General Functions Library

User Reference Manual

56800E
Digital Signal Controller

56800E_GFLIB
Rev. 3
06/2011

freescale.com

The following revision history table summarizes changes contained in this document.

Table 0-1. Revision History

Date	Revision Label	Description
	0	Initial release
	1	Reformatted and updated revision
	2	FSLESL 2.0
	3	Fixed inconsistence in the API Summary table

Chapter 1 License Agreement

FREESCALE SEMICONDUCTOR SOFTWARE LICENSE AGREEMENT. This is a legal agreement between you (either as an individual or as an authorized representative of your employer) and Freescale Semiconductor, Inc. ("Freescale"). It concerns your rights to use this file and any accompanying written materials (the "Software"). In consideration for Freescale 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.

OBJECT PROVIDED, OBJECT REDISTRIBUTION LICENSE GRANT. Freescale grants to you, free of charge, the non-exclusive, non-transferable right (1) to reproduce the Software, (2) to distribute the Software, and (3) to sublicense to others the right to use the distributed Software. The Software is provided to you only in object (machine-readable) form. You may exercise the rights above only with respect to such object form. You may not translate, reverse engineer, decompile, or disassemble the Software except to the extent applicable law specifically prohibits such restriction. In addition, you must prohibit your sublicensees from doing the same. If you violate any of the terms or restrictions of this Agreement, Freescale 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. Freescale 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, Freescale does not grant to you any express or implied rights under any Freescale 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. Freescale is NOT obligated to provide any support, upgrades or new releases of the Software. If you wish, you may contact Freescale and report problems and provide suggestions regarding the Software. Freescale has no obligation whatsoever to respond in any way to such a problem report or suggestion. Freescale 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, FREESCALE 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 FREESCALE 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 Freescale 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 FREESCALE 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; EXPORT RESTRICTIONS. You must use the Software in accordance with all applicable U.S. laws, regulations and statutes. You agree that neither you nor your licensees (if any) intend to or will, directly or indirectly, export or transmit the Software to any country in violation of U.S. export restrictions.

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 Freescale Semiconductor, Inc., 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 Freescale 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 Freescale trademarks, brand names, or logos.

ENTIRE AGREEMENT. This Agreement constitutes the entire agreement between you and Freescale 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 Freescale.

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 Freescale 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 Freescale 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.

Chapter 2 INTRODUCTION

2.1 Overview

This reference manual describes General Functions Library for Freescale 56800E family of Digital Signal Controllers. This library contains optimized functions for 56800E family of controllers. The library is supplied in a binary form, which is unique by its simplicity to integrate with user application.

2.2 Supported Compilers

The General Functions Library (GFLIB) is written in assembly language with a C-callable interface. The library was built and tested using the following compiler:

- CodeWarrior™ Development Studio for Freescale™ DSC56800/E Digital Signal Controllers, version 8.3

The library is delivered in the *56800E_GFLIB.lib* library module. The interfaces to the algorithms included in this library have been combined into a single public interface include file, the *gflib.h*. This was done to reduce the number of files required for inclusion by the application programs. Refer to the specific algorithm sections of this document for details on the software application programming interface (API), defined and functionality provided for the individual algorithms.

2.3 Installation

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

Each General Functions Library release is installed in its own new folder, named *56800E_GFLIB_rX.X*, where *X.X* denotes the actual release number. This way of library installation allows the users to maintain older releases and projects and gives them a free choice to select the active library release.

To start the installation process, follow the following steps:

1. Execute the *56800E_FSLESL_rXX.exe* file.
2. Follow the FSLESL software installation instructions on your screen.

2.4 Library Integration

General Functions Library is added into a new CodeWarrior project by taking the following steps:

1. Create a new empty project.
2. Create *GFLIB* group in your new open project. Note that this step is not mandatory, it is mentioned here just for the purpose of maintaining file consistency in the CodeWarrior project window. In the CodeWarrior menu, choose Project > Create Group..., type GFLIB into the dialog window that pops up, and click <OK>.
3. Refer the *56800E_GFLIB.lib* file in the project window. This can be achieved by dragging the library file from the proper library subfolder and dropping it into the *GFLIB* group in the CodeWarrior project window. This step will automatically add the *GFLIB* path into the project access paths, such as the user can take advantage of the library functions to achieve flawless project compilation and linking.
4. It is similar with the reference file *gflib.h*. This file can be dragged from the proper library subfolder and dropped into the GFLIB group in the CodeWarrior project window.
5. The following program line must be added into the user-application source code in order to use the library functions.

```
#include "gflib.h"
```

2.5 API Definition

The description of each function described in this General Functions Library user reference manual consists of a number of subsections:

Synopsis

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

Prototype

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

Arguments

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

Description

This subsection is a description of the function or macro. It explains algorithms being used by functions or macros.

Return

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

Range Issues

This optional subsection specifies the ranges of input variables.

Special Issues

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

Implementation

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

See Also

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

Performance

This section specifies the actual requirements of the function or macro in terms of required code memory, data memory, and number of clock cycles to execute. If the clock cycles have two numbers for instance 21/22, then the number 21 is measured on the MCF56F80xx core and the number 22 is measured on the MCF56F83xx core.

2.6 Data Types

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

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

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

2.6.1 Signed Integer (SI)

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

$$-2^{[N-1]} \le SI \le [2^{[N-1]} - 1]$$ **Eqn. 2-1**

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

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

2.6.2 Unsigned Integer (UI)

The unsigned integer numbers are positive only, and they have nearly twice the magnitude of a signed number of the same size. The unsigned integer numbers lie in the following range:

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

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

2.6.3 Signed Fractional (SF)

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

$$-1{,}0 \le SF \le 1{,}0 - 2^{-[N-1]}$$ **Eqn. 2-3**

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

2.6.4 Unsigned Fractional (UF)

The unsigned fractional numbers can be positive only, and they have nearly twice the magnitude of a signed number with the same number of bits. The unsigned fractional numbers lie in the following range:

$$0{,}0 \le UF \le 2{,}0 - 2^{-[N-1]}$$ **Eqn. 2-4**

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

2.7 User Common Types

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

Mnemonics	Size — bits	Description
Word8	8	To represent 8-bit signed variable/value.
UWord8	8	To represent 16-bit unsigned variable/value.
Word16	16	To represent 16-bit signed variable/value.
UWord16	16	To represent 16-bit unsigned variable/value.
Word32	32	To represent 32-bit signed variable/value.
UWord32	32	To represent 16-bit unsigned variable/value.
Int8	8	To represent 8-bit signed variable/value.
UInt8	8	To represent 16-bit unsigned variable/value.
Int16	16	To represent 16-bit signed variable/value.
UInt16	16	To represent 16-bit unsigned variable/value.
Int32	32	To represent 32-bit signed variable/value.
UInt32	32	To represent 16-bit unsigned variable/value.
Frac16	16	To represent 16-bit signed variable/value.
Frac32	32	To represent 32-bit signed variable/value.
NULL	constant	Represents NULL pointer.
bool	16	Boolean variable.
false	constant	Represents false value.
true	constant	Represents true value.
FRAC16()	macro	Transforms float value from <–1, 1) range into fractional representation <–32768, 32767>.
FRAC32()	macro	Transforms float value from <–1, 1) range into fractional representation <–2147483648, 2147483648>.

2.8 Special Issues

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

Chapter 3 FUNCTION API

3.1 API Summary

Table 3-1. API Functions Summary

Name	Arguments	Output	Description
GFLIB_SinTlr	Frac16 f16In	Frac16	The function calculates the sine value of the argument using 9th order Taylor polynomial approximation.
GFLIB_Sin12Tlr	Frac16 f16In	Frac16	The function calculates the sine value of the argument using 9th order Taylor polynomial approximation. This funcion is quicker with lower precision in camparison to GFLIB_SinTlr.
GFLIB_SinLut	Frac16 f16Arg	Frac16	The function calculates the sine value of the argument using lookup table.
GFLIB_CosTlr	Frac16 f16In	Frac16	The function calculates the cosine value of the argument using 9th order Taylor polynomial approximation.
GFLIB_Cos12Tlr	Frac16 f16In	Frac16	The function calculates the cosine value of the argument using 9th order Taylor polynomial approximation. This funcion is quicker with lower precision in camparison to GFLIB_Cos12Tlr.
GFLIB_CosLut	Frac16 f16Arg	Frac16	The function calculates the cosine value of the argument using lookup table.
GFLIB_Tan	Frac16 f16Arg	Frac16	The function calculates the tangent value of the argument using piece-wise polynomial approximation.
GFLIB_Asin	Frac16 f16Arg	Frac16	The function calculates the arcus sine value of the argument using piece-wise polynomial approximation.
GFLIB_Acos	Frac16 f16Arg	Frac16	The function calculates the arcus cosine value of the argument using piece-wise polynomial approximation.
GFLIB_Atan	Frac16 f16Arg	Frac16	The function calculates the arcus tangent value of the argument using piece-wise polynomial approximation.
GFLIB_AtanYX	Frac16 f16ValY Frac16 f16ValX Int16 *pi16ErrFlag	Frac16	The function calculates the arcus tangent value based on the provided x, y co-ordinates as arguments using division and piece-wise polynomial approximation.

Table 3-1. API Functions Summary

GFLIB_AtanYXShifted	Frac16 f16ValY Frac16 f16ValX GFLIB_ATANYXSHIFTED_T *pudtAtanYXCoeff	Frac16	The function computes angle of two sine waves shifted in phase one to each other.
GFLIB_SqrtPoly	Frac32 f32Arg	Frac16	The function calculates the square root value of the argument using piece-wise polynomial approximation with post-adjustment method.
GFLIB_SqrtIter	Frac32 f32Arg	Frac16	The function calculates the square root value using four iterations. This funcion is quicker with lower precision in camparison to **GFLIB_SqrtPoly**.
GFLIB_Lut	Frac16 f16Arg Frac16 *pf16Table UWord16 uw16TableSize	Frac16	The function approximates a one-dimensional arbitrary user function using the interpolation lookup method. The user function is stored in the table of size specified in uw16TableSize and pointed to by *pTable pointer.
GFLIB_Ramp16	Frac16 f16Desired Frac16 f16Actual GFLIB_RAMP16_T *pudtParam	Frac16	The function calculates 16-bit version of up/down ramp with step increment/decrement defined in pudtParam structure.
GFLIB_Ramp32	Frac32 f32Desired Frac32 f32Actual GFLIB_RAMP32_T *pudtParam	Frac32	The function calculates 32-bit version of up/down ramp with step increment/decrement defined in pudtParam structure.
GFLIB_DynRamp16	Frac16 f16Desired Frac16 f16Instant UWord16 uw16SatFlag GFLIB_DYNRAMP16_T *pudtParam	Frac16	The function calculates a 16-bit version of the ramp with a different set of up/down parameters depending on the state of uw16SatFlag. If uw16SatFlag is set, the ramp counts up/down towards the f16Instant value.
GFLIB_DynRamp32	Frac32 f32Desired Frac32 f32Instant UWord16 uw16SatFlag GFLIB_DYNRAMP32_T *pudtParam	Frac32	The function calculates a 32-bit version of the ramp with a different set of the up/down parameters depending on the state of uw16SatFlag. If uw16SatFlag is set, the ramp counts up/down towards the f32Instant value.
GFLIB_Limit16	Frac16 f16Arg GFLIB_LIMIT16_T *pudtLimit	Frac16	The function calculates 16-bit scalar upper/lower limitation of the input signal.
GFLIB_Limit32	Frac32 f32Arg GFLIB_LIMIT32_T *pudtLimit	Frac32	The function calculates 32-bit scalar upper/lower limitation of the input signal.
GFLIB_LowerLimit16	Frac16 f16Arg Frac16 f16LowerLimit	Frac16	The function calculates 16-bit scalar lower limitation of the input signal.

Table 3-1. API Functions Summary

GFLIB_LowerLimit32	Frac32 f32Arg Frac32 f32LowerLimit	Frac32	The function calculates 32-bit scalar lower limitation of the input signal.
GFLIB_UpperLimit16	Frac16 f16Arg Frac16 f16UpperLimit	Frac16	The function calculates 16-bit scalar upper limitation of the input signal.
GFLIB_UpperLimit32	Frac32 f32Arg Frac32 f32UpperLimit	Frac32	The function calculates 32-bit scalar upper limitation of the input signal.
GFLIB_Sgn	Frac16 f16Arg	Frac16	The function calculates signum of the input argument. The function returns: $7FFF if X > 0 0 if X = 0 $8000 if X < 0
GFLIB_Sgn2	Frac16 f16Arg	Frac16	The function calculates signum of the input argument with zero being considered as positive value. The function returns: $7FFFif X >= 0 $8000 if X < 0
GFLIB_Hyst	GFLIB_HYST_T *pudtHystVar	Frac16	The function switches output between two predefined values when the input crosses the threshold values.
GFLIB_ControllerPIp	Frac16 f16InputErrorK GFLIB_CONTROLLER_PI_P_PARAMS_T *pudtPiParams Int16 *pi16SatFlag	Frac16	The function calculates the parallel form of Proportional-Integral (PI) regulator.
GFLIB_ControllerPIr	Frac16 f16Error GFLIB_CONTROLLER_PI_RECURRENT_T *pudtCtrl	Frac16	The function calculates the recurrent form of Proportional-Integral (PI) regulator.
GFLIB_ControllerPIrLim	Frac16 f16Error GFLIB_CONTROLLER_PI_RECURRENT_LIM_T *pudtCtrl	Frac16	The function calculates the recurrent form of Proportional-Integral (PI) regulator with limitation.
GFLIB_ControllerPIDp	Frac16 f16InputErrorK Frac16 f16InputDErrorK GFLIB_CONTROLLER_PID_P_PARAMS_T *pudtPidParams Int16 *pi16SatFlag Frac16 *pf16InputDErrorK_1	Frac16	The function calculates the parallel form of Proportional-Integral-Derivative (PID) regulator.
GFLIB_ControllerPIDr	Frac16 f16Error GFLIB_CONTROLLER_PID_RECURRENT_ASM_T *pudtCtrl	Frac16	The function calculates the recurrent form of Proportional-Integral-Derivative (PID) regulator.

3.2 GFLIB_SinTlr

The function calculates the sine value of the argument using the ninth order Taylor polynomial approximation.

3.2.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_SinTlr(Frac16 f16In)
```

3.2.2 Prototype

```
asm Frac16 GFLIB_SinTlrFAsm(Frac16 f16In, const GFLIB_SIN_TAYLOR_COEF_T
*pudtSinData)
```

3.2.3 Arguments

Table 3-2. Function Arguments

Name	In/Out	Format	Range	Description
f16In	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in header file GFLIB_types.h
*pudtSinData	In	N/A	N/A	Optional argument; pointer to Taylor polynomial coefficients table.

Table 3-3. `

Typedef	Name	In/Out	Format	Range	Description
GFLIB_SIN_TAYLOR_COEF_T	f32A[5]	In	SF32	0x80000000... 0x7FFFFFFF	Array of 32-bit taylor polynom coefficients.

3.2.4 Availability

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

This library module is targeted for the DSC 56F80xx platforms.

3.2.5 Dependencies

List of all dependent files:

- GFLIB_SinCosTlrAsm.h
- GFLIB_SinCosTlrDefAsm.h
- GFLIB_types.h

3.2.6 Description

The **GFLIB_SinTlr** function computes the sin(pi * x) using the ninth order Taylor polynomial approximation. The ninth order polynomial approximation is sufficient for the 16-bit result.

The function sin(x) using the ninth order Taylor polynomial is expressed as follows:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \frac{x^9}{9!} \qquad \textbf{Eqn. 3-1}$$

$$\sin(x) = x(d_1 + x^2(d_3 + x^2(d_5 + x^2(d_7 + x^2 d_9)))) \qquad \textbf{Eqn. 3-2}$$

where the constants:

$d_1 = 1$

$d_3 = -1 / 3!$

$d_5 = 1 / 5!$

$d_7 = -1 / 7!$

$d_9 = 1 / 9!$

The function sin(pi * x) using the ninth order Taylor polynomial:

$$\sin(\pi \cdot x) = x(c_1 + x^2(c_3 + x^2(c_5 + x^2(c_7 + x^2 c_9)))) \qquad \textbf{Eqn. 3-3}$$

where:

$c_1 = d_1 * pi^1 = pi$

$c_3 = d_3 * pi^3 = -pi^3 / 3!$

$c_5 = d_5 * pi^5 = pi^5 / 5!$

$c_7 = d_7 * pi^7 = -pi^7 / 7!$

$c_9 = d_9 * pi^9 = pi^9 / 9!$

The ninth order polynomial approximation of the sine function has a very good accuracy in the range <-pi/2; pi/2> of the argument, but in wider ranges the calculation error is quickly growing up. To avoid this inaccuracy there is used the symmetry of the sine function [sin(alpha) = sin(pi - alpha)], by this technique the input argument is transferred to be always in the range <-pi/2; pi/2>, therefore the Taylor polynomial is calculated only in the range of the argument <-pi/2; pi/2>.

To make calculations more precise (because in calculations there is used value of x^2 rounded to a 16-bit fractional number), the given argument value x (transferred to be in the range <-0.5; 0.5) due to the sine function symmetry) is shifted by 1 bit to the left (multiplied by 2), then the value of x^2 used in the calculations is in the range <-1; 1) instead of <-0.25; 0.25). Shifting of the x value by 1 bit to the left increases the accuracy of the calculated sin(pi * x) function.

Because the x value is shifted one bit to the left the polynomial coefficients 'c' need to be scaled (shifted to the right):

$b_1 = c_1 / 21 = \ pi / 2$

$b_3 = c_3 / 23 = -pi^3 / 3! / 23$

$b_5 = c_5 / 25 = \ pi^5 / 5! / 25$

$b_7 = c_7 / 27 = -pi^7 / 7! / 27$

$b_9 = c_9 / 29 = \ pi^9 / 9! / 29$

To avoid the saturation error during the polynomial calculation the coefficients 'b' are divided by 2. After the polynomial calculation the result is multiplied by 2 (shifted 1 bit to the left) to take the right result of the function sin(pi * x) in the range <-1; 1) of the given x.

$a_1 = b_1 / 2 = \ pi / 22 = \ 0.785398163$

$a_3 = b_3 / 2 = -pi^3 / 3! / 24 = -0.322982049$

$a_5 = b_5 / 2 = \ pi^5 / 5! / 26 = \ 0.039846313$

$a_7 = b_7 / 2 = -pi^7 / 7! / 28 = -0.002340877$

$a_9 = b_9 / 2 = \ pi^9 / 9! / 210 = \ 0.000080220$

$$\sin(\pi \cdot x) = (x \ll 1)((a_1 + (x \ll 1)^2(a_3 + (x \ll 1)^2(a_5 + (x \ll 1)^2(a_7 + (x \ll 1)a_9)))) \ll 1)$$

Eqn. 3-4

For a better accuracy the 'a' coefficients are used as 32-bit signed fractional constants in the multiplication operations, $(x<<l)^2$ is a 16-bit fractional variable, the result of the '$(x<<1)^2$(a#...)' multiplication is a 32-bit fractional number.

The polynomial coefficients in the 32-bit signed fractional representation:

a_1 = 0x6487ED51

a_3 = 0xD6A88634

a_5 = 0x0519AF1A

a_7 = 0xFFB34B4D

a_9 = 0x0002A0F0

$$\begin{aligned}\sin(\pi \cdot x) = {} & (x \ll 1)(\mathrm{0x6487ED51} + \\ & (x \ll 1)^2(\mathrm{0xD6A88634} + \\ & (x \ll 1)^2(\mathrm{0x0519AF1A} + \\ & (x \ll 1)^2(\mathrm{0xFFB34B4D} + \\ & (x \ll 1)^2(\mathrm{0x0002A0F0})))) \ll 1\end{aligned}$$

Eqn. 3-5

3.2.7 Returns

The function returns the result of $\sin(\pi \cdot x)$.

3.2.8 Range Issues

The input data value is in the range of <-1,1), which corresponds to the angle in the range of <-pi,pi). The output data value is in the range of <-1,1). This means that the function value of the input argument 0.5, which corresponds to pi/2, is 0x7FFF and -0.5, which corresponds to -pi/2, is 0x8000.

3.2.9 Special Issues

The function **GFLIB_SinTlr** is the saturation mode independent.

3.2.10 Implementation

The **GFLIB_SinTlr** function is implemented as a function call.

Example 3-1. Implementation Code

```
#include "gflib.h"

static Frac16 mf16Input;
static Frac16 mf16Output;

/* input data value in range <-1,1) corresponds to <-pi,pi) */
#define PIBY4 0.25 /* 0.25 equals to pi / 4 */

void main(void)
{
        /* input value pi / 4 */
        mf16Input = FRAC16(PIBY4);

        /* Compute the sine value */
        mf16Output = GFLIB_SinTlr(mf16Input);
}
```

3.2.11 See Also

See **GFLIB_Sin12Tlr**, **GFLIB_SinLut**, **GFLIB_CosTlr**, **GFLIB_Cos12Tlr**, **GFLIB_CosLut** and **GFLIB_Tan** for more information.

3.2.12 Performance

Table 3-4. Performance of GFLIB_SinTlr Function

Code Size (words)	38	
Data Size (words)	10	
Execution Clock	Min	52/51 cycles
	Max	52/51 cycles

3.3 GFLIB_Sin12Tlr

The function calculates the sine value of the argument using the ninth order Taylor polynomial approximation. This function has quicker calculation paid by reduced precision to 12 bits.

3.3.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_Sin12Tlr(Frac16 f16In)
```

3.3.2 Prototype

```
asm Frac16 GFLIB_Sin12TlrFAsm(Frac16 f16In, const
GFLIB_SIN12_TAYLOR_COEF_T *pudtSinData)
```

3.3.3 Arguments

Table 3-5. Function Arguments

Name	In/Out	Format	Range	Description
f16In	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in header file GFLIB_types.h
*pudtSinData	In	N/A	N/A	Optional argument; pointer to Taylor polynomial coefficients table.

