY_SQRT3

```
#include <SWLIBS_Defines.h>
```

9.400.1 Macro Definition

```
#define FLOAT_DIVBY_SQRT3 ((tFloat) 0.5773502691896258)
```

9.400.2 Description

One over sqrt(3) in single precision floating point format.

9.401 Define FLOAT_SQRT3_DIVBY_2

```
#include <SWLIBS_Defines.h>
```

9.401.1 Macro Definition

```
#define FLOAT_SQRT3_DIVBY_2 ((tFloat) 0.866025403784439)
```

9.401.2 Description

Sqrt(3) divided by two in single precision floating point format.

9.402 Define FLOAT_SQRT3_DIVBY_4

```
#include <SWLIBS_Defines.h>
```

9.402.1 Macro Definition

```
#define FLOAT_SQRT3_DIVBY_4 ((tFloat) 0.4330127018922190)
```

9.402.2 Description

Sqrt(3) divided by four in single precision floating point format.

9.403 Define FLOAT_SQRT3_DIVBY_4_CORRECTION

```
#include <SWLIBS_Defines.h>
```

9.403.1 Macro Definition

```
#define FLOAT_SQRT3_DIVBY_4_CORRECTION ((tFloat)0)
```

9.403.2 Description

Sqrt(3) divided by four correction constant.

9.404 Define FLOAT_2_PI

```
#include <SWLIBS_Defines.h>
```

9.404.1 Macro Definition

```
#define FLOAT_2_PI ((tFloat) 6.28318530717958)
```

9.404.2 Description

2* π in single precision floating point format.

9.405 Define FLOAT_PI

```
#include <SWLIBS_Defines.h>
```

9.405.1 Macro Definition

```
#define FLOAT_PI ((tFloat) 3.14159265358979)
```

9.405.2 Description

π in single precision floating point format.

9.406 Define FLOAT_PI_DIVBY_2

```
#include <SWLIBS_Defines.h>
```

9.406.1 Macro Definition

```
#define FLOAT_PI_DIVBY_2 ((tFloat) 1.57079632679490)
```

9.406.2 Description

π/2 in single precision floating point format.

9.407 Define FLOAT_TAN_PI_DIVBY_6

```
#include <SWLIBS_Defines.h>
```

9.407.1 Macro Definition

```
#define FLOAT_TAN_PI_DIVBY_6 ((tFloat)0.577350269189626000)
```

9.407.2 Description

Tan(π/6) in single precision floating point format.

9.408 Define FLOAT_TAN_PI_DIVBY_12

```
#include <SWLIBS_Defines.h>
```

9.408.1 Macro Definition

```
#define FLOAT_TAN_PI_DIVBY_12 ((tFloat)0.267949192431123000)
```

9.408.2 Description

Tan(π/12) in single precision floating point format.

9.409 Define FLOAT_PI_DIVBY_6

```
#include <SWLIBS_Defines.h>
```

9.409.1 Macro Definition

```
#define FLOAT_PI_DIVBY_6 ((tFloat)0.523598775598299000)
```

9.409.2 Description

π/6 in single precision floating point format.

9.410 Define FLOAT_PI_SINGLE_CORRECTION

```
#include <SWLIBS_Defines.h>
```

9.410.1 Macro Definition

```
#define FLOAT_PI_SINGLE_CORRECTION ((tFloat)4.37102068E-8)
```

9.410.2 Description

Double to single precision correction constant for π, equal to (π(Double) - π(Single)).

9.411 Define FLOAT_PI_CORRECTION

```
#include <SWLIBS_Defines.h>
```

9.411.1 Macro Definition

```
#define FLOAT_PI_CORRECTION ((tFloat)8.74204136E-8)
```

9.411.2 Description

Double to single precision correction constant for π, equal to (2*(π(Double) - π(Single)).