Table 3-6. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GFLIB_SIN12_TAYLOR_COEF_T	f32A[5]	In	SF32	0x80000000... 0x7FFFFFFF	Array of 32-0bit taylor polynom coefficients.

3.3.4 Availability

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

This library module is targeted for the DSC 56F80xx platforms.

3.3.5 Dependencies

List of all dependent files:

- GFLIB_SinCosTlrAsm.h
- GFLIB_SinCosTlrDefAsm.h
- GFLIB_types.h

3.3.6 Description

The **GFLIB_Sin12Tlr** function computes the sin(pi * x) using the ninth order Taylor polynomial approximation. The ninth order polynomial approximation is sufficient for the 16-bit result.

The function sin(x) using the ninth order Taylor polynomial is expressed as follows:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \frac{x^9}{9!} \qquad \textbf{Eqn. 3-6}$$

$$\sin(x) = x(d_1 + x^2(d_3 + x^2(d_5 + x^2(d_7 + x^2 d_9)))) \qquad \textbf{Eqn. 3-7}$$

where the constants:

$d_1 = 1$
$d_3 = -1 / 3!$
$d_5 = 1 / 5!$
$d_7 = -1 / 7!$
$d_9 = 1 / 9!$

The function sin(pi * x) using the ninth order Taylor polynomial:

$$\sin(\pi \cdot x) = x(c_1 + x^2(c_3 + x^2(c_5 + x^2(c_7 + x^2 c_9)))) \qquad \textbf{Eqn. 3-8}$$

where:

$c_1 = d_1 * pi^1 = pi$
$c_3 = d_3 * pi^3 = -pi^3 / 3!$
$c_5 = d_5 * pi^5 = pi^5 / 5!$
$c_7 = d_7 * pi^7 = -pi^7 / 7!$
$c_9 = d_9 * pi^9 = pi^9 / 9!$

The ninth order polynomial approximation of the sine function has a very good accuracy in the range <-pi/2; pi/2> of the argument, but in wider ranges the calculation error is quickly growing up. To avoid this inaccuracy there is used the symmetry of the sine function [sin(alpha) = sin(pi - alpha)], by this technique the input argument is transferred to be always in the range <-pi/2; pi/2>, therefore the Taylor polynomial is calculated only in the range of the argument <-pi/2; pi/2>.

To make calculations more precise (because in calculations there is used value of x^2 rounded to a 16-bit fractional number), the given argument value x (transferred to be in the range <-0.5; 0.5) due to the sine function symmetry) is shifted by 1 bit to the left (multiplied by 2), then the value of x^2 used in the calculations is in the range <-1; 1) instead of <-0.25; 0.25). Shifting of the x value by 1 bit to the left increases the accuracy of the calculated sin(pi * x) function.

Because the x value is shifted one bit to the left the polynomial coefficients 'c' need to be scaled (shifted to the right):

$b_1 = c_1 / 21 = \ pi / 2$

$b_3 = c_3 / 23 = -pi^3 / 3! / 23$

$b_5 = c_5 / 25 = \ pi^5 / 5! / 25$

$b_7 = c_7 / 27 = -pi^7 / 7! / 27$

$b_9 = c_9 / 29 = \ pi^9 / 9! / 29$

To avoid the saturation error during the polynomial calculation the coefficients 'b' are divided by 2. After the polynomial calculation the result is multiplied by 2 (shifted 1 bit to the left) to take the right result of the function sin(pi * x) in the range <-1; 1) of the given x.

$a_1 = b_1 / 2 = \ pi / 22 = \ 0.785398163$

$a_3 = b_3 / 2 = -pi^3 / 3! / 24 = -0.322982049$

$a_5 = b_5 / 2 = \ pi^5 / 5! / 26 = \ 0.039846313$

$a_7 = b_7 / 2 = -pi^7 / 7! / 28 = -0.002340877$

$a_9 = b_9 / 2 = \ pi^9 / 9! / 210 = \ 0.000080220$

$$\sin(\pi \cdot x) = (x \ll 1)((a_1 + (x \ll 1)^2(a_3 + (x \ll 1)^2(a_5 + (x \ll 1)^2(a_7 + (x \ll 1)a_9)))) \ll 1)$$

Eqn. 3-9

For a better accuracy the 'a' coefficients are used as 16-bit signed fractional constants in the multiplication operations, $(x<<l)^2$ is a 16-bit fractional variable, the result of the '$(x<<1)^2$(a#...)' multiplication is a 32-bit fractional number.

The polynomial coefficients in the 16-bit signed fractional representation:

$a_1 = 0x6488$

$a_3 = 0xD6A9$

$a_5 = 0x051A$

$a_7 = 0xFFB3$

$a_9 = 0x0003$

$$\begin{aligned}\sin(\pi \cdot x) = {} & (x \ll 1)(0x6488 + \\ & (x \ll 1)^2(0xD6A9 + \\ & (x \ll 1)^2(0x051A + \\ & (x \ll 1)^2(0xFFB3 + \\ & (x \ll 1)^2(0x0003))))) \ll 1\end{aligned}$$

Eqn. 3-10

3.3.7 Returns

The function returns the result of $\sin(\pi \cdot x)$.

3.3.8 Range Issues

The input data value is in the range of <-1,1), which corresponds to the angle in the range of <-pi,pi). The output data value is in the range of <-1,1). This means that the function value of the input argument 0.5, which corresponds to pi/2, is 0x7FFF and -0.5, which corresponds to -pi/2, is 0x8000.

3.3.9 Special Issues

The function **GFLIB_Sin12Tlr** is the saturation mode independent.

3.3.10 Implementation

The **GFLIB_Sin12Tlr** function is implemented as a function call.

Example 3-2. Implementation Code

```
#include "gflib.h"

static Frac16 mf16Input;
static Frac16 mf16Output;

/* input data value in range <-1,1) corresponds to <-pi,pi) */
#define PIBY4 0.25 /* 0.25 equals to pi / 4 */

void main(void)
{
        /* input value pi / 4 */
        mf16Input = FRAC16(PIBY4);

        /* Compute the sine value */
        mf16Output = GFLIB_Sin12Tlr(mf16Input);
}
```

3.3.11 See Also

See **GFLIB_SinTlr**, **GFLIB_SinLut**, **GFLIB_CosTlr**, **GFLIB_Cos12Tlr**, **GFLIB_CosLut** and **GFLIB_Tan** for more information.

3.3.12 Performance

Table 3-7. Performance of GFLIB_Sin12Tlr Function

Code Size (words)	25	
Data Size (words)	5	
Execution Clock	Min	38 cycles
	Max	38 cycles

3.4 GFLIB_SinLut

The function calculates the sine value of the argument using lookup table.

3.4.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_SinLut(Frac16 f16Arg)
```

3.4.2 Prototype

```
asm Frac16 GFLIB_SinLutFAsm(Frac16 f16Arg, Frac16 *pudtSinTable, UWord16
uw16TableSize)
```

3.4.3 Arguments

Table 3-8. Function Arguments

Name	In/Out	Format	Range	Description
f16Arg	In	SF16	0x8000... 0x7FFF	input argument; the **Frac16** data type is defined in header file GFLIB_types.h
*pudtSinTable	In	N/A	N/A	Pointer to the 1q sine values table
uw16TableSize	In	UI16	0x0... 0xFFFF	The sine table size in bit shifts of number 1.

3.4.4 Availability

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

This library module is targeted for the DSC 56F80xx platforms.

3.4.5 Dependencies

List of all dependent files:

- GFLIB_SinLutAsm.h
- GFLIB_SinCosDefAsm.h
- GFLIB_types.h

3.4.6 Description

The **GFLIB_SinLut** uses a table of precalculated function points. These points are selected with a fixed step and must be in a number of 2^n, where n can be 1 through to 15. The table contains $2^n + 1$ points.

The function finds two nearest precalculated points of the input argument and using the linear interpolation between these two points calculates the output value.

The sin function is a symmetrical along the defined interval, therefore the table contains precalculated values for the range -pi/2 to 0. For the values outside this interval, the function transforms the input value to the -pi/2 to 0 interval and calculates as if it was in this interval. In the end if the input was in the interval of 0 to pi the output is negated.

Figure 0-1 shows the function that has 9 table points, i.e. $2^3 + 1$, therefore the table size is 3.

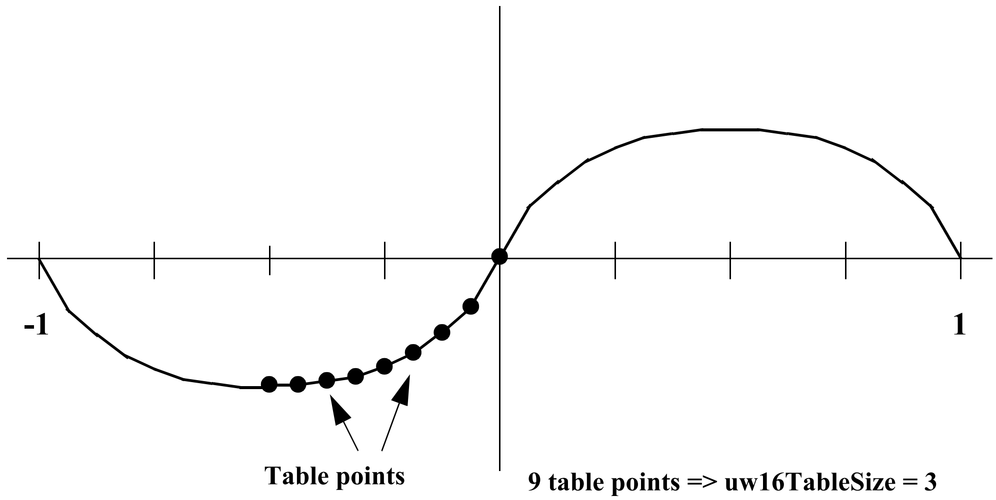

Figure 3-1. Algorithm Diagram

The **GFLIB_SinLut** function by default uses a sin table of 257 points.

3.4.7 Returns

The function returns the result of $\sin(\pi \cdot x)$.

3.4.8 Range Issues

The input data value is in the range of <-1,1), which corresponds to the angle in the range of <-pi,pi). The output data value is in the range of <-1,1). This means that with the input value 0, it has the output result of 0. Similarly if the input value is 0.5, the output is 1.

3.4.9 Special Issues

The function **GFLIB_SinLut** requires the saturation mode to be set.

3.4.10 Implementation

The **GFLIB_SinLut** function is implemented as a function call.

Example 3-3. Implementation Code

```
#include "gflib.h"

static Frac16 mf16Input;
static Frac16 mf16Output;

/* input data value in range <-1,1) corresponds to <-pi,pi) */
#define PIBY4 0.25 /* 0.25 equals to pi / 4 */

void main(void)
{
        /* input value pi / 4 */
        mf16Input = FRAC16(PIBY4);

        /* turns on the saturation */
        __turn_on_sat();

        /* Compute the sine value */
        mf16Output = GFLIB_SinLut(mf16Input);

        /* turns off the saturation */
        __turn_off_sat();

}
```

3.4.11 See Also

See **GFLIB_SinTlr**, **GFLIB_Sin12Tlr**, **GFLIB_CosLut**, **GFLIB_CosTlr**, **GFLIB_Cos12Tlr** and **GFLIB_Tan** for more information.

3.4.12 Performance

Table 3-9. Performance of GFLIB_SinLut function

Code Size (words)	43	
Data Size (words)	258	
Execution Clock	Min	61/62 cycles
	Max	61/62 cycles

3.5 GFLIB_CosTlr

The function calculates the cosine value of the argument using the 9th order Taylor polynomial approximation. The function is implemented as inline reusing the **GFLIB_SinTlr** function.

3.5.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_CosTlr(Frac16 f16In)
```

3.5.2 Prototype

```
inline Frac16 GFLIB_CosTlrFAsmi(Frac16 f16In, const
GFLIB_SIN_TAYLOR_COEF_T *pudtSinData)
```

3.5.3 Arguments

Table 3-10. Function Arguments

Name	In/Out	Format	Range	Description
f16In	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in header file GFLIB_types.h
*pudtSinData	In	N/A	N/A	Pointer to Taylor polynomial coefficients

Table 3-11. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GFLIB_SIN_TAYLOR_COEF_T	f32A[5]	In	SF32	0x80000000... 0x7FFFFFFF	Array of 32bit taylor polynom coefficients.

3.5.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.5.5 Dependencies

List of all dependent files:

- GFLIB_SinCosTlrAsm.h
- GFLIB_SinCosTlrDefAsm.h
- GFLIB_types.h

3.5.6 Description

The **GFLIB_CosTlr** function computes cos(pi * x) using 9th order Taylor polynomial approximation of the sine function where its equation is:

$$\cos(angle) = \sin\left(\frac{\pi}{2} - ABS(angle)\right) \qquad \textbf{Eqn. 3-11}$$

Then the cosine function is calculated using the sine function. The 9th order polynomial approximation is sufficient for the 16-bit result.

The function sin(x) using the 9th order Taylor polynomial:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \frac{x^9}{9!} \qquad \textbf{Eqn. 3-12}$$

$$\sin(x) = x(d_1 + x^2(d_3 + x^2(d_5 + x^2(d_7 + x^2 d_9)))) \qquad \textbf{Eqn. 3-13}$$

where constants:

$d_1 = 1$
$d_3 = -1 / 3!$
$d_5 = 1 / 5!$
$d_7 = -1 / 7!$
$d_9 = 1 / 9!$

The function sin(pi * x) using the 9th order Taylor polynomial:

$$\sin(\pi \cdot x) = x(c_1 + x^2(c_3 + x^2(c_5 + x^2(c_7 + x^2 c_9)))) \qquad \textbf{Eqn. 3-14}$$

where the constants:

$c_1 = d_1 * pi^1 = pi$
$c_3 = d_3 * pi^3 = -pi^3 / 3!$
$c_5 = d_5 * pi^5 = pi^5 / 5!$
$c_7 = d_7 * pi^7 = -pi^7 / 7!$
$c_9 = d_9 * pi^9 = pi^9 / 9!$

The 9th order polynomial approximation of the sine function has a very good accuracy in the range <-pi/2; pi/2> of the argument, but in wider ranges the calculation error is quickly growing up. To avoid this inaccuracy there is used the symmetry of the sine function [sin(alpha) = sin(pi - alpha)], by this technique the input argument is transferred to be always in the range <-pi/2; pi/2>, therefore the Taylor polynomial is calculated only in the range of argument <-pi/2; pi/2>.

To make the calculations more precise (because in calculations there is used the value of x^2 rounded to a 16-bit fractional number), the given argument value x (transferred to be in the range <-0.5; 0.5) due to sine function symmetry) is shifted by 1 bit to the left (multiplied by 2), then the value of x^2 used in the calculations is in the range <-1; 1) instead of <-0.25; 0.25). Shifting of the x value by 1 bit to the left increases the accuracy of the calculated sin(pi * x) function.

Because the x value is shifted one bit to the left the polynomial coefficients 'c' needs to be scaled (shifted to the right):

$b_1 = c_1 / 21 = pi / 2$

$b_3 = c_3 / 23 = -pi^3 / 3! / 23$

$b_5 = c_5 / 25 = pi^5 / 5! / 25$

$b_7 = c_7 / 27 = -pi^7 / 7! / 27$

$b_9 = c_9 / 29 = pi^9 / 9! / 29$

To avoid the saturation error during the polynomial calculation the coefficients 'b' are divided by 2. After the polynomial calculation the result is multiplied by 2 (shifted 1 bit to the left) to get the correct result of the function sin(pi * x) in the range <-1; 1) of the given x.

$a_1 = b_1 / 2 = pi / 22 = 0.785398163$

$a_3 = b_3 / 2 = -pi^3 / 3! / 24 = -0.322982049$

$a_5 = b_5 / 2 = pi^5 / 5! / 26 = 0.039846313$

$a_7 = b_7 / 2 = -pi^7 / 7! / 28 = -0.002340877$

$a_9 = b_9 / 2 = pi^9 / 9! / 210 = 0.000080220$

$$\sin(\pi \cdot x) = (x \ll 1)((a_1 + (x \ll 1)^2(a_3 + (x \ll 1)^2(a_5 + (x \ll 1)^2(a_7 + (x \ll 1)a_9)))) \ll 1)$$ **Eqn. 3-15**

For a better accuracy the 'a' coefficients are used as 32-bit signed fractional constants in the multiplication operations, $(x<<1)^2$ is a16-bit fractional variable, the result of the '$(x<<1)^2$(a#...)' multiplication operation is a 32-bit fractional number.

The polynomial coefficients in the 32-bit signed fractional representation:

a_1 = 0x6487ED51

a_3 = 0xD6A88634

a_5 = 0x0519AF1A

a_7 = 0xFFB34B4D

a_9 = 0x0002A0F0

$$\sin(\pi \cdot x) = (x \ll 1)(0x6487ED51 + (x \ll 1)^2(0xD6A88634 + (x \ll 1)^2(0x0519AF1A + (x \ll 1)^2(0xFFB34B4D + (x \ll 1)^2(0x0002A0F0))))) \ll 1$$

Eqn. 3-16

3.5.7 Returns

The function returns the result of $\cos(\pi \cdot x)$.

3.5.8 Range Issues

The input data value is in the range of <-1,1), which corresponds to the angle in the range of <-pi,pi). The output data value is in the range of <-1,1). It means that the function value of the input argument 0.5, which corresponds to pi/2, is 0x7FFF and -0.5, which corresponds to -pi/2, is 0x8000.

3.5.9 Special Issues

The function **GFLIB_CosTlr** is the saturation mode independent.

3.5.10 Implementation

The **GFLIB_CosTlr** function is implemented as a function call.

Example 3-4. Implementation Code

```
#include "gflib.h"

static Frac16 mf16Input;
static Frac16 mf16Output;

/* input data value in range <-1,1) corresponds to <-pi,pi) */
#define PIBY4 0.25 /* 0.25 equals to pi / 4 */

void main(void)
{
        /* input value pi / 4 */
        mf16Input = FRAC16(PIBY4);

        /* Compute the cosine value */
        mf16Output = GFLIB_CosTlr(mf16Input);

}
```

3.5.11 See Also

See **GFLIB_Cos12Tlr**, **GFLIB_CosLut**, **GFLIB_SinTlr**, **GFLIB_Sin12Tlr**, **GFLIB_SinLut** and **GFLIB_Tan** for more information.

3.5.12 Performance

Table 3-12. Performance of GFLIB_CosTlr function

Code Size (words)	22 + 38 (GFLIB_SinTlr)	
Data Size (words)	0 + 10 (GFLIB_SinTlr)	
Execution Clock	Min	56 cycles
	Max	56 cycles

3.6 GFLIB_Cos12Tlr

The function calculates the cosine value of the argument using the 9th order Taylor polynomial approximation. The function is implemented as inline reusing the **GFLIB_Sin12Tlr** function. This function has quicker calculation paid by reduced precision to 12 bits.

3.6.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_Cos12Tlr(Frac16 f16In)
```

3.6.2 Prototype

```
extern inline Frac16 GFLIB_Cos12TlrFAsmi(Frac16 f16In, const
GFLIB_SIN_TAYLOR_COEF_T *pudtSinData)
```

3.6.3 Arguments

Table 3-13. Function Arguments

Name	In/Out	Format	Range	Description
f16In	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in header file GFLIB_types.h
*pudtSinData	In	N/A	N/A	Pointer to Taylor polynomial coefficients

Table 3-14. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GFLIB_SIN_TAYLOR_COEF_T	f32A[5]	In	SF32	0x80000000... 0x7FFFFFFF	Array of 32bit taylor polynom coefficients.

3.6.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.6.5 Dependencies

List of all dependent files:

- GFLIB_SinCosTlrAsm.h
- GFLIB_SinCosTlrDefAsm.h
- GFLIB_types.h

3.6.6 Description

The **GFLIB_Cos12Tlr** function computes cos(pi * x) using 9th order Taylor polynomial approximation of the sine function where its equation is:

$$\cos(\alpha) = \sin\left[\frac{\pi}{2} + |\alpha|\right] \quad \text{Eqn. 3-17}$$

Then the cosine function is calculated using the sine function. The 9th order polynomial approximation is sufficient for the 16-bit result.

The function sin(x) using the 9th order Taylor polynomial:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \frac{x^9}{9!} \quad \text{Eqn. 3-18}$$

$$\sin(x) = x(d_1 + x^2(d_3 + x^2(d_5 + x^2(d_7 + x^2 d_9)))) \quad \text{Eqn. 3-19}$$

where constants:

$d_1 = 1$
$d_3 = -1 / 3!$
$d_5 = 1 / 5!$
$d_7 = -1 / 7!$
$d_9 = 1 / 9!$

The function sin(pi * x) using the 9th order Taylor polynomial:

$$\sin(\pi \cdot x) = x(c_1 + x^2(c_3 + x^2(c_5 + x^2(c_7 + x^2 c_9)))) \quad \text{Eqn. 3-20}$$

where the constants:

$c_1 = d_1 * pi^1 = pi$
$c_3 = d_3 * pi^3 = -pi^3 / 3!$
$c_5 = d_5 * pi^5 = pi^5 / 5!$
$c_7 = d_7 * pi^7 = -pi^7 / 7!$
$c_9 = d_9 * pi^9 = pi^9 / 9!$

The 9th order polynomial approximation of the sine function has a very good accuracy in the range <-pi/2; pi/2> of the argument, but in wider ranges the calculation error is quickly growing up. To avoid this inaccuracy there is used the symmetry of the sine function [sin(alpha) = sin(pi - alpha)], by this technique the input argument is transferred to be always in the range <-pi/2; pi/2>, therefore the Taylor polynomial is calculated only in the range of argument <-pi/2; pi/2>.

To make the calculations more precise (because in calculations there is used the value of x^2 rounded to a 16-bit fractional number), the given argument value x (transferred to be in the range <-0.5; 0.5) due to sine function symmetry) is shifted by 1 bit to the left (multiplied by 2), then the value of x^2 used in the calculations is in the range <-1; 1) instead of <-0.25; 0.25). Shifting of the x value by 1 bit to the left increases the accuracy of the calculated sin(pi * x) function.

Because the x value is shifted one bit to the left the polynomial coefficients 'c' needs to be scaled (shifted to the right):

$b_1 = c_1$ / 21 = pi / 2

$b_3 = c_3$ / 23 = -pi^3 / 3! / 23

$b_5 = c_5$ / 25 = pi^5 / 5! / 25

$b_7 = c_7$ / 27 = -pi^7 / 7! / 27

$b_9 = c_9$ / 29 = pi^9 / 9! / 29

To avoid the saturation error during the polynomial calculation the coefficients 'b' are divided by 2. After the polynomial calculation the result is multiplied by 2 (shifted 1 bit to the left) to get the correct result of the function sin(pi * x) in the range <-1; 1) of the given x.

$a_1 = b_1$ / 2 = pi / 22 = 0.785398163

$a_3 = b_3$ / 2 = -pi^3 / 3! / 24 = -0.322982049

$a_5 = b_5$ / 2 = pi^5 / 5! / 26 = 0.039846313

$a_7 = b_7$ / 2 = -pi^7 / 7! / 28 = -0.002340877

$a_9 = b_9$ / 2 = pi^9 / 9! / 210 = 0.000080220

$$\sin(\pi \cdot x) = (x \ll 1)((a_1 + (x \ll 1)^2(a_3 + (x \ll 1)^2(a_5 + (x \ll 1)^2(a_7 + (x \ll 1)a_9)))) \ll 1)$$ **Eqn. 3-21**

For a better accuracy the 'a' coefficients are used as 32-bit signed fractional constants in the multiplication operations, $(x<<1)^2$ is a16-bit fractional variable, the result of the '$(x<<1)^2$(a#...)' multiplication operation is a 32-bit fractional number.

The polynomial coefficients in the 32-bit signed fractional representation:

a_1 = 0x6488

a_3 = 0xD6A9

a_5 = 0x051A

a_7 = 0xFFB3

a_9 = 0x0003

$$\sin(\pi \cdot x) = (x \ll 1)(0x6488 + (x \ll 1)^2(0xD6A9 + (x \ll 1)^2(0x051A + (x \ll 1)^2(0xFFB3 + (x \ll 1)^2(0x0003))))) \ll 1 \quad \text{Eqn. 3-22}$$

3.6.7 Returns

The function returns the result of $\cos(\pi \cdot x)$.

3.6.8 Range Issues

The input data value is in the range of <-1,1), which corresponds to the angle in the range of <-pi,pi). The output data value is in the range of <-1,1). It means that the function value of the input argument 0.5, which corresponds to pi/2, is 0x7FFF and -0.5, which corresponds to -pi/2, is 0x8000.

3.6.9 Special Issues

The function **GFLIB_Cos12Tlr** is the saturation mode independent.

3.6.10 Implementation

The **GFLIB_Cos12Tlr** function is implemented as a function call.

Example 3-5. Implementation Code

```
#include "gflib.h"

static Frac16 mf16Input;
static Frac16 mf16Output;

/* input data value in range <-1,1) corresponds to <-pi,pi) */
#define PIBY4 0.25 /* 0.25 equals to pi / 4 */

void main(void)
{
        /* input value pi / 4 */
        mf16Input = FRAC16(PIBY4);

        /* Compute the cosine value */
        mf16Output = GFLIB_Cos12Tlr(mf16Input);

}
```

3.6.11 See Also

See **GFLIB_CosTlr**, **GFLIB_CosLut**, **GFLIB_SinTlr**, **GFLIB_Sin12Tlr**, **GFLIB_SinLut** and **GFLIB_Tan** for more information.

3.6.12 Performance

Table 3-15. Performance of GFLIB_Cos12Tlr function

Code Size (words)	11 + 25 (GFLIB_Sin12Tlr)	
Data Size (words)	0 + 5 (GFLIB_Sin12Tlr)	
Execution Clock	Min	42/43 cycles
	Max	42/43 cycles

3.7 GFLIB_CosLut

The function calculates the cosine value of the argument using lookup table.

3.7.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_CosLut(Frac16 f16Arg)
```

3.7.2 Prototype

```
asm Frac16 GFLIB_CosLutFAsm(Frac16 f16Arg, Frac16 *pudtSinTable, UWord16
uw16TableSize)
```

3.7.3 Arguments

Table 3-16. Function Arguments

Name	In/Out	Format	Range	Description
f16Arg	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in header file GFLIB_types.h
*pudtSinTable	In	N/A	N/A	Pointer to the 1q sin values table
uw16TableSize	In	UI16	0x0... 0xFFFF	The sine table size in bit shifts of number 1.

3.7.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.7.5 Dependencies

List of all dependent files:

- GFLIB_CosLutAsm.h
- GFLIB_SinCosDefAsm.h
- GFLIB_types.h

3.7.6 Description

The **GFLIB_CosLut** function uses a table of precalculated function points. These points are selected with a fixed step and must be in a number of 2^n, where n can be 1 through to 15. The table contains $2^n + 1$ points.

The function finds two nearest precalculated points of the input argument and using the linear interpolation between these two points calculates the output value.

The cos function is a symmetrical along the defined interval, therefore the table contains precalculated values for the range -pi to -pi/2. This interval is used to share the sine function table to save memory space. For the values outside this interval, the function transforms the input value to the -pi to -pi/2 interval and calculates as if it was in this interval. In the end if the input was in the interval of -pi/2 to pi/2 the output is negated.

Figure 3-2 shows the function that has 9 table points, i.e. $2^3 + 1$, therefore the table size is 3.

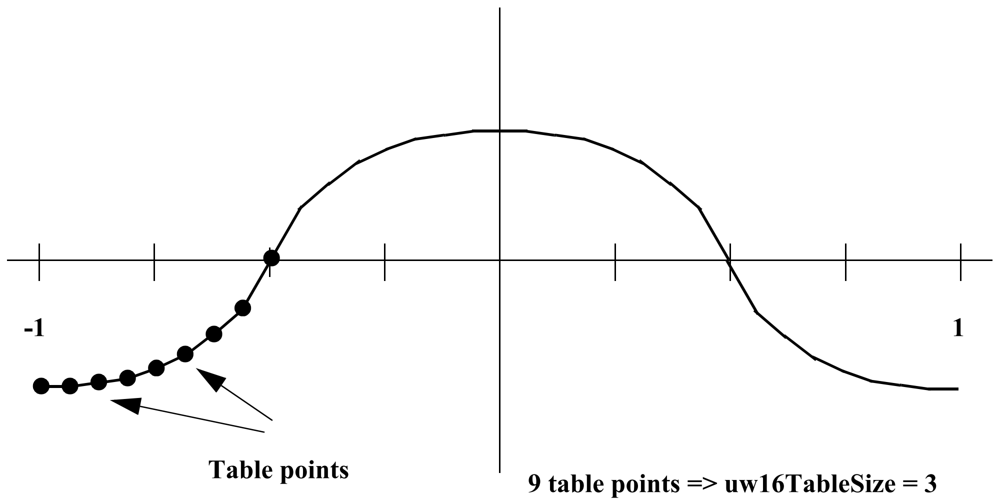

Figure 3-2. Algorithm diagram

The function **GFLIB_CosLut** by default uses a sin table of 257 points.

3.7.7 Returns

The function returns the result of $\cos(\pi \cdot f16Arg)$.

3.7.8 Range Issues

The input data value is in the range of <-1,1), which corresponds to the angle in the range of <-pi,pi) and the output data value is in the range <-1, 1). It means that with the input value 0, it has the output result of 1. Similarly if the input value is +-1, the output is -1.

3.7.9 Special Issues

The function **GFLIB_CosLut** requires the saturation mode to be set.

3.7.10 Implementation

The **GFLIB_CosLut** function is implemented as a function call.

Example 3-6. Implementation Code

```
#include "gflib.h"

static Frac16 mf16Input;
static Frac16 mf16Output;

/* input data value in range <-1,1) corresponds to <-pi,pi) */
#define PIBY4 0.25 /* 0.25 equals to pi / 4 */

void main(void)
{
        /* input value pi / 4 */
        mf16Input = FRAC16(PIBY4);

        /* turns on the saturation */
        __turn_on_sat();

        /* Compute the cosine value */
        mf16Output = GFLIB_CosLut(mf16Input);

        /* turns off the saturation */
        __turn_off_sat();

}
```

3.7.11 See Also

See **GFLIB_CosTlr**, **GFLIB_Cos12Tlr**, **GFLIB_SinLut**, **GFLIB_SinTlr**, **GFLIB_Sin12Tlr** and **GFLIB_Tan** for more information.

3.7.12 Performance

Table 3-17. Performance of GFLIB_CosLut function

Code Size (words)	41	
Data Size (words)	258	
Execution Clock	Min	59 cycles
	Max	59 cycles

3.8 GFLIB_Tan

The function calculates the tangent value of the argument using the piece-wise polynomial approximation.

3.8.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_Tan(Frac16 f16Arg)
```

3.8.2 Prototype

```
asm Frac16 GFLIB_TanFAsm(Frac16 f16Arg, const
GFLIB_TAN_COEFFICIENTS_ADDR_T *pudtTanPoly)
```

3.8.3 Arguments

Table 3-18. Function Arguments

Name	In/Out	Format	Range	Description
f16Arg	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in header file GFLIB_types.h
*pudtTanPoly	In	N/A	N/A	Optional argument; pointer to a structure containing polynomial coefficients for the intervals where the function is calculated.

3.8.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.8.5 Dependencies

List of all dependent files:

- GFLIB_TanAsm.h
- GFLIB_TanAsmDef.h
- GFLIB_types.h

3.8.6 Description

The **GFLIB_Tan** function computes tan(pi*x) using the piece-wise polynomial approximation.

Due to the limits of the fractional arithmetic all tangent values falling beyond <-1, 1), are truncated to -1 and 1. This function evaluates polynomial according to the Horner scheme. The input and output values are additionally scaled and centered for the best accuracy.

$$u = (x - x_{\text{offset}}) \cdot 2^{k_X} \qquad \text{Eqn. 3-23}$$

$$v = ((a_n \cdot u + a_{n-1}) \cdot u + a_{n-2}) \cdot u \ldots + a_0 \qquad \text{Eqn. 3-24}$$

$$y = v \cdot 2^{k_Y} + y_{\text{offset}} \qquad \text{Eqn. 3-25}$$

where

x is the input argument (f16Arg)

y is the evaluated value

u, v are the intermediate variables

$a_n, a_{n-1} \ldots a_0$ are the coefficients of the polynomial

k_X is the scaling factor for the input value

x_{offset} is the offset for the input value

k_Y is the scaling factor for the output value

y_{offset} is the offset for the output value

The function input range is divided into three intervals within which all the coefficients are calculated as follows:

The coefficients for the input argument falling into the interval $<-0{,}125\pi\ ;\ 0)$

$k_X = 4$

$x_{\text{offset}} = -2048$

$a_6, a_5, a_4, a_3, a_2, a_1, a_0 = -1,\ 7,\ -28, 385, -1045, 26754, -4548$

$y_{\text{offset}} = -176322970L$

$k_Y = 3$

The coefficients for the interval $< -0{,}25\pi\ ;\ -0{,}125\pi)$

$k_X = 4$

$x_{\text{offset}} = -6144$

$a_6, a_5, a_4, a_3, a_2, a_1, a_0 = -5, 23, -104, 559, -2442, 18613, -11022$

$y_{\text{offset}} = -536870912L$

$k_Y = 2$

The coefficients for the interval <0>

$k_X = 0$

$x_{\text{offset}} = 0$

$a_6, a_5, a_4, a_3, a_2, a_1, a_0 = 0, 0, 0, 0$

$y_{\text{offset}} = 0L$

$$k_Y \quad = 0$$

The scaling factors and offsets should be set to use the whole available fractional range of <-1, 1).

The exact values are provided in the table Figure 3-19 (see Figure 3-3).

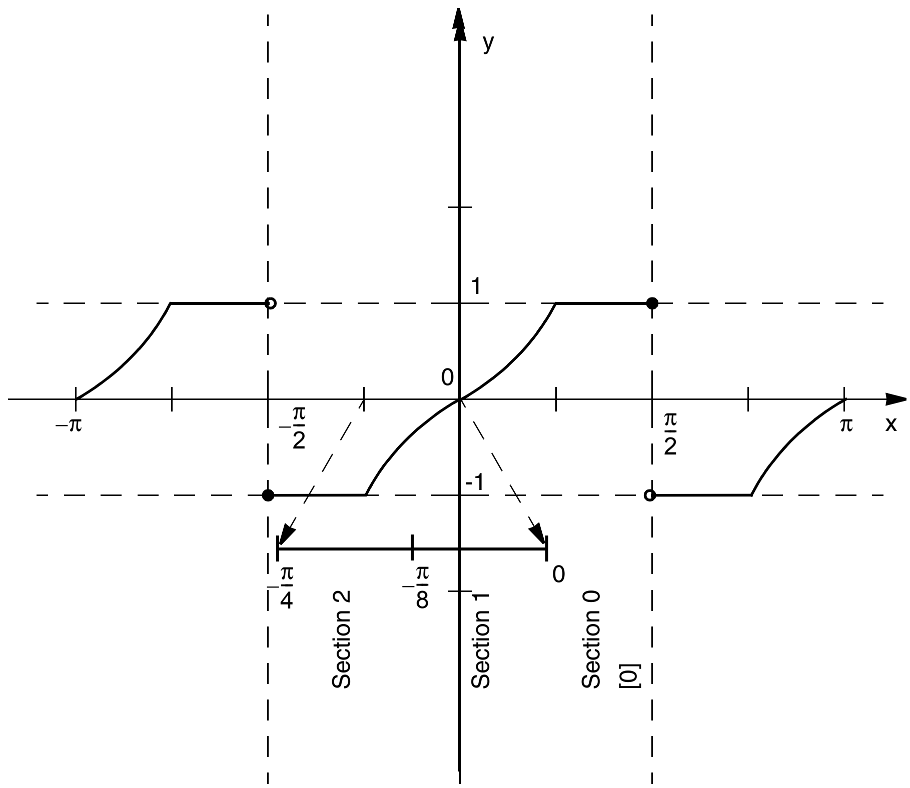

Figure 3-3. GFLIB_Tan Function

Table 3-19. GFLIB_Tan Function Values

Input Argument Value		Computed Result	
Real	**Fractional (Hex)**	**Real**	**Fractional (Hex)**
$\left(-\pi, -\frac{3}{4} \cdot \pi\right)$	(8000, A000)	Tan	Tan
$-\frac{3}{4} \cdot \pi$ s	A000	1.0	7FFF
$\left(-\frac{3}{4} \cdot \pi, -\frac{1}{2} \cdot \pi\right)$	(A000, C000)	1.0	7FFF
$-\frac{1}{2} \cdot \pi$	C000	-1.0	8000

Table 3-19. GFLIB_Tan Function Values

Input Argument Value		Computed Result	
Real	Fractional (Hex)	Real	Fractional (Hex)
$\left(-\frac{1}{2}\cdot\pi, -\frac{1}{4}\cdot\pi\right)$	(C000, E000)	-1.0	8000
$-\frac{1}{4}\cdot\pi$	E000	-1.0	8000
$\left(-\frac{1}{4}\cdot\pi, \frac{1}{4}\cdot\pi\right)$	(E000, 2000)	Tan	Tan
$\frac{1}{4}\cdot\pi$	2000	1.0	7FFF
$\left(\frac{1}{4}\cdot\pi, \frac{1}{2}\cdot\pi\right)$	(2000,4000)	1.0	7FFF
$\frac{1}{2}\cdot\pi$	4000	1.0	7FFF
$\left(\frac{1}{2}\cdot\pi, \frac{3}{2}\cdot\pi\right)$	(4000, 6000)	-1.0	8000
$\left(\frac{3}{4}\cdot\pi, \pi\right)$	(6000,7FFF)	Tan	Tan

3.8.7 Returns

The function returns tan(pi*x) with limits imposed by the fractional arithmetic.

3.8.8 Range Issues

The input data value is in the range of <-1,1), which corresponds to the angle in the range of <-pi,pi). The output data value is in the range of <-1,1), i.e. the function value of 0.25 is 1. If the input is -0.25, the result is -1.

If a tangent value is beyond the permissible range of <-1, 1), then it is truncated to -1 and 1 respectively.

3.8.9 Special Issues

The **GFLIB_Tan** function is the saturation mode independent.

3.8.10 Implementation

The **GFLIB_Tan** function is implemented as a function call.

Example 3-7. Implementation Code