9.412 Define FLOAT_PI_DIVBY_4

```
#include <SWLIBS_Defines.h>
```

9.412.1 Macro Definition

```
#define FLOAT_PI_DIVBY_4 ((tFloat) 0.7853981633974480)
```

9.412.2 Description

π/4 in single precision floating point format.

9.413 Define FLOAT_4_DIVBY_PI

```
#include <SWLIBS_Defines.h>
```

9.413.1 Macro Definition

```
#define FLOAT_4_DIVBY_PI ((tFloat) 1.2732395447351600)
```

9.413.2 Description

Number four divided by π in single precision floating point format.

9.414 Define FLOAT_0_5

```
#include <SWLIBS_Defines.h>
```

9.414.1 Macro Definition

```
#define FLOAT_0_5 ((tFloat) 0.5)
```

9.414.2 Description

Value 0.5 in single precision floating point format.

9.415 Define FLOAT_MINUS_0_5

```
#include <SWLIBS_Defines.h>
```

9.415.1 Macro Definition

```
#define FLOAT_MINUS_0_5 ((tFloat) -0.5)
```

9.415.2 Description

Value -0.5 in single precision floating point format.

9.416 Define FLOAT_PLUS_1

```
#include <SWLIBS_Defines.h>
```

9.416.1 Macro Definition

```
#define FLOAT_PLUS_1 ((tFloat) 1)
```

9.416.2 Description

Value 1 in single precision floating point format.

9.417 Define FLOAT_MINUS_1

```
#include <SWLIBS_Defines.h>
```

9.417.1 Macro Definition

```
#define FLOAT_MINUS_1 ((tFloat) -1)
```

9.417.2 Description

Value -1 in single precision floating point format.

9.418 Define FLOAT_MIN_NORM

```
#include <SWLIBS_Defines.h>
```

9.418.1 Macro Definition

```
#define FLOAT_MIN_NORM ((tFloat) 1.175494350822288e-38)
```

9.418.2 Description

Constant representing the smallest positive normalized 32-bit floating-point value.

9.419 Define NULL

```
#include <SWLIBS_Typedefs.h>
```

9.419.1 Macro Definition

```
#define NULL 0
```

9.419.2 Description

9.420 Define FALSE

```
#include <SWLIBS_Typedefs.h>
```

9.420.1 Macro Definition

```
#define FALSE ((tBool)0)
```

9.420.2 Description

Boolean type FALSE constant

9.421 Define TRUE

```
#include <SWLIBS_Typedefs.h>
```

9.421.1 Macro Definition

```
#define TRUE ((tBool)1)
```

9.421.2 Description

Boolean type TRUE constant

9.422 Define SWLIBS_2Syst

```
#include <SWLIBS_Typedefs.h>
```

9.422.1 Macro Definition

```
#define SWLIBS_2Syst SWLIBS_2Syst_FLT
```

9.422.2 Description

Definition of SWLIBS_2Syst as alias for SWLIBS_2Syst_F32 array in case the 32-bit fractional implementation is selected.

Definition of SWLIBS_2Syst as alias for SWLIBS_2Syst_F16 array in case the 16-bit fractional implementation is selected.

Definition of SWLIBS_2Syst as alias for SWLIBS_2Syst_FLT array in case the single precision floating point implementation is selected.

9.423 Define SWLIBS_2Syst

```
#include <SWLIBS_Typedefs.h>
```

9.423.1 Macro Definition

```
#define SWLIBS_2Syst SWLIBS_2Syst_FLT
```

9.423.2 Description

Definition of SWLIBS_2Syst as alias for SWLIBS_2Syst_F32 array in case the 32-bit fractional implementation is selected.

Definition of SWLIBS_2Syst as alias for SWLIBS_2Syst_F16 array in case the 16-bit fractional implementation is selected.

Definition of SWLIBS_2Syst as alias for SWLIBS_2Syst_FLT array in case the single precision floating point implementation is selected.