```
#include "gflib.h"

static Frac16 mf16Input;
static Frac16 mf16Output;

/* input data value in range <-1,1) corresponds to <-pi,pi) */
#define PIBY4 0.25 /* 0.25 equals to pi / 4 */

void main(void)
{
        /* input value pi / 4 */
        mf16Input = FRAC16(PIBY4);

        /* Compute the tan value */
        mf16Output = GFLIB_Tan(mf16Input);
}
```

3.8.11 See Also

See **GFLIB_SinTlr**, **GFLIB_Sin12Tlr**, **GFLIB_SinLut**, **GFLIB_CosTlr**, **GFLIB_Cos12Tlr** and **GFLIB_CosLut** for more information.

3.8.12 Performance

Table 3-20. Performance of GFLIB_Tan function

Code Size (words)	59	
Data Size (words)	39	
Execution Clock	Min	76cycles
	Max	76 cycles

3.9 GFLIB_Asin

The function calculates the arcus sine value of the argument using the piece-wise polynomial approximation.

3.9.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_Asin(Frac16 f16Arg)
```

3.9.2 Prototype

```
asm Frac16 GFLIB_AsinFAsm(Frac16 f16Arg,
GFLIB_ASINACOS_COEFFICIENTS_ADDR_T *pudtAsinPoly)
```

3.9.3 Arguments

Table 3-21. Function Arguments

Name	In/Out	Format	Range	Description
f16Arg	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in header file GFLIB_types.h
*pudtAsinPoly	In	N/A	N/A	Optional argument; pointer to data needed for the calculation.

3.9.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.9.5 Dependencies

List of all dependent files:

- GFLIB_AsinAcosAsm.h
- GFLIB_AsinAcosAsmDef.h
- GFLIB_types.h

3.9.6 Description

The **GFLIB_Asin** function computes the asin(x)/(pi/2) using the piece-wise polynomial approximation (see Figure 3-4). This function evaluates the polynomial according to the Horner scheme. The input and output values are additionally scaled and centered for the best accuracy.

$$u = (x - x_{\text{offset}}) \cdot 2^{k_X}$$ **Eqn. 3-26**

$$v = ((a_n \cdot u + a_{n-1}) \cdot u + a_{n-2}) \cdot u \ldots + a_0$$ **Eqn. 3-27**

$$y = v \cdot 2^{k_Y} + y_{\text{offset}}$$ **Eqn. 3-28**

where

x is the input argument (f16Arg)

y is the evaluated value

u, v are the intermediate variables

$a_n, a_{n-1} \ldots a_0$ are the coefficients of the polynomial

k_X is the scaling factor for the input value

x_{offset} is the offset for the input value

k_Y is the scaling factor for the output value

y_{offset} is the offset for the output value

The function input range is divided into three intervals within which all the coefficients are calculated as follows:

The coefficients for the input argument falling into the interval <-0.5 ; 0)

$k_X = 3$

$x_{\text{offset}} = -4096$

$a_3, a_2, a_1, a_0 = 59, -169, 21026, -11514$

$y_{\text{offset}} = -19252860L$

$k_Y = 5$

The coefficients for the interval <-1; - 0.5)

$u = (-\sqrt{|x|} - x_{\text{offset}}) \cdot 2^{k_X}$

$v = (((a_n \cdot u + a_{n-1}) \cdot u + a_{n-2}) \cdot u \ldots + a_0)$

$y = v \cdot 2^{k_Y} + y_{\text{offset}}$

$k_X = 3$

$x_{\text{offset}} = -12288$

$a_3, a_2, a_1, a_0 = 104, -623, 22502, -9536$

$y_{\text{offset}} = -111848107L$

$k_Y = 5$

The coefficients for the interval <0>

$k_X = 0$

$x_{\text{offset}} = 0$

$a_3, a_2, a_1, a_0 = 0, 0, 0, 0$

$y_{\text{offset}} = 0L$

$$k_Y = 0$$

The scaling factors and offsets should be set to use the whole available fractional range of <-1, 1).

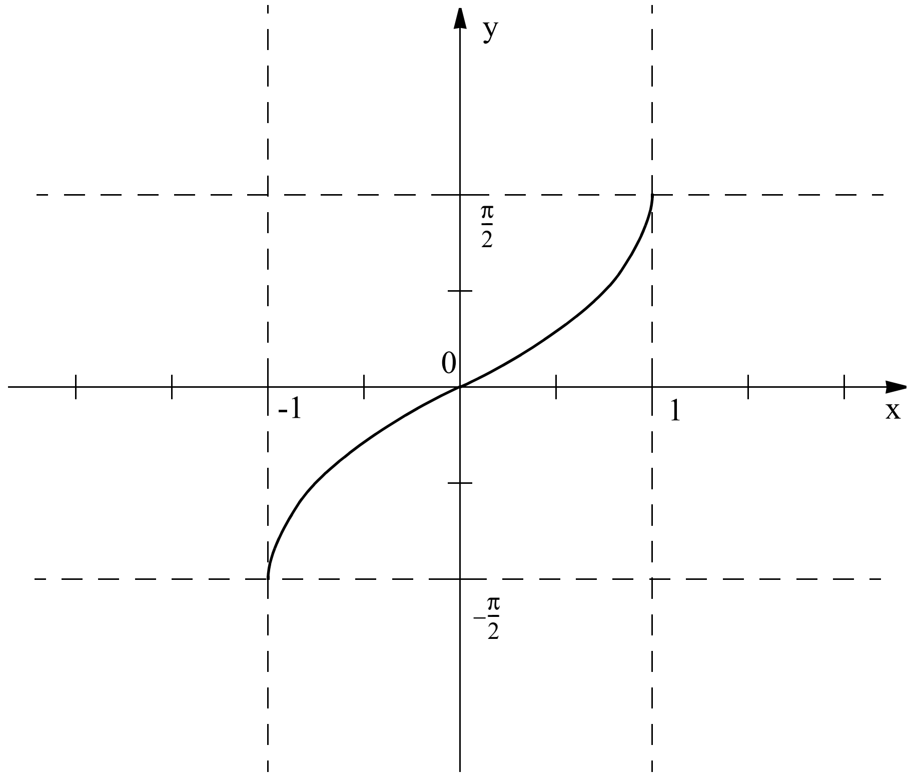

Figure 3-4. GFLIB_Asin Function

3.9.7 Returns

The function returns asin(x)/(pi/2).

3.9.8 Range Issues

The input data value is in the range of <-1,1). The output data value is in the range <-0.5, 0.5), which corresponds to the angle in the range of <-pi/2,pi/2).

3.9.9 Special Issues

The **GFLIB_Asin** function is the saturation mode independent.

3.9.10 Implementation

The **GFLIB_Asin** function is implemented as a function call.

Example 3-8. Implementation Code

```
#include "gflib.h"
```

```
static Frac16 mf16Input;
static Frac16 mf16Output;

void main(void)
{
         /* input value 0.5 */
         mf16Input = FRAC16(0.5);

         /* Compute the arcsin value */
         mf16Output = GFLIB_Asin(mf16Input);
}
```

3.9.11 See Also

See **GFLIB_Acos**, **GFLIB_Atan**, **GFLIB_AtanYX** and **GFLIB_AtanYXShifted** for more information.

3.9.12 Performance

Table 3-22. Performance of GFLIB_Asin function

Code Size (words)	73 + 65 (GFLIB_SqrtPoly)	
Data Size (words)	27+34 (GFLIB_SqrtPoly)	
Execution Clock	Min	86/85 cycles
	Max	185/178 cycles

3.10 GFLIB_Acos

The function calculates the arcus cosine value of the argument using the piece-wise polynomial approximation.

3.10.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_Acos(Frac16 f16Arg)
```

3.10.2 Prototype

```
asm Frac16 GFLIB_AcosFAsm(Frac16 f16Arg,
GFLIB_ASINACOS_COEFFICIENTS_ADDR_T *pudtAcosPoly)
```

3.10.3 Arguments

Table 3-23. Function Arguments

Name	In/Out	Format	Range	Description
f16Arg	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in header file GFLIB_types.h
*pudtAcosPoly	In	N/A	N/A	Optional argument; pointer to data needed for the calculation.

3.10.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.10.5 Dependencies

List of all dependent files:

- GFLIB_AsinAcosAsm.h
- GFLIB_AsinAcosAsmDef.h
- GFLIB_types.h

3.10.6 Description

The GFLIB_Acos function computes the acos(f16Arg)/(pi/2) using the arcus sine calculation as follows:

$$A\cos(x) = ((-0{,}5) + A\sin(x)) \cdot (-1) \qquad \textbf{Eqn. 3-29}$$

The sine calculation uses the piece-wise polynomial approximation (see Figure 3-5).

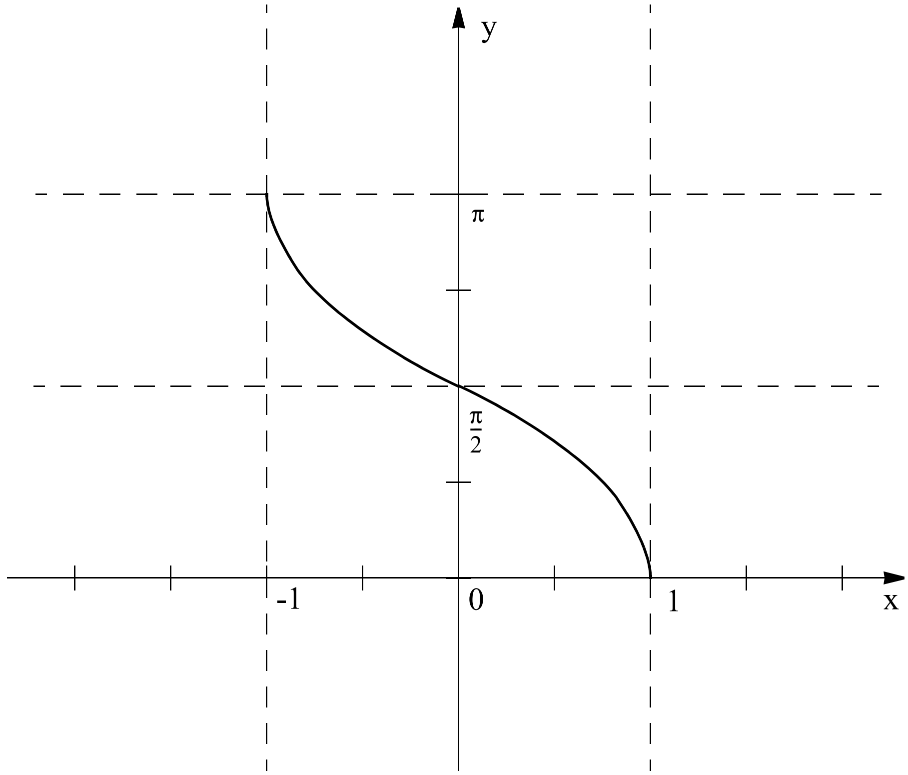

Figure 3-5. GFLIB_Acos Function

3.10.7 Returns

The function returns acos(x)/(pi/2).

3.10.8 Range Issues

The input data value is in the range of <-1,1). The output data value is in the range <0,1), which corresponds to the angle in the range of <0,pi).

3.10.9 Special Issues

The function **GFLIB_Acos** is the saturation mode independent.

The function is very short and to reduce the time consumption of this function call, it may be reasonable to implement it as an inline function.

3.10.10 Implementation

The **GFLIB_Acos** function is implemented as a function call.

Example 3-9. Implementation Code

```
#include "gflib.h"

static Frac16 mf16Input;
static Frac16 mf16Output;

void main(void)
{
        /* input value 0.5 */
        mf16Input = FRAC16(0.5);

        /* Compute the arccos value */
        mf16Output = GFLIB_Acos(mf16Input);

}
```

3.10.11 See Also

See **GFLIB_Asin**, **GFLIB_Atan**, **GFLIB_AtanYX** and **GFLIB_AtanYXShifted** for more information.

3.10.12 Performance

Table 3-24. Performance of GFLIB_Acos function

Code Size (words)	8 + 73 (GFLIB_Asin) + 65 (GFLIB_SqrtPoly)[1]	
Data Size (words)	27 + 34 (GFLIB_SqrtPoly)[1]	
Execution Clock	Min	100/99 cycles
	Max	199/193 cycles

[1] Code size of GFLIB_Acos() function is 8words. GFLIB_Acos() function however uses additional two functions for calculation: GFLIB_Asin() and GFLIB_Sqrt(), which if not already used within the application source code is included into the code together with GFLIB_Acos().

3.11 GFLIB_Atan

The function calculates the arcus tangent value of the argument using the piece-wise polynomial approximation.

3.11.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_Atan(Frac16 f16Arg)
```

3.11.2 Prototype

```
asm Frac16 GFLIB_AtanFAsm(Frac16 f16Arg, GFLIB_ATAN_COEFFICIENTS_ADDR_T
*pudtAtanPoly)
```

3.11.3 Arguments

Table 3-25. Function Arguments

Name	In/Out	Format	Range	Description
f16Arg	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in header file GFLIB_types.h
*pudtAtanPoly	In	N/A	N/A	Optional argument; pointer to data needed for the calculation.

3.11.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.11.5 Dependencies

List of all dependent files:

- GFLIB_AtanAsm.h
- GFLIB_AtanAsmDef.h
- GFLIB_types.h

3.11.6 Description

The **GFLIB_Atan** function computes atan(x)/(pi/4) using the piece-wise polynomial approximation (see Figure 3-6). This function evaluates the polynomial according to the Horner scheme. The input and output values are additionally scaled and centered for the best accuracy.

$$u = (x - x_{offset}) \cdot 2^{k_X}$$ **Eqn. 3-30**

$$v = ((a_n \cdot u + a_{n-1}) \cdot u + a_{n-2}) \cdot u \ldots + a_0$$ **Eqn. 3-31**

$$y = v \cdot 2^{k_Y} + y_{offset}$$ **Eqn. 3-32**

where

x is the input argument (f16Arg)

y is the evaluated value

u, v are the intermediate variables

$a_n, a_{n-1} \ldots a_0$ are the coefficients of the polynomial

k_X is the scaling factor for the input value

x_{offset} is the offset for the input value

k_Y is the scaling factor for the output value

y_{offset} is the offset for the output value

The function input range is divided into three intervals within which all the coefficients are calculated as follows (in the integer format):

The coefficients for the input argument falling into the interval <-0.5 ; 0)

k_X = 2

x_{offset} = -8192

a_4, a_3, a_2, a_1, a_0 = -56, -289, 1154, 19633, -14522

y_{offset} = -24248975L

k_Y = 4

The coefficients for the interval <-1 ; - 0.5)

k_X = 2

x_{offset} = -24576

a_4, a_3, a_2, a_1, a_0 = -37, 145, 3204, 26704, -9087

y_{offset} = -201326592L

k_Y = 5

The coefficients for the interval <0>

k_X = 0

x_{offset} = 0

a_4, a_3, a_2, a_1, a_0 = 0, 0, 0, 0

y_{offset} = 0L

k_Y = 0

The scaling factors and offsets should be set to use the whole available fractional range of <-1, 1).

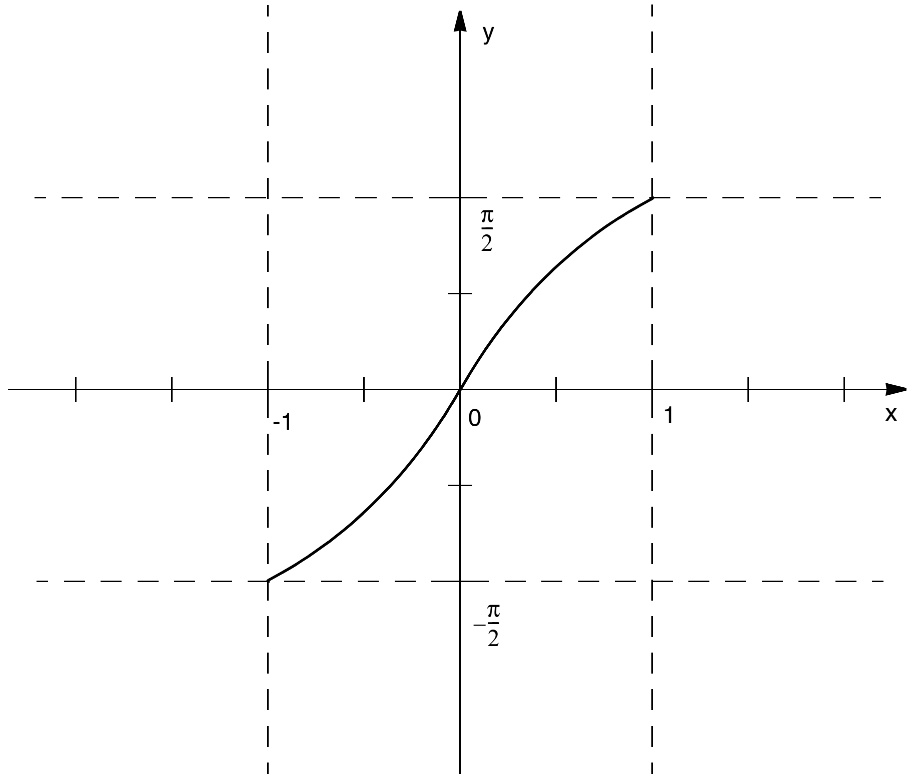

Figure 3-6. GFLIB_Atan Function

3.11.7 Returns

The function returns atan(x)/(pi/4).

3.11.8 Range Issues

The input data value is in the range of <-1,1). The output data value is in the range <-0.25, 0.25), which corresponds to the angle in the range of <-pi/4,pi/4).

3.11.9 Special Issues

The **GFLIB_Atan** function is the saturation mode independent.

3.11.10 Implementation

The **GFLIB_Atan** function is implemented as a function call.

Example 3-10. Implementation Code

```
#include "gflib.h"

static Frac16 mf16Input;
static Frac16 mf16Output;

void main(void)
```

```
{
        /* input value 0.5 */
        mf16Input = FRAC16(0.5);

        /* Compute the arctan value */
        mf16Output = GFLIB_Atan(mf16Input);

}
```

3.11.11 See Also

See **GFLIB_Asin**, **GFLIB_Acos**, **GFLIB_AtanYX** and **GFLIB_AtanYXShifted** for more information.

3.11.12 Performance

Table 3-26. Performance of GFLIB_Atan function

Code Size (words)	44	
Data Size (words)	33	
Execution Clock	Min	61/60 cycles
	Max	61/60 cycles

3.12 GFLIB_AtanYX

The function calculates the arcus tangent value based on the provided x, y co-ordinates as arguments using the division and the piece-wise polynomial approximation

3.12.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_AtanYX(Frac16 f16ValY, Frac16 f16ValX, const Int16
*pi16ErrFlag)
```

3.12.2 Prototype

```
asm Frac16 GFLIB_AtanYXFAsm(Frac16 f16ValY, Frac16 f16ValX, const Int16
*pi16ErrFlag, GFLIB_ATANYX_COEFFICIENTS_ADDR_T *pudtAtanYXPoly)
```

3.12.3 Arguments

Table 3-27. Function Arguments

Name	In/Out	Format	Range	Description
f16ValY	In	SF16	0x8000... 0x7FFF	Y-coordinate input argument; the **Frac16** data type is defined in header file GFLIB_types.h
f16ValX	In	SF16	0x8000... 0x7FFF	X-coordinate input argument; the **Frac16** data type is defined in header file GFLIB_types.h
*pi16ErrFlag	Out	SI16	0...1	Pointer to the error flag. 0 - no error, 1 - both coordinates are zero.
*pudtAtanYXPoly	In	N/A	N/A	Optional argument; pointer to data needed for the calculation.

3.12.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.12.5 Dependencies

List of all dependent files:

- GFLIB_AtanYXAsm.h
- GFLIB_AtanYXAsmDef.h
- GFLIB_types.h

3.12.6 Description

The **GFLIB_AtanYX** function computes the angle where its tangent is y/x. This calculation is based on the input arguments division (y divided by x) and the piece-wise polynomial approximation. This function is the fractional counterpart of the well-known atan2() function defined in the ANSI-C math library.

In comparison to **GFLIB_Atan**, the **GFLIB_AtanYX** function correctly places the calculated angle in the whole <-pi, pi) range, according to the signs of the x and y coordinates provided as the arguments.

If the x and y coordinates are 0 (zero), the function returns 0 and the address pointed by the pi16ErrFlag pointer is set to 1 (otherwise it is set to 0).

3.12.7 Returns

The function returns the angle that has its tangent equal to y/x (arcus tangent from the x, y coordinates).

3.12.8 Range Issues

The input data value is in the range of <-1,1). The output data value is in the range <-1, 1), which corresponds to the angle in the range of <-pi,pi).

3.12.9 Special Issues

The **GFLIB_AtanYX** function is the saturation mode independent.

3.12.10 Implementation

The **GFLIB_AtanYX** function is implemented as a function call.

Example 3-11. Implementation Code

```
#include "gflib.h"

static Frac16 mf16InputY;
static Frac16 mf16InputX;
static Frac16 mf16Output;
static Int16 mi16Flag;

void main(void)
{
        /* x input value 0.5 */
        mf16InputX = FRAC16(0.5);

        /* y input value 1.0 */
        mf16InputY = FRAC16(1.0);

        /* Compute the arctan value */
        mf16Output = GFLIB_AtanYX(mf16InputY, mf16InputX, &mi16Flag);
}
```

3.12.11 See Also

See GFLIB_Asin, GFLIB_Acos, GFLIB_Atan and GFLIB_AtanYXShifted for more information.

3.12.12 Performance

Table 3-28. Performance of GFLIB_AtanYX function

Code Size (words)	100	
Data Size (words)	33	
Execution Clock	Min	46/46 cycles
	Max	122/119 cycles

3.13 GFLIB_AtanYXShifted

The function calculates angle of two sine waves shifted in phase one to each other.

3.13.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_AtanYXShifted(Frac16 f16ValY, Frac16 f16ValX,
GFLIB_ATANYXSHIFTED_T *pudtAtanYXCoeff)
```

3.13.2 Prototype

```
asm Frac16 GFLIB_AtanYXShiftedFAsm(Frac16 f16ValY, Frac16 f16ValX,
GFLIB_ATANYXSHIFTED_T *pudtAtanYXCoeff)
```

3.13.3 Arguments

Table 3-29. Function Arguments

Name	In/Out	Format	Range	Description
f16ValY	In	SF16	0x8000... 0x7FFF	Y input data value, equal to $\sin\theta$
f16ValX	In	SF16	0x8000... 0x7FFF	X input data value, equal to $\sin(\theta + \Delta\theta)$
*pudtAtanYXCoeff	In	N/A	N/A	Input pointer of the struct initialization parameters

Table 3-30. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GFLIB_ATANYXSHIFTED_T	f16Ky	In	SF16	0x8000... 0x7FFF	Multiplication coefficient of the y signal
	f16Kx	In	SF16	0x8000... 0x7FFF	Multiplication coefficient of the x signal
	i16Ny	In	SI16	0x8000... 0x7FFF	Scaling coefficient of the y signal
	i16Nx	In	SI16	0x8000... 0x7FFF	Scaling coefficient of the x signal
	f16ThetaAdj	In	SF16	0x8000... 0x7FFF	Adjusting angle

3.13.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.13.5 Dependencies

List of all dependent files:

1. GFLIB_AtanYXShiftedAsm.h
2. GFLIB_AtanYXShiftedAsmDef.h
3. GFLIB_types.h

3.13.6 Description

The parameters should be initialization before the first call of the **GFLIB_AtanYXShifted** function.

The initialization parameters can be calculated according to the algorithm provided as a Matlab code, see Example 3-12.

The **GFLIB_AtanYXShifted** function computes an angle assuming that the arguments denote the following values:

$$y = \sin\theta$$
$$x = \sin(\theta + \Delta\theta)$$

Eqn. 3-33

where:

x, y signal values provided as the arguments

θ angle to be computed by the function

$\Delta\theta$ phase difference between the x, y signals

If $\Delta\theta = \pi/2$ or $\Delta\theta = -\pi/2$ then the function is similar to the **GFLIB_AtanYX** function, however the **GFLIB_AtanYX** function in this case is more effective.

The function uses the following algorithm for computing the angle:

$$b = \frac{S}{2 \cdot \cos\frac{\Delta\theta}{2}} \cdot (y + x)$$
$$a = \frac{S}{2 \cdot \sin\frac{\Delta\theta}{2}} \cdot (x - y)$$
$$\theta = \text{atan2}(b,a) - \Delta\theta/2 + \theta_{offset}$$

Eqn. 3-34

where:

x, y signal values provided as arguments in Equation 3-33

θ angle to be computed by the function in Equation 3-33

$\Delta\theta$ angle offset

θ_{offset} scaling coefficient to prevent an overflow over 1, S is nearly 1 and $S < 1$

a,b intermediate variables

For the fractional conventions purposes the algorithm is implemented such that addition values are used as shown in the equation Equation 3-35 below:

$$\frac{S}{2 \cdot \cos\frac{\Delta\theta}{2}} = K_y \cdot 2^{N_y}$$
$$\frac{S}{2 \cdot \sin\frac{\Delta\theta}{2}} = K_x \cdot 2^{N_x}$$
$$\theta_{adj} = \Delta\theta/2 - \theta_{offset}$$

Eqn. 3-35

where:

K_y multiplication coefficient of the y signal

N_y scaling coefficient of the y signal

K_x multiplication coefficient of the x signal

N_x scaling coefficient of the x signal

θ_{adj} adjusting angle

The signal values need to be provided as the function arguments. The phase difference $\Delta\theta$ needs to be set through the initialization parameters.

The function initialization parameters can be calculated as shown in the following Matlab code:

Example 3-12. Initialization Parameters Calculation in Matlab

```
function [KY, KX, NY, NX, THETAADJ] = atan2shiftedpar(dthdeg,
thoffsetdeg)
% ATAN2SHIFTEDPAR  calculation of parameters for atan2shifted() function
%
%    [NY, NX, KY, KX, THETAADJ] = atan2shiftedpar(dthdeg, thoffsetdeg)
%
%   dthdeg = phase shift between sine waves in degrees
%   thoffsetdeg = angle 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)
    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)/180;
```

For example if $\Delta\theta = 69{,}33°$, $\theta = 0°$, then:

```
dtheta = 69.33deg, thetaoffset = 0deg
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 = (Frac16) 19919
KX = 0.87905/2^0 = 0.87905201358520957 = (Frac16) 28805
THETAADJ = 0.19259643554687500 = (Frac16)  6311
```

The function requires both signals to have the same amplitude equal to 1.0. If not, an additional error is contributed, see Equation 3-36 and Equation 3-37.

The algorithm may become numerically instable under some conditions. For example, if $\Delta\theta$ approaches 0, then the intermediate variable a cannot be computed due to discontinuity at 0.

Using the theory of the partial derivatives a more precise formula can be derived to express the error of the presented algorithm:

$$\varepsilon_{\theta_C} = \left\lceil \frac{0{,}318}{\sin(\Delta\theta_{mod}/2)} \cdot \varepsilon_{yx} + 0{,}159 \cdot \coth\Delta\theta_{mod} \cdot \varepsilon_A + \left(1 + \frac{0{,}318}{\sin\Delta\theta_{mod}}\right) \cdot \varepsilon_{\Delta\theta_C/2} + 1{,}23 \cdot LSB \right\rceil \quad \text{if } 0 < \Delta\theta_m < \pi/2 \qquad \text{Eqn. 3-36}$$

$$\varepsilon_{\theta_C} = \left\lceil \frac{0{,}318}{\cos(\Delta\theta_{mod}/2)} \cdot \varepsilon_{yx} + 0{,}159 \cdot \coth\Delta\theta_{mod} \cdot \varepsilon_A + \left(1 + \frac{0{,}318}{\sin\Delta\theta_{mod}}\right) \cdot \varepsilon_{\Delta\theta_C/2} + 1{,}23 \cdot LSB \right\rceil \quad \text{if } \pi/2 \le \Delta\theta_m \qquad \text{Eqn. 3-37}$$

where:

- LSB Least Significant Bit, for 16-bits fractional numbers equal to 2^{-15}
- $\lceil\ldots\rceil$ ceiling function (the least integer larger or equal to)
- ε_{θ_C} error of the computed angle expressed in LSB
- ε_A difference in signal amplitudes, absolute value, in LSB

ε_{yx} error contributed by the x, y signals, $\varepsilon_y = \varepsilon_x = \varepsilon_{yx}$ in LSB

$\varepsilon_{\Delta\theta_C/2}$ measurement error of the phase difference between the signals divided by 2 plus the error due to the finite binary resolution

$\Delta\theta_{mod}$ phase difference $\Delta\theta$ modulo π

The signal error ε_{yx} includes an error due to the limited resolution (for example of an ADC converter) and signal distortion by higher order harmonics.

The reference for the derivation of the error can be found at the end of this section.

It should be noticed that the error reaches its minimum at $\Delta\theta$ equal to $\pi/2$ or $(-\pi)/2$ and in this case the function becomes similar to the trivial arcus tangent function. On the other hand the error is approaching the infinity with $\Delta\theta$ approaching 0 or $\Delta\theta\pi$. Therefore the function is able to provide the most accurate results if $\Delta\theta$ does not differ much from $\pi/2$ or $(-\pi)/2$.

Due to the cyclic nature of the angle ($\pi = -\pi$) even a small error may cause the angle shift by $2 \cdot \pi$. For example if the correct angle is π (or 7FFF hex), due to contribution of various errors listed above, the computation may result in the value of $-\pi$ (or 8000 hex). It should be noticed that due to the random nature of errors it is not possible to detect this type of errors by the functions itself. It is recommended to consider if such a behavior may cause any computational problems in the final application.

3.13.6.1 Derivation of Numerical Error

The error of the computed angle consists of:

- error of the supplied signals values due to the finite resolution of the ADC converter: ε_y and ε_x
- error contributed by higher order harmonics appearing in the signal values: ε_y and ε_x (included in the variable)
- computational error of the multiplication due to the finite length of the micro-controller registers: ε_b and ε_a
- error of the angle difference representation $\Delta\theta$ in the finite precision arithmetic: $\varepsilon_{(\Delta\theta_c)/2}$
- error due to the angle difference measurement: (included in the variable)
- error due to differences in signal amplitudes

Using elementary mathematical functions and operators, the computed angle can be expressed as:

$$\theta_C = \frac{1}{\pi} \cdot \operatorname{atan}\left(\tan\left(\frac{\Delta\theta_C}{2}\right) \cdot \frac{S \cdot A_y \cdot \sin(\Delta\theta_A) + S \cdot A_x \cdot \sin(\theta_A + \Delta\theta_A) + \varepsilon_y + \varepsilon_x + \varepsilon_b \cdot 2 \cdot \cos\frac{\Delta\theta_C}{2}}{S \cdot A_x \cdot \sin(\theta_A) - S \cdot A_x \cdot \sin(\theta_A + \Delta\theta_A) + \varepsilon_y - \varepsilon_x + \varepsilon_a \cdot 2 \cdot \sin\frac{\Delta\theta_C}{2}}\right) - \varepsilon_{\Delta\theta_C/2} \qquad \text{Eqn. 3-38}$$

where:

A_y amplitude of the signal y, nearly 1.0

A_x amplitude of the signal x, nearly 1.0

θ_C computed angle

θ_A the actual angle

$\Delta\theta_A$ the actual phase difference between the signals

ε_y error contributed by the signal includes the error contributed by the ADC converter and the higher order harmonics

ε_x error contributed by the signal includes the error contributed by the ADC converter and the higher order harmonics

$\varepsilon_{\Delta\theta_C/2}$ error contributed by the representation of the fractional format (numerical resolution) plus the measurement error of the divided by 2

ε_b error contributed by the multiplication to compute the intermediate variable b, see Equation 3-34

ε_a error contributed by the multiplication to compute the intermediate variable a, see Equation 3-34

The error of the computed angle is equal to:

$$\delta\theta_C = \max_{\theta_A}\left(\left|\sum_u \frac{\partial\theta_A}{\partial u}\cdot\delta u\right|\right)\Bigg|_{\substack{\Delta\theta_C/2\\ \varepsilon_y = 0\\ \varepsilon_x = 0\\ \varepsilon_b = 0\\ \varepsilon_a = 0\\ A_y = 1\\ A_x = 1}}$$

$$u = \Delta\theta_C/2, \varepsilon_y, \varepsilon_x, \varepsilon_b, \varepsilon_a, A_y, A_x$$

Eqn. 3-39

where:

u independent variable

$\max_{\theta_A}$ maximum over the whole range of

δ partial difference operator

$\frac{\partial}{\partial}$ partial derivative operator

Furthermore we can assume:

$$\begin{gathered}\varepsilon_y = \delta\varepsilon_y \ge 0 \\ \varepsilon_x = \delta\varepsilon_x \ge 0 \\ \varepsilon_b = \delta\varepsilon_b \ge 0 \\ \varepsilon_a = \delta\varepsilon_a \ge 0 \\ \varepsilon_{\Delta\theta_C/2} = \delta(\Delta\theta_C/2) \ge 0 \\ \varepsilon_A = |A_y - A_x| \ge 0 \\ \Delta\theta_C \cong \Delta\theta_A = \Delta\theta \\ S \cong 1 \\ \varepsilon_{\theta_C} = \delta\theta_C \\ A_y \cong A_x = A_{yx} \cong 1,\ A_{yx} < 1\end{gathered} \qquad \textbf{Eqn. 3-40}$$

where:

ε_A error contributed by the difference in signal amplitudes

ε_{θ_C} the resulting error of the computed angle

It should be noticed that the error contributed by the signals (ε_y and ε_x), the errors due to the finite precision arithmetic (ε_b, ε_a and $\varepsilon_{\Delta\theta_C/2}$) have a random nature. On contrary, the error due to the unequal signal amplitudes ε_A is a systematic error[1].

Then the maximum error computed from the partial differences is equal to:

$$\theta_C = \frac{1}{\pi}\cdot \max_{\theta_A}\left(\left|\frac{\sin(\theta_A+\Delta\theta)}{A_{yx}\cdot\sin\Delta\theta}\right|\cdot\varepsilon_x + \left|\frac{\sin\theta_A}{A_{yx}\cdot\sin\Delta\theta}\right|\cdot\varepsilon_y + \left|\frac{\cos\left(\theta_A+\frac{\Delta\theta}{2}\right)}{A_{yx}}\right|\cdot\varepsilon_b + \left|\frac{\sin\left(\theta_A+\frac{\Delta\theta}{2}\right)}{A_{yx}}\right|\cdot\varepsilon_a\right. \qquad \textbf{Eqn. 3-41}$$

$$\left. + \left|\frac{\coth\Delta\theta}{2\cdot A_{yx}}\right|\cdot\varepsilon_A + \left|\frac{\sin(2\cdot\theta_A+\Delta\theta)}{\sin\Delta\theta}\right|\cdot\varepsilon_{\Delta\theta_C/2}\right) + \varepsilon_{\Delta\theta_C/2}$$

Apart of the error sources mentioned above, an additional error is contributed by the **GFLIB_AtanYX** function. In case of the **GFLIB_AtanYX** function from the GFLIB the computation error is equal to one LSB.

The errors of the y and x signals, ε_y and ε_x, appear due to the finite resolution of the A/D converter and higher order harmonics and is equal to a certain amount of LSB.

The computation error due to the multiplication in the numerator and the denominator in Equation 3-34, ε_b and ε_a, is equal to the half of LSB (each one).

1. The division into random and systematic errors is conventional, indeed the $\varepsilon_{\Delta\theta_C/2}$ error can be also treated as a systematic error as well as ε_A

The error due to finite, binary representation of $\Delta\theta_C/2$ is equal to the half of LSB. Finally the measurement error of the phase angle which is equal to a certain amount of LSB needs to be considered.

The error due to the signal amplitude difference, e ε_A, is equal to a certain amount of LSB. It should be noticed that the error appears only if the amplitudes of both signals differ.

After the appropriate calculation and substitutions the upper boundary for the error becomes Equation 3-36.

3.13.7 Returns

The function returns an angle of two sine waves shifted in phase.

3.13.8 Range Issues

The input data value is in the range of <-1,1). The output data value is in the range <-1, 1), which corresponds to the angle in the range of <-pi,pi).

The computation error strongly depends on the phase difference between the sine waves which may cause numerical issues.

3.13.9 Special Issues

The function internally calls the **GFLIB_AtanYX** function.

The function is not sensitive to the saturation mode setting. However it should be noticed that the function actually uses the saturation mode bit. The saturation mode bit is stored and restored during a function call.

The function uses REP instruction to perform shifting with the scaling coefficients N_y, N_x used as repetition amounts. Therefore large values of the scaling coefficient may increase the interrupt latency time.

3.13.10 Implementation

The **GFLIB_AtanYXShifted** function is implemented as a function call with an initialization.

Example 3-13. Implementation Code

```
#include "gflib.h"

static Frac16 mf16InputY;
static Frac16 mf16InputX;
static Frac16 mf16OutputZ;

/* Define appropriate data */
/* dtheta = 5deg, thetaoffset = 0deg */
#define NY 0
```

```
#define NX 4
#define KY FRAC16(0.50046106925577549)
#define KX FRAC16(0.71640268727196699)
#define THETAADJ FRAC16(0.01388549804687500)

/* Initialize */
static GFLIB_ATANYXSHIFTED_T mudtMyAtanYX = {NY, NX, KY, KX, THETAADJ};

void main(void)
{
         /* ~= sin(-166.811) */
         mf16InputY = -7477;

         /* ~= sin(-166.811 + 5) */
         mf16InputX = -10229;

         /* Compute angle */
         mf16OutputZ = GFLIB_AtanYXShifted(mf16InputY, mf16InputX,
&mudtMyAtanYX);
}
```

3.13.11 See Also

See **GFLIB_Asin**, **GFLIB_Acos**, **GFLIB_Atan** and **GFLIB_AtanYX** for more information.

3.13.12 Performance

Table 3-31. Performance of GFLIB_AtanYXShifted function

Code Size (words)	52 + 100 (GFLIB_AtanYX)	
Data Size (words)	0 + 33 (GFLIB_AtanYX)	
Execution Clock	Min	105/102 cycles
	Max	185/179 cycles

3.14 GFLIB_SqrtPoly

The function calculates the square root value of the argument using the piece-wise polynomial approximation with the post-adjustment method.

3.14.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_SqrtPoly(Frac32 f32Arg)
```

3.14.2 Prototype

```
asm Frac16 GFLIB_SqrtPolyFAsm(Frac32 f32Arg, const
GFLIB_SQRT_POLY_TABLE_T *pudtPolyTable)
```

3.14.3 Arguments

Table 3-32. Function Arguments

Name	In/Out	Format	Range	Description
f32Arg	In	SF32	0x80000000... 0x7FFFFFFF	Input argument; the Frac32 data type is defined in header file GFLIB_types.h
*pudtPolyTable	In	N/A	N/A	Optional argument; pointer to data needed for the calculation.

3.14.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.14.5 Dependencies

List of all dependent files:

- GFLIB_SqrtAsm.h
- GFLIB_SqrtDefAsm.h
- GFLIB_types.h

3.14.6 Description

The function **GFLIB_SqrtPoly** calculates computes the square root of the input argument using the piece-wise polynomial approximation and the post-adjustment of the least significant bits. The function calculates the square root correctly for the values equal to or larger than zero. For the negative arguments the function may behave unpredictably.

The algorithm calculates the raw result using a polynom of the 4th order with 3 intervals. Then the raw result is iterated in 3 steps to get a more precise result.

3.14.7 Returns

For the argument equal to or larger than zero the function **GFLIB_SqrtPoly** returns the square root of the argument, which is a number which if squared is the best approximation of the argument. The function correctly rounds the least significant bit. For the negative arguments the function returns undefined values.

3.14.8 Range Issues

The input data value is in the range of <0,1), expressed with the 32-bit precision. The output data value is in the range of <0, 1) expressed with the 16-bit precision.

3.14.9 Special Issues

The **GFLIB_SqrtPoly** function is the saturation mode independent.

3.14.10 Implementation

The **GFLIB_SqrtPoly** function is implemented as a function call.

Example 3-14. Implementation Code

```
#include "gflib.h"

static Frac32 mf32Input;
static Frac16 mf16Output;

void main(void)
{
        /* input value 0.5 */
        mf32Input = FRAC32(0.5);

        /* Compute the sine value */
        mf16Output = GFLIB_SqrtPoly(mf32Input);


}
```

3.14.11 See Also

See **GFLIB_SqrtIter** for more information.

3.14.12 Performance

Table 3-33. Performance of GFLIB_SqrtPoly function

Code Size (words)	65	
Data Size (words)	34	
Execution Clock	Min	22/22 cycles
	Max	90/85 cycles

3.15 GFLIB_SqrtIter

The function calculates the square root value of the argument using four iterations.

3.15.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_SqrtIter(Frac32 f32Arg)
```

3.15.2 Prototype

```
asm Frac16 GFLIB_SqrtIterFAsm(Frac32 f32Arg)
```

3.15.3 Arguments

Table 3-34. Function Arguments

Name	In/Out	Format	Range	Description
f32Arg	In	SF32	0x80000000... 0x7FFFFFFF	Input argument; the Frac32 data type is defined in header file GFLIB_types.h

3.15.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.15.5 Dependencies

List of all dependent files:

- GFLIB_SqrtAsm.h
- GFLIB_SqrtDefAsm.h
- GFLIB_types.h

3.15.6 Description

The function **GFLIB_SqrtIter** calculates computes the square root of the input argument using the following steps:

1. The argument is normalized to a range of a 32-bit signed number where the number of shift is saved.
2. The four iterations use this equation:

$$Y_{k+1} = 2 \cdot Y_k \cdot \left(\frac{3}{4} - Y_k^{\ 2} \cdot \frac{X}{4}\right) \qquad \textbf{Eqn. 3-42}$$

3. The final square root is

$$Y = 2 \cdot X \cdot Y_4 \qquad \textbf{Eqn. 3-43}$$

4. As the number was shifted to the left at the beginning, at the end it must be shifted to the right by the half of the shifts. Here arise two cases:
   a) The number of shifts is odd, the 0.5 shift can be represented by the following formula. So the result is multiplied by 0.70711 and then it is shifted to the right by the entire number of half of the shifts.

$$2^{-\frac{1}{2}} = \frac{1}{\sqrt{2}} = 0{,}70711 \qquad \textbf{Eqn. 3-44}$$

   b) The number of shifts is even, so the result is shifted to the right by the half of the shifts.

The algorithm uses the hardware do - loop instructions for the iteration process.

3.15.7 Returns

For the argument equal to or larger than zero the function **GFLIB_SqrtIter** returns the square root of the argument. For the negative arguments the function returns undefined values.

3.15.8 Range Issues

The input data value is in the range of <0,1), expressed with the 32-bit precision. The output data value is in the range of <0, 1) expressed with the 16-bit precision.

3.15.9 Special Issues

The **GFLIB_SqrtIter** function is the saturation mode independent.

3.15.10 Implementation

The **GFLIB_SqrtIter** function is implemented as a function call.

Example 3-15. Implementation Code

```
#include "gflib.h"

static Frac32 mf32Input;
static Frac16 mf16Output;
```

```
void main(void)
{
        /* input value 0.5 */
        mf32Input = FRAC32(0.5);

        /* Compute the sine value */
        mf16Output = GFLIB_SqrtIter(mf32Input);


}
```

3.15.11 See Also

See **GFLIB_SqrtPoly** for more information.

3.15.12 Performance

Table 3-35. Performance of GFLIB_SqrtIter function

Code Size (words)	28	
Data Size (words)	0	
Execution Clock	Min	65 cycles
	Max	65 cycles

3.16 GFLIB_Lut

The function approximates a one-dimensional arbitrary user function using the interpolation lookup method. The user function is stored in the table of size specified in uw16TableSize and pointed to by *pf16Table pointer.

3.16.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_Lut(Frac16 f16Arg, Frac16 *pf16Table, UWord16
uw16TableSize)
```

3.16.2 Prototype

```
asm Frac16 GFLIB_LutFAsm(Frac16 f16Arg, const Frac16 *pf16Table, UWord16
uw16TableSize)
```

3.16.3 Arguments

Table 3-36. Function Arguments

Name	In/Out	Format	Range	Description
f16Arg	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in header file GFLIB_types.h
*pf16Table	In	N/A	N/A	Pointer to the values table
uw16TableSize	In	UI16	0x0... 0xFFFF	The table size in bit shifts of number 1.

3.16.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.16.5 Dependencies

List of all dependent files:

- GFLIB_LutAsm.h
- GFLIB_types.h

3.16.6 Description

The **GFLIB_Lut** fuses a table of the precalculated function points. These points are selected with a fixed step and must be in a number of 2^n, where n can be 1 through to 15.

The function finds two nearest precalculated points of the input argument and using the linear interpolation between these two points calculates the output value.

This algorithm serves for periodical functions i.e. if the input argument lies behind the last precalculated point of the function, the interpolation is calculated between the last and the first point of the table as if it was a periodical function.

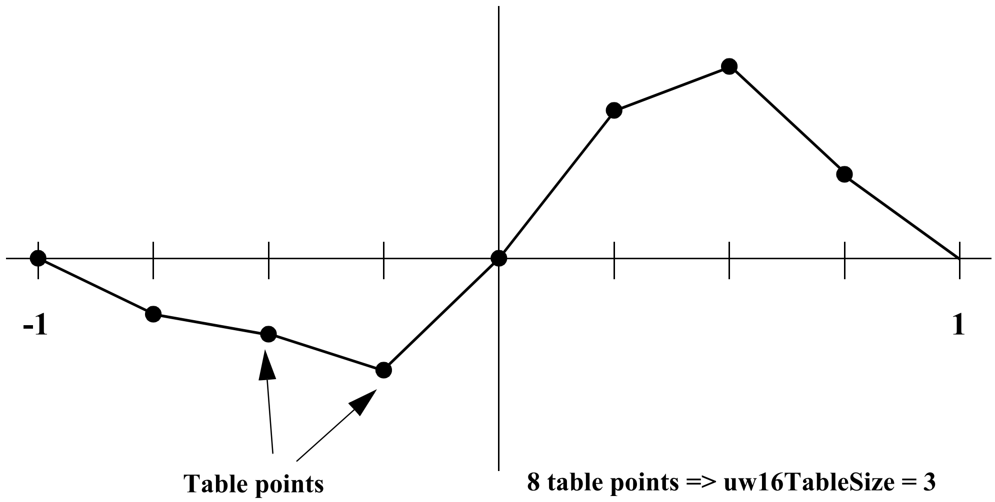

Figure 3-7. Algorithm Diagram

3.16.7 Returns

The function **GFLIB_Lut** returns a value calculated from a linear interpolation between two points according to the input argument.

3.16.8 Range Issues

The input value is in the range of <0,1), expressed with the 16-bit precision. The output data value is in the range of <0, 1) expressed with 16-bits precision. The table size depends on the available memory.

3.16.9 Special Issues

The function **GFLIB_Lut** requires the saturation mode to be set OFF.

3.16.10 Implementation

The **GFLIB_Lut** function is implemented as a function call.

Example 3-16. Implementation Code

```
#include "gflib.h"

static Frac16 mf16Input;
```

```
static Frac16 mf16Output;

/* 8-values table, 2 ^ 3 = 8 */
static UWord16 muw16TableSize = 3;
static Frac16 mudtTableValues[] =
{
        0, -16384, -32768, -16384, 0, 16384, 32767, 16384
};

void main(void)
{
        /* input value 0.6 */
        mf16Input = FRAC16(-0.125);

        /* turns off the saturation */
        __turn_off_sat();

        /* Compute the look-up table algorithm */
        mf16Output = GFLIB_Lut(mf16Input, mudtTableValues,
muw16TableSize);
}
```

3.16.11 Performance

Table 3-37. Performance of GFLIB_Lut function

Code Size (words)	32	
Data Size (words)	0	
Execution Clock	Min	48/48 cycles
	Max	48/48 cycles

3.17 GFLIB_Ramp16

The function calculates a 16-bit version of the up/down ramp with the step increment/decrement defined in the pParam structure.

3.17.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_Ramp16(Frac16 f16Desired, Frac16 f16Actual, const
GFLIB_RAMP16_T *pudtParam)
```

3.17.2 Prototype

```
asm Frac16 GFLIB_Ramp16FAsm(Frac16 f16Desired, Frac16 f16Actual, const
GFLIB_RAMP16_T *pudtParam)
```

3.17.3 Arguments

Table 3-38. Function Arguments

Name	In/Out	Format	Range	Description
f16Desired	In	SF16	0x8000... 0x7FFF	Desired value; the **Frac16** data type is defined in header file GFLIB_types.h
f16Actual	In	SF16	0x8000... 0x7FFF	Actual value; the **Frac16** data type is defined in header file GFLIB_types.h
*pudtParam	In	N/A	N/A	Pointer to structure containing the ramp-up and -down increments

Table 3-39. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GFLIB_RAMP16_T	f16RampUp	In	SF16	0x8000... 0x7FFF	Ramp up increment
	f16RampDown	In	SF16	0x8000... 0x7FFF	Ramp down increment

3.17.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.17.5 Dependencies

List of all dependent files:

- GFLIB_RampAsm.h

- GFLIB_types.h

3.17.6 Description

The **GFLIB_Ramp16** calculates the 16-bit ramp of the actual value by the up or down increments contained in the pudtParam structure.

If the desired value is greater than the actual value, the function adds the ramp-up value to the actual value. The output cannot be greater than the desired value.

If the desired value is lower than the actual value, the function subtracts the ramp-down value from the actual value. The output cannot be lower than the desired value.

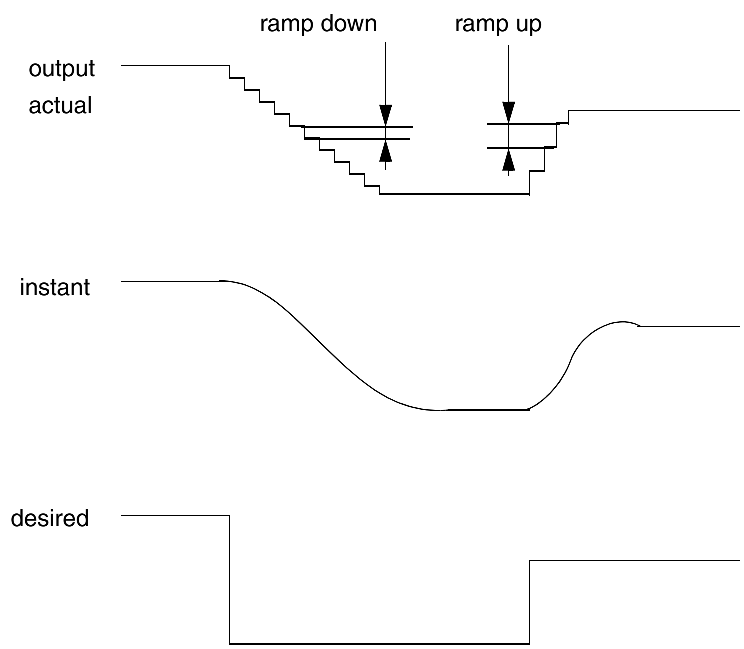

Figure 3-8. Algorithm Diagram

3.17.7 Returns

If f16Desired is greater than f16Actual, the function returns f16Actual + the ramp-up value until f16Desired is reached.

If f16Desired is less than f16Actual, the function returns f16Actual - the ramp-down value until the f16Desired is reached.

3.17.8 Range Issues

The input data value is in the range of <-1,1) and the output data values are in the range <-1,1).

3.17.9 Special Issues

The function **GFLIB_Ramp16** is the saturation mode independent.

3.17.10 Implementation

The **GFLIB_Ramp16** function is implemented as a function call.

Example 3-17. Implementation Code

```
#include "gflib.h"

static Frac16 mf16DesiredValue;
static Frac16 mf16ActualValue;

/* Ramp parameters */
static GFLIB_RAMP16_T mudtRamp16;

void Isr(void);

void main(void)
{
         /* Ramp parameters initialization */
         mudtRamp16.f16RampUp = FRAC16(0.25);
         mudtRamp16.f16RampDown = FRAC16(0.25);

         /* Desired value initialization */
         mf16DesiredValue = FRAC16(1.0);

         /* Actual value initialization */
         mf16ActualValue = 0;
}

/* Periodical function or interrupt */
void Isr(void)
{
         /* Ramp generation */
         mf16ActualValue = GFLIB_Ramp16(mf16DesiredValue,
mf16ActualValue, &mudtRamp16);
}
```

3.17.11 See Also

See **GFLIB_Ramp32**, **GFLIB_DynRamp16** and **GFLIB_DynRamp32** for more information.

3.17.12 Performance

Table 3-40. Performance of GFLIB_Ramp16 function

Code Size (words)	18	
Data Size (words)	0	
Execution Clock	Min	36/37 cycles
	Max	36/37 cycles

3.18 GFLIB_Ramp32

The function calculates a 32-bit version of the up/down ramp with the step increment/decrement defined in the pParam structure.

3.18.1 Synopsis

```
#include "gflib.h"
Frac32 GFLIB_Ramp32(Frac32 f32Desired, Frac32 f32Actual, const
GFLIB_RAMP32_T *pudtParam)
```

3.18.2 Prototype

```
asm Frac32 GFLIB_Ramp32FAsm(Frac32 f32Desired, Frac32 f32Actual, const
GFLIB_RAMP32_T *pudtParam)
```

3.18.3 Arguments

Table 3-41. Function Arguments

Name	In/Out	Format	Range	Description
f32Desired	In	SF32	0x80000000... 0x7FFFFFFF	Desired value; the Frac32 data type is defined in header file GFLIB_types.h
f32Actual	In	SF32	0x80000000... 0x7FFFFFFF	Actual value; the Frac32 data type is defined in header file GFLIB_types.h
*pudtParam	In	N/A	N/A	Pointer to structure containing the ramp-up and -down increments

Table 3-42. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GFLIB_RAMP32_T	f32RampUp	In	SF32	0x80000000... 0x7FFFFFFF	Ramp up increment
	f32RampDown	In	SF32	0x80000000... 0x7FFFFFFF	Ramp down increment

3.18.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.18.5 Dependencies

List of all dependent files:

- GFLIB_RampAsm.h

- GFLIB_types.h

3.18.6 Description

The **GFLIB_Ramp32** calculates the 32-bit ramp of the actual value by the up or down increments contained in the pudtParam structure.

If the desired value is greater than the actual value, the function adds the ramp-up value to the actual value. The output cannot be greater than the desired value.

If the desired value is lower than the actual value, the function subtracts the ramp-down value from the actual value. The output cannot be lower than the desired value.

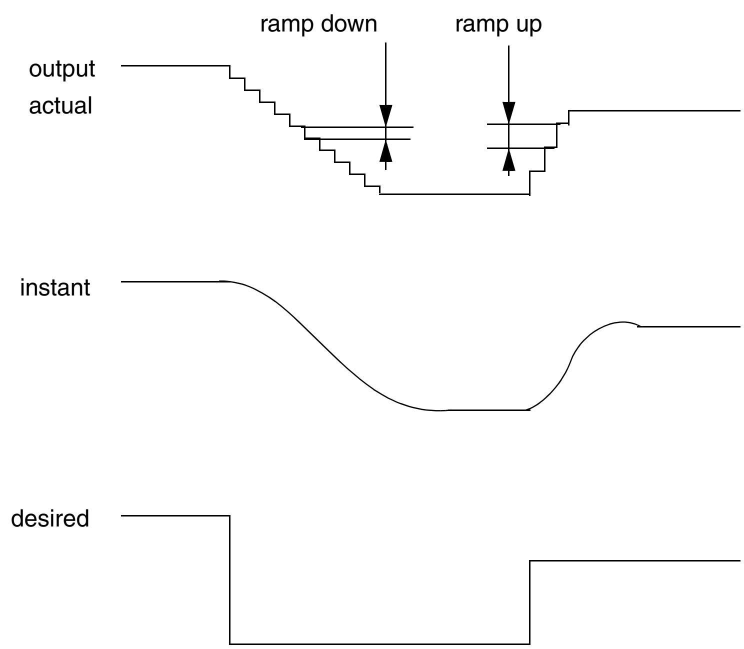

Figure 3-9. Algorithm Diagram

3.18.7 Returns

If f32Desired is greater than f32Actual, the function returns f32Actual + the ramp-up value until the f32Desired is reached.

If f32Desired is less than f32Actual, the function returns f32Actual - the ramp-down value until f32Desired is reached.

3.18.8 Range Issues

The input data value is in the range of <-1,1) in the 32-bit dynamics and the output data values are in the range <-1,1) in the 32-bit dynamics.

3.18.9 Special Issues

The function **GFLIB_Ramp32** is the saturation mode independent.

3.18.10 Implementation

The **GFLIB_Ramp32** function is implemented as a function.

Example 3-18. Implementation Code

```
#include "gflib.h"

static Frac32 mf32DesiredValue;
static Frac32 mf32ActualValue;

/* Ramp parameters */
static GFLIB_RAMP32_T mudtRamp32;

void Isr(void);

void main(void)
{
        /* Ramp parameters initialization */
        mudtRamp32.f32RampUp = FRAC32(0.25);
        mudtRamp32.f32RampDown = FRAC32(0.25);

        /* Desired value initialization */
        mf32DesiredValue = FRAC32(1.0);

        /* Actual value initialization */
        mf32ActualValue = 0;
}

/* Periodical function or interrupt */
void Isr(void)
{
        /* Ramp generation */
        mf32ActualValue = GFLIB_Ramp32(mf32DesiredValue,
mf32ActualValue, &mudtRamp32);
}
```

3.18.11 See Also

See **GFLIB_Ramp16**, **GFLIB_DynRamp16** and **GFLIB_DynRamp32** for more information.

3.18.12 Performance

Table 3-43. Performance of GFLIB_Ramp32 function

Code Size (words)	24	
Data Size (words)	0	
Execution Clock	Min	49/42 cycles
	Max	49/42 cycles

3.19 GFLIB_DynRamp16

This calculates a 16-bit version of the ramp with a different set of up/down parameters depending on the state of uw16SatFlag. If uw16SatFlag is set, the ramp counts up/down towards the f16Instant value.

3.19.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_DynRamp16(Frac16 f16Desired, Frac16 f16Instant, UWord16
uw16SatFlag, GFLIB_DYNRAMP16_T *pudtParam)
```

3.19.2 Prototype

```
asm Frac16 GFLIB_DynRamp16FAsm(Frac16 f16Desired, Frac16 f16Instant,
UWord16 uw16SatFlag, GFLIB_DYNRAMP16_T *pudtParam)
```

3.19.3 Arguments

Table 3-44. Function Arguments

Name	In/Out	Format	Range	Description
f16Desired	In	SF16	0x8000... 0x7FFF	Desired value; the Frac16 data type is defined in header file GFLIB_types.h
f16Instant	In	SF16	0x8000... 0x7FFF	Instant value; the Frac16 data type is defined in header file GFLIB_types.h
uw16SatFlag	In	UI16	0x0... 0xFFFF	Saturation flag
*pudtParam	In	N/A	N/A	Pointer to structure containing the ramp-up and -down increments, saturation ramp-up and -down increments and the last actual values

Table 3-45. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GFLIB_DYNRAMP16_T	f16RampUp	In	SF16	0x8000... 0x7FFF	Ramp up increment
	f16RampDown	In	SF16	0x8000... 0x7FFF	Ramp down increment
	f16RampUpSat	In	SF16	0x8000... 0x7FFF	Ramp up increment when saturation
	f16RampDownSat	In	SF16	0x8000... 0x7FFF	Ramp down increment when saturation
	f16Actual	In/Out	SF16	0x8000... 0x7FFF	Actual value

3.19.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.19.5 Dependencies

List of all dependent files:

- GFLIB_DynRampAsm.h
- GFLIB_types.h

3.19.6 Description

If the saturation flag is zero, the **GFLIB_DynRamp16** function calculates the 16-bit ramp of the actual value (which is contained in the pParam structure) toward the desired value by the up or down increments contained in the pudtParam structure. If the saturation flag is non-zero, the function calculates the ramp toward the instant value using the saturation up or down increments contained in the pudtParam structure.

If the desired value is greater than the actual value, the function adds the ramp-up value to the actual value. The output cannot be greater than the desired value (saturation flag is zero) nor the instant value (saturation flag is non-zero).

If the desired value is lower than the actual value, the function subtracts the ramp-down value from the actual value. The output cannot be lower than the desired value (saturation flag is zero) nor the instant value (saturation flag is non-zero).

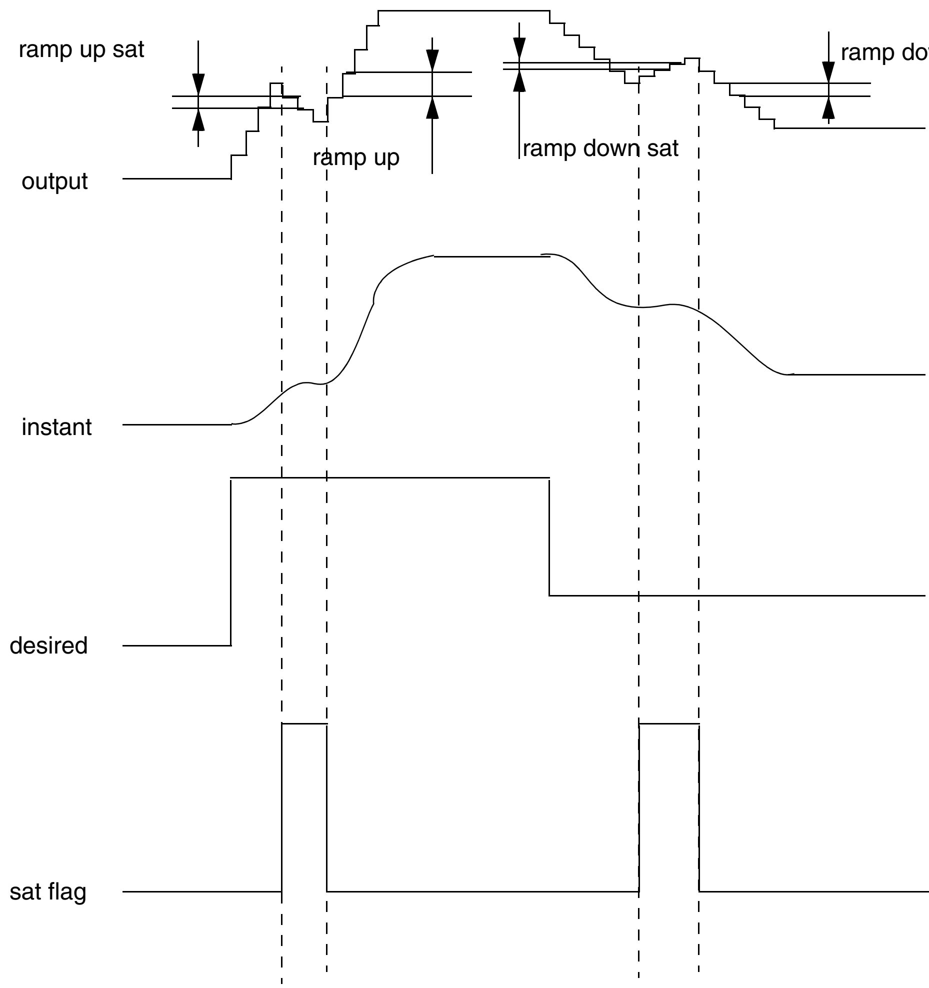

Figure 3-10. Algorithm Diagram

3.19.7 Returns

The **GFLIB_DynRamp16** function returns the 16-bit ramp value as described in Section 3.19.6, "Description".

3.19.8 Range Issues

The input data value is in the range of <-1,1) and the output data values are in the range <-1,1).

3.19.9 Special Issues

The **GFLIB_DynRamp16** function calculation is correct regardless of saturation mode.

3.19.10 Implementation

The **GFLIB_DynRamp16** function is implemented as a function call.

Example 3-19. Implementation Code

```
#include "gflib.h"

static Frac16 mf16DesiredValue;
static Frac16 mf16InstantValue;
static UWord16 muw16SatFlag;
static Frac16 mf16RampOutput;


/* Ramp parameters */
static GFLIB_DYNRAMP16_T mudtDynRamp16;

void Isr(void);

void main(void)
{
        /* Ramp parameters initialization */
        mudtDynRamp16.f16RampUp = FRAC16(0.25);
        mudtDynRamp16.f16RampDown = FRAC16(0.25);
        mudtDynRamp16.f16RampUpSat = FRAC16(0.125);
        mudtDynRamp16.f16RampDownSat = FRAC16(0.125);

        /* Desired value initialization */
        mf16DesiredValue = FRAC16(1.0);

        /* Instant value initialization */
        mf16InstantValue = 0;

        /* Actual value initialization */
        mudtDynRamp16.f16Actual = mf16InstantValue;

        /* Saturation flag initialization */
        muw16SatFlag = 0;

}

/* Periodical function or interrupt */
void Isr(void)
{
        /* Ramp generation */
        mf16RampOutput = GFLIB_DynRamp16(mf16DesiredValue,
mf16InstantValue, muw16SatFlag, &mudtDynRamp16);
}
```

3.19.11 See Also

See **GFLIB_Ramp16**, **GFLIB_Ramp32** and **GFLIB_DynRamp32** for more information.

3.19.12 Performance

Table 3-46. Performance of GFLIB_DynRamp16 function

Code Size (words)	31	
Data Size (words)	0	
Execution Clock	Min	49/50 cycles
	Max	49/50 cycles

3.20 GFLIB_DynRamp32

The function calculates a 32-bit version of the ramp with a different set of the up/down parameters depending on the state of uw16SatFlag. If uw16SatFlag is set, the ramp counts up/down towards the f32Instant value.

3.20.1 Synopsis

```
#include "gflib.h"
Frac32 GFLIB_DynRamp32(Frac32 f32Desired, Frac32 f32Instant, UWord16
uw16SatFlag, GFLIB_DYNRAMP32_T *pudtParam)
```

3.20.2 Prototype

```
asm Frac32 GFLIB_DynRamp32FAsm(Frac32 f32Desired, Frac32 f32Instant,
UWord16 uw16SatFlag, GFLIB_DYNRAMP32_T *pudtParam)
```

3.20.3 Arguments

Table 3-47. Function Arguments

Name	In/Out	Format	Range	Description
f32Desired	In	SF32	0x80000000... 0x7FFFFFFF	Desired value; the Frac32 data type is defined in header file GFLIB_types.h
f32Instant	In	SF32	0x80000000... 0x7FFFFFFF	Instant value (measured or reported value); the Frac32 data type is defined in header file GFLIB_types.h
uw16SatFlag	In	UI16	0x0... 0xFFFF	Saturation flag
*pudtParam	In	N/A	N/A	Pointer to structure containing the ramp-up and -down increments, saturation ramp-up and -down increments and the last actual values

Table 3-48. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GFLIB_DYNRAMP32_T	f32RampUp	In	SF32	0x80000000... 0x7FFFFFFF	Ramp up increment
	f32RampDown	In	SF32	0x80000000... 0x7FFFFFFF	Ramp down increment
	f32RampUpSat	In	SF32	0x80000000... 0x7FFFFFFF	Ramp up increment when saturation
	f32RampDownSat	In	SF32	0x80000000... 0x7FFFFFFF	Ramp down increment when saturation
	f32Actual	In/Out	SF32	0x80000000... 0x7FFFFFFF	Actual value

3.20.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.20.5 Dependencies

List of all dependent files:

- GFLIB_DynRampAsm.h
- GFLIB_types.h

3.20.6 Description

If the saturation flag is zero, the **GFLIB_DynRamp32** function calculates the 32-bit ramp of the actual value (which is contained in the pParam structure) toward the desired value by the up or down increments contained in the pudtParam structure. If the saturation flag is non-zero, the function calculates the ramp toward the instant value using the saturation up or down increments contained in the pudtParam structure.

If the desired value is greater than the actual value, the function adds the ramp-up value to the actual value. The output cannot be greater than the desired value (saturation flag is zero) nor the instant value (saturation flag is non-zero).

If the desired value is lower than the actual value, the function subtracts the ramp-down value from the actual value. The output cannot be lower than the desired value (saturation flag is zero) nor the instant value (saturation flag is non-zero).

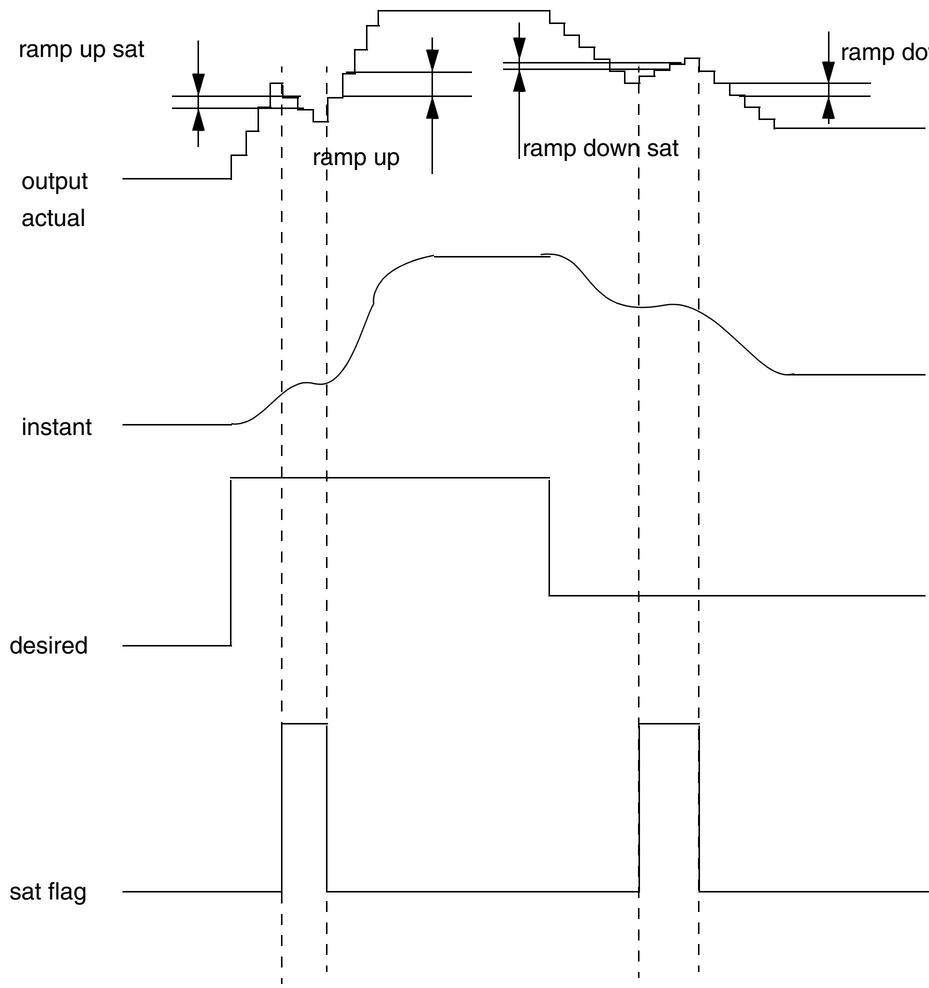

Figure 3-11. Algorithm Diagram

3.20.7 Returns

The **GFLIB_DynRamp32** function returns a 32-bit value.

3.20.8 Range Issues

The input data value is in the range of <-1,1) and the output data values are in the range <-1,1).

3.20.9 Special Issues

The **GFLIB_DynRamp32** function calculation is correct regardless of saturation mode.

3.20.10 Implementation

The **GFLIB_DynRamp32** function is implemented as a function call.

Example 3-20. Implementation Code

```
#include "gflib.h"

static Frac32 mf32DesiredValue;
static Frac32 mf32InstantValue;
static UWord16 muw16SatFlag;
static Frac32 mf32RampOutput;


/* Ramp parameters */
static GFLIB_DYNRAMP32_T mudtDynRamp32;

void Isr(void);

void main(void)
{
        /* Ramp parameters initialization */
        mudtDynRamp32.f32RampUp = FRAC32(0.25);
        mudtDynRamp32.f32RampDown = FRAC32(0.25);
        mudtDynRamp32.f32RampUpSat = FRAC32(0.125);
        mudtDynRamp32.f32RampDownSat = FRAC32(0.125);

        /* Desired value initialization */
        mf32DesiredValue = FRAC32(1.0);

        /* Instant value initialization */
        mf32InstantValue = 0;

        /* Actual value initialization */
        mudtDynRamp32.f32Actual = mf32InstantValue;

        /* Saturation flag initialization */
        muw16SatFlag = 0;

}

/* Periodical function or interrupt */
void Isr(void)
{
        /* Ramp generation */
        mf32RampOutput = GFLIB_DynRamp32(mf32DesiredValue,
mf32InstantValue, muw16SatFlag, &mudtDynRamp32);
}
```

3.20.11 See Also

See **GFLIB_Ramp16**, **GFLIB_Ramp32** and **GFLIB_DynRamp16** for more information.

3.20.12 Performance

Table 3-49. Performance of GFLIB_DynRamp32 function

Code Size (words)	35	
Data Size (words)	0	
Execution Clock	Min	60/51 cycles
	Max	60/51 cycles

3.21 GFLIB_Limit16

The function calculates the 16-bit scalar upper/lower limitation of the input signal.

3.21.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_Limit16(Frac16 f16Arg, const GFLIB_LIMIT16_T *pudtLimit)
```

3.21.2 Prototype

```
asm Frac16 GFLIB_Limit16FAsm(Frac16 f16Arg, const GFLIB_LIMIT16_T
*pudtLimit)
```

3.21.3 Arguments

Table 3-50. Function Arguments

Name	In/Out	Format	Range	Description
f16Arg	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in header file GFLIB_types.h
*pudtLimit	In	N/A	N/A	Pointer to structure containing the upper and lower limits

Table 3-51. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GFLIB_LIMIT16_T	f16UpperLimit	In	SF16	0x8000... 0x7FFF	Upper limit
	f16LowerLimit	In	SF16	0x8000... 0x7FFF	Lower limit

3.21.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.21.5 Dependencies

List of all dependent files:

- GFLIB_LimitAsm.h
- GFLIB_types.h

3.21.6 Description

The **GFLIB_Limit16** function returns the trimmed number according to the 16-bit upper and lower limits.

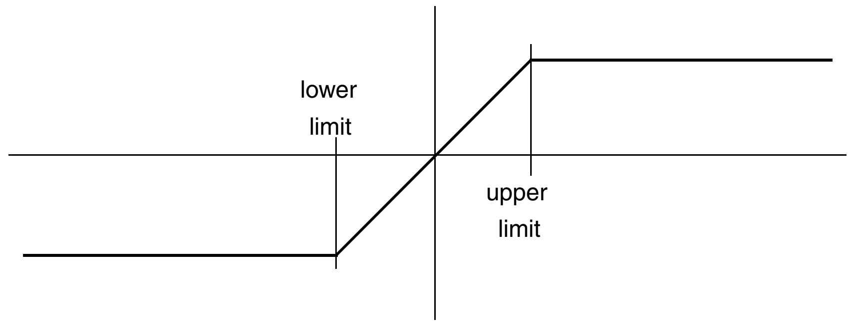

Figure 3-12. Algorithm Transition

3.21.7 Returns

The **GFLIB_Limit16** function returns the trimmed number according to the 16-bit upper and lower limits.

3.21.8 Range Issues

The input data value is in the range of <-1,1) and the output data values are in the range <f16UpperLimit, f16LowerLimit>.

3.21.9 Special Issues

The **GFLIB_Limit16** function calculation is correct regardless of saturation mode.

3.21.10 Implementation

The **GFLIB_Limit16** function is implemented as a function call.

Example 3-21. Implementation Code

```
#include "gflib.h"

static Frac16 mf16InputValue;
static Frac16 mf16OutputValue;

/* Trim parameters */
static GFLIB_LIMIT16_T mudtLimit16;

void main(void)
{
         /* Limit parameters initialization */
         mudtLimit16.f16UpperLimit = FRAC16(0.5);
         mudtLimit16.f16LowerLimit = FRAC16(-0.5);
```

```
        /* Desired value initialization */
        mf16InputValue = FRAC16(0.6);

        /* Limitation */
        mf16OutputValue = GFLIB_Limit16(mf16InputValue, &mudtLimit16);
}
```

3.21.11 See Also

See **GFLIB_Limit32**, **GFLIB_LowerLimit16**, **GFLIB_LowerLimit32**, **GFLIB_UpperLimit16** and **GFLIB_UpperLimit32** for more information.

3.21.12 Performance

Table 3-52. Performance of GFLIB_Limit16 function

Code Size (words)	9	
Data Size (words)	0	
Execution Clock	Min	25/25 cycles
	Max	25/25 cycles

3.22 GFLIB_Limit32

The function calculates the 32-bit scalar upper/lower limitation of the input signal.

3.22.1 Synopsis

```
#include "gflib.h"
Frac32 GFLIB_Limit32(Frac32 f32Arg, const GFLIB_LIMIT32_T *pudtLimit)
```

3.22.2 Prototype

```
asm Frac32 GFLIB_Limit32FAsm(Frac32 f32Arg, const GFLIB_LIMIT32_T
*pudtLimit)
```

3.22.3 Arguments

Table 3-53. Function Arguments

Name	In/Out	Format	Range	Description
f32Arg	In	SF32	0x80000000... 0x7FFFFFFF	Input argument; the Frac32 data type is defined in header file GFLIB_types.h
*pudtLimit	In	N/A	N/A	Pointer to structure containing the upper and lower limits

Table 3-54. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GFLIB_LIMIT32_T	f32UpperLimit	In	SF32	0x80000000... 0x7FFFFFFF	Ramp up increment
	f32LowerLimit	In	SF32	0x80000000... 0x7FFFFFFF	Ramp down increment

3.22.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.22.5 Dependencies

List of all dependent files:

- GFLIB_LimitAsm.h
- GFLIB_types.h

3.22.6 Description

The **GFLIB_Limit32** function returns the trimmed number according to the 32-bit upper and lower limits.

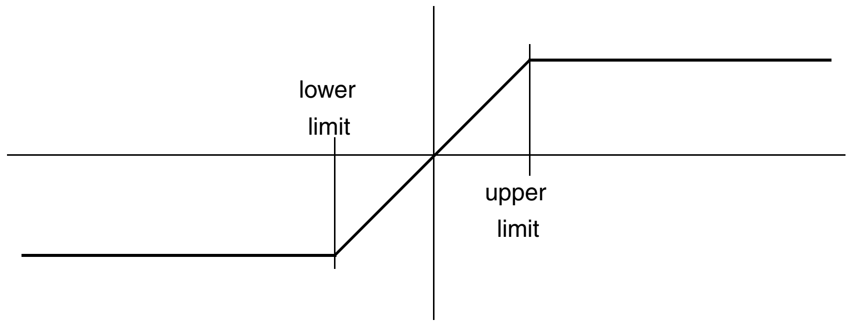

Figure 3-13. Algorithm Transition

3.22.7 Returns

The **GFLIB_Limit32** function returns the trimmed number according to the 32-bit upper and lower limits.

3.22.8 Range Issues

The input data value is in the range of <-1,1) and the output data values are in the range <f32UpperLimit, f32LowerLimit>

3.22.9 Special Issues

The **GFLIB_Limit32** function calculation is correct regardless of saturation mode.

3.22.10 Implementation

The **GFLIB_Limit32** function is implemented as a function call.

Example 3-22. Implementation Code

```
#include "gflib.h"

static Frac32 mf32InputValue;
static Frac32 mf32OutputValue;

/* Trim parameters */
static GFLIB_LIMIT32_T mudtLimit32;

void main(void)
{
        /* Limit parameters initialization */
        mudtLimit32.f32UpperLimit = FRAC32(0.5);
        mudtLimit32.f32LowerLimit = FRAC32(-0.5);
```

```
        /* Desired value initialization */
        mf32InputValue = FRAC32(0.6);

        /* Limitation */
        mf32OutputValue = GFLIB_Limit32(mf32InputValue, &mudtLimit32);
}
```

3.22.11 See Also

See **GFLIB_Limit16**, **GFLIB_LowerLimit16**, **GFLIB_LowerLimit32**, **GFLIB_UpperLimit16** and **GFLIB_UpperLimit32** for more information.

3.22.12 Performance

Table 3-55. Performance of GFLIB_Limit32 function

Code Size (words)	7	
Data Size (words)	0	
Execution Clock	Min	26/24 cycles
	Max	26/24 cycles

3.23 GFLIB_LowerLimit16

The function calculates the 16-bit scalar lower limitation of the input signal.

3.23.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_LowerLimit16(Frac16 f16Arg, Frac16 f16LowerLimit)
```

3.23.2 Prototype

```
asm Frac16 GFLIB_LowerLimit16FAsm(Frac16 f16Arg, Frac16 f16LowerLimit)
```

3.23.3 Arguments

Table 3-56. Function Arguments

Name	In/Out	Format	Range	Description
f16Arg	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in header file GFLIB_types.h
f16LowerLimit	In	SF16	0x8000... 0x7FFF	Lower limit trim; the Frac16 data type is defined in header file GFLIB_types.h

3.23.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.23.5 Dependencies

List of all dependent files:

- GFLIB_LimitAsm.h
- GFLIB_types.h

3.23.6 Description

The **GFLIB_LowerLimit16** function returns the trimmed number according to the 16-bit lower limit. The functionality can be explained with use of Figure 3-14.

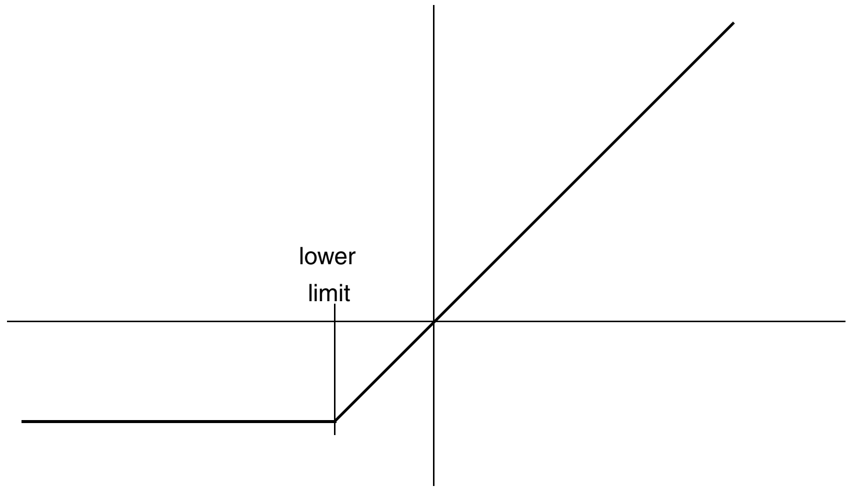

Figure 3-14. Algorithm Transition

3.23.7 Returns

The **GFLIB_LowerLimit16** function returns the trimmed number according to the 16-bit lower limit.

3.23.8 Range Issues

The input data value is in the range of <-1,1) and the output data values are in the range <f16LowerLimit, 1).

3.23.9 Special Issues

The **GFLIB_LowerLimit16** function calculation is correct regardless of saturation mode.

3.23.10 Implementation

The **GFLIB_LowerLimit16** function is implemented as a function.

Example 3-23. Implementation Code

```
#include "gflib.h"

static Frac16 mf16InputValue;
static Frac16 mf16OutputValue;

/* Trim parameter */
static Frac16 mf16TrimValue;

void main(void)
{
        /* Limit parameter initialization */
        mf16TrimValue = FRAC16(-0.5);
```

```
        /* Desired value initialization */
        mf16InputValue = FRAC16(-0.6);

        /* Limitation */
        mf16OutputValue = GFLIB_LowerLimit16(mf16InputValue,
mf16TrimValue);
}
```

3.23.11 See Also

See **GFLIB_Limit16**, **GFLIB_Limit32**, **GFLIB_LowerLimit32**, **GFLIB_UpperLimit16** and **GFLIB_UpperLimit32** for more information.

3.23.12 Performance

Table 3-57. Performance of GFLIB_LowerLimit16 function

Code Size (words)	5	
Data Size (words)	0	
Execution Clock	Min	18/19 cycles
	Max	18/19 cycles

3.24 GFLIB_LowerLimit32

The function calculates the 32-bit scalar lower limitation of the input signal.

3.24.1 Synopsis

```
#include "gflib.h"
Frac32 GFLIB_LowerLimit32(Frac32 f32Arg, Frac32 f32LowerLimit)
```

3.24.2 Prototype

```
asm Frac32 GFLIB_LowerLimit32FAsm(Frac32 f32Arg, Frac32 f32LowerLimit)
```

3.24.3 Arguments

Table 3-58. Function Arguments

Name	In/Out	Format	Range	Description
f32Arg	In	SF32	0x80000000... 0x7FFFFFFF	Input argument; the Frac32 data type is defined in header file GFLIB_types.h
f32LowerLimit	In	SF32	0x80000000... 0x7FFFFFFF	Lower limit trim; the Frac32 data type is defined in header file GFLIB_types.h

3.24.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.24.5 Dependencies

List of all dependent files:

- GFLIB_LimitAsm.h
- GFLIB_types.h

3.24.6 Description

The **GFLIB_LowerLimit32** function returns the trimmed number according to the 32-bit lower limit. The functionality can be explained with use of Figure 3-15.

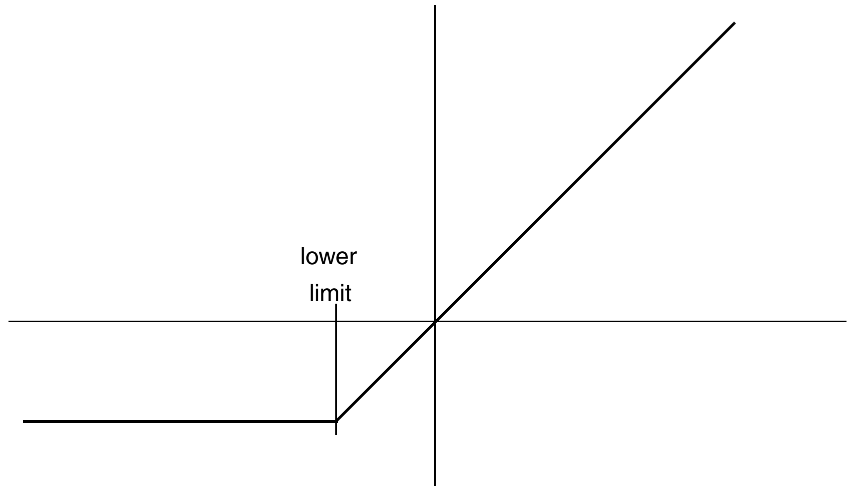

Figure 3-15. Algorithm Transition

3.24.7 Returns

The **GFLIB_LowerLimit32** function returns the trimmed number according to the 32-bit lower limit.

3.24.8 Range Issues

The input data value is in the range of <-1,1) and the output data values are in the range <f32LowerLimit, 1).

3.24.9 Special Issues

The **GFLIB_LowerLimit32** function calculation is correct regardless of saturation mode.

3.24.10 Implementation

The **GFLIB_LowerLimit32** function is implemented as a function call.

Example 3-24. Implementation Code

```
#include "gflib.h"

static Frac32 mf32InputValue;
static Frac32 mf32OutputValue;

/* Trim parameter */
static Frac32 mf32TrimValue;

void main(void)
{
        /* Limit parameter initialization */
        mf32TrimValue = FRAC32(-0.5);
```

```
        /* Desired value initialization */
        mf32InputValue = FRAC32(-0.6);

        /* Limitation */
        mf32OutputValue = GFLIB_LowerLimit32(mf32InputValue,
mf32TrimValue);
}
```

3.24.11 See Also

See **GFLIB_Limit16**, **GFLIB_Limit32**, **GFLIB_LowerLimit16**, **GFLIB_UpperLimit16** and **GFLIB_UpperLimit32** for more information.

3.24.12 Performance

Table 3-59. Performance of GFLIB_LowerLimit32 function

Code Size (words)	3	
Data Size (words)	0	
Execution Clock	Min	17/17 cycles
	Max	17/17 cycles

3.25 GFLIB_UpperLimit16

The function calculates the 16-bit scalar upper limitation of the input signal.

3.25.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_UpperLimit16(Frac16 f16Arg, Frac16 f16UpperLimit)
```

3.25.2 Prototype

```
asm Frac16 GFLIB_UpperLimit16FAsm(Frac16 f16Arg, Frac16 f16UpperLimit)
```

3.25.3 Arguments

Table 3-60. Function Arguments

Name	In/Out	Format	Range	Description
f16Arg	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in header file GFLIB_types.h
f16UpperLimit	In	SF16	0x8000... 0x7FFF	Upper limit trim; the Frac16 data type is defined in header file GFLIB_types.h

3.25.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.25.5 Dependencies

List of all dependent files:

- GFLIB_LimitAsm.h
- GFLIB_types.h

3.25.6 Description

The **GFLIB_UpperLimit16** function returns the trimmed number according to the 16-bit upper limit. The functionality can be explained with use of Figure 3-16.

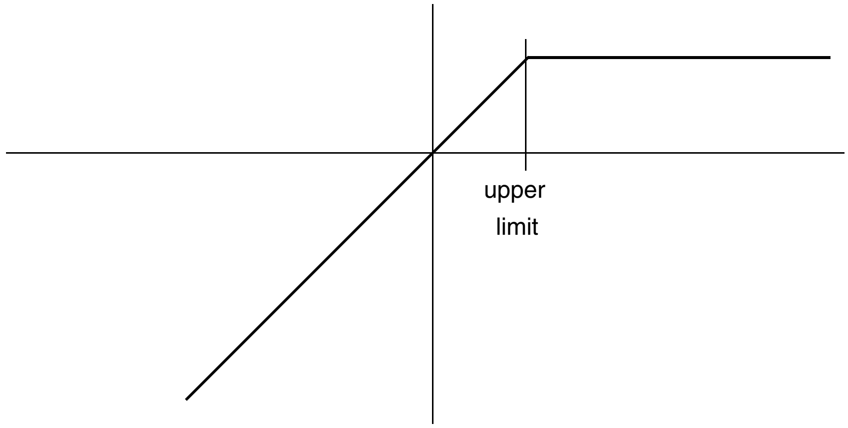

Figure 3-16. Algorithm Transition

3.25.7 Returns

The **GFLIB_UpperLimit16** function returns the trimmed number according to the 16-bit upper limit.

3.25.8 Range Issues

The input data value is in the range of <-1,1) and the output data values are in the range <-1, f16UpperLimit>.

3.25.9 Special Issues

The **GFLIB_UpperLimit16** function calculation is correct regardless of saturation mode.

3.25.10 Implementation

The **GFLIB_UpperLimit16** function is implemented as a function call.

Example 3-25. Implementation Code

```
#include "gflib.h"

static Frac16 mf16InputValue;
static Frac16 mf16OutputValue;

/* Trim parameter */
static Frac16 mf16TrimValue;

void main(void)
{
        /* Limit parameter initialization */
        mf16TrimValue = FRAC16(0.5);

        /* Desired value initialization */
        mf16InputValue = FRAC16(0.6);
```

```
        /* Limitation */
        mf16OutputValue = GFLIB_UpperLimit16(mf16InputValue,
mf16TrimValue);
}
```

3.25.11 See Also

See **GFLIB_Limit16**, **GFLIB_Limit32**, **GFLIB_LowerLimit16**, **GFLIB_LowerLimit32** and **GFLIB_UpperLimit32** for more information.

3.25.12 Performance

Table 3-61. Performance of GFLIB_UpperLimit16 function

Code Size (words)	5	
Data Size (words)	0	
Execution Clock	Min	18/18 cycles
	Max	18/18 cycles

3.26 GFLIB_UpperLimit32

The function calculates the 32-bit scalar upper limitation of the input signal.

3.26.1 Synopsis

```
#include "gflib.h"
Frac32 GFLIB_UpperLimit32(Frac32 f32Arg, Frac32 f32UpperLimit)
```

3.26.2 Prototype

```
asm Frac32 GFLIB_UpperLimit32FAsm(Frac32 f32Arg, Frac32 f32UpperLimit)
```

3.26.3 Arguments

Table 3-62. Function Arguments

Name	In/Out	Format	Range	Description
f32Arg	In	SF32	0x80000000... 0x7FFFFFFF	Input argument; the Frac32 data type is defined in header file GFLIB_types.h
f32UpperLimit	In	SF32	0x80000000... 0x7FFFFFFF	Lower limit trim; the Frac32 data type is defined in header file GFLIB_types.h

3.26.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.26.5 Dependencies

List of all dependent files:

- GFLIB_LimitAsm.h
- GFLIB_types.h

3.26.6 Description

The **GFLIB_UpperLimit32** function returns the trimmed number according to the 32-bit upper limit. The functionality can be explained with use of Figure 3-17

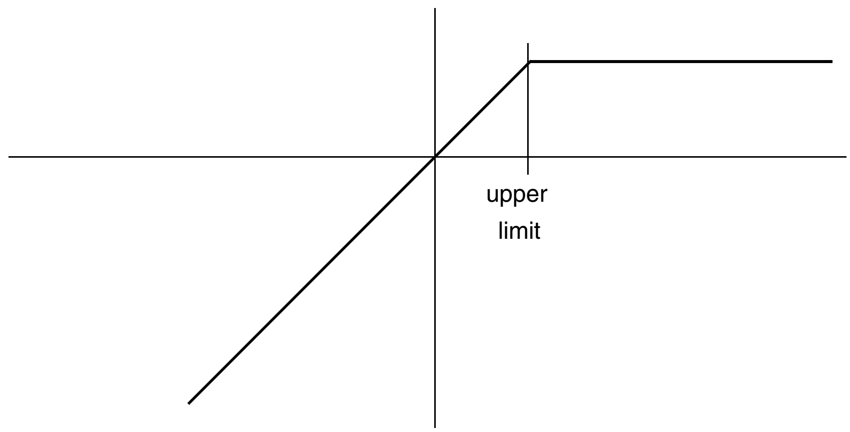

Figure 3-17. Algorithm Transition

3.26.7 Returns

The **GFLIB_UpperLimit32** function returns the trimmed number according to the 32-bit upper limit.

3.26.8 Range Issues

The input data value is in the range of <-1,1) and the output data values are in the range <-1, f32UpperLimit>.

3.26.9 Special Issues

The **GFLIB_UpperLimit32** function calculation is correct regardless of saturation mode.

3.26.10 Implementation

The **GFLIB_UpperLimit32** function is implemented as a function call.

Example 3-26. Implementation Code

```
#include "gflib.h"

static Frac32 mf32InputValue;
static Frac32 mf32OutputValue;

/* Trim parameter */
static Frac32 mf32TrimValue;

void main(void)
{
        /* Limit parameter initialization */
        mf32TrimValue = FRAC32(0.5);

        /* Desired value initialization */
        mf32InputValue = FRAC32(0.6);
```

```
        /* Limitation */
        mf32OutputValue = GFLIB_UpperLimit32(mf32InputValue,
mf32TrimValue);
}
```

3.26.11 See Also

See **GFLIB_Limit16**, **GFLIB_Limit32**, **GFLIB_LowerLimit16**, **GFLIB_LowerLimit32** and **GFLIB_UpperLimit16** for more information.

3.26.12 Performance

Table 3-63. Performance of GFLIB_UpperLimit32 function

Code Size (words)	3	
Data Size (words)	0	
Execution Clock	Min	17/16 cycles
	Max	17/16 cycles

3.27 GFLIB_Sgn

The function calculates signum of the input argument. The function returns:

0x7FFF if X > 0

0 if X = 0

0x8000 if X < 0

3.27.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_Sgn(Frac16 f16Arg)
```

3.27.2 Prototype

```
asm Frac16 GFLIB_SgnFAsm(Frac16 f16Arg)
```

3.27.3 Arguments

Table 3-64. Function Arguments

Name	In/Out	Format	Range	Description