9.424 Define SWLIBS_2Syst

```
#include <SWLIBS_Typedefs.h>
```

9.424.1 Macro Definition

```
#define SWLIBS_2Syst SWLIBS_2Syst_FLT
```

9.424.2 Description

Definition of SWLIBS_2Syst as alias for SWLIBS_2Syst_F32 array in case the 32-bit fractional implementation is selected.

Definition of SWLIBS_2Syst as alias for SWLIBS_2Syst_F16 array in case the 16-bit fractional implementation is selected.

Definition of SWLIBS_2Syst as alias for SWLIBS_2Syst_FLT array in case the single precision floating point implementation is selected.

9.425 Define SWLIBS_3Syst

```
#include <SWLIBS_Typedefs.h>
```

9.425.1 Macro Definition

```
#define SWLIBS_3Syst SWLIBS_3Syst_FLT
```

9.425.2 Description

Definition of SWLIBS_3Syst as alias for SWLIBS_3Syst_F32 array in case the 32-bit fractional implementation is selected.

Definition of SWLIBS_3Syst as alias for SWLIBS_3Syst_F16 array in case the 16-bit fractional implementation is selected.

Definition of SWLIBS_3Syst as alias for SWLIBS_3Syst_FLT array in case the single precision floating point implementation is selected.

9.426 Define SWLIBS_3Syst

```
#include <SWLIBS_Typedefs.h>
```

9.426.1 Macro Definition

```
#define SWLIBS_3Syst SWLIBS_3Syst_FLT
```

9.426.2 Description

Definition of SWLIBS_3Syst as alias for SWLIBS_3Syst_F32 array in case the 32-bit fractional implementation is selected.

Definition of SWLIBS_3Syst as alias for SWLIBS_3Syst_F16 array in case the 16-bit fractional implementation is selected.

Definition of SWLIBS_3Syst as alias for SWLIBS_3Syst_FLT array in case the single precision floating point implementation is selected.

9.427 Define SWLIBS_3Syst

```
#include <SWLIBS_Typedefs.h>
```

9.427.1 Macro Definition

```
#define SWLIBS_3Syst SWLIBS_3Syst_FLT
```

9.427.2 Description

Definition of SWLIBS_3Syst as alias for SWLIBS_3Syst_F32 array in case the 32-bit fractional implementation is selected.

Definition of SWLIBS_3Syst as alias for SWLIBS_3Syst_F16 array in case the 16-bit fractional implementation is selected.

Definition of SWLIBS_3Syst as alias for SWLIBS_3Syst_FLT array in case the single precision floating point implementation is selected.

9.428 Define SWLIBS_VERSION_DEFAULT

```
#include <SWLIBS_Version.c>
```

9.428.1 Macro Definition

```
#define SWLIBS_VERSION_DEFAULT {SWLIBS_ID, SWLIBS_VERSION, SWLIBS_SUPPORTED_IMPLEMENTATION }
```

9.428.2 Description

9.429 Define SWLIBS_MCID_SIZE

```
#include <SWLIBS_Version.h>
```

9.429.1 Macro Definition

```
#define SWLIBS_MCID_SIZE ((unsigned char)4U)
```

9.429.2 Description

9.430 Define SWLIBS_MCVERSION_SIZE

```
#include <SWLIBS_Version.h>
```

9.430.1 Macro Definition

```
#define SWLIBS_MCVERSION_SIZE ((unsigned char)3U)
```

9.430.2 Description

9.431 Define SWLIBS_MCIMPLEMENTATION_SIZE

```
#include <SWLIBS_Version.h>
```

9.431.1 Macro Definition

```
#define SWLIBS_MCIMPLEMENTATION_SIZE ((unsigned char)9U)
```

9.431.2 Description

9.432 Define SWLIBS_ID

```
#include <SWLIBS_Version.h>
```

9.432.1 Macro Definition

```
#define SWLIBS_ID {(unsigned char)0x90U,(unsigned char)0x71U,(unsigned char)0x77U,(unsigned
char)0x68U}
```

9.432.2 Description

Library identification string.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number MPC574XPMCLUG
Revision 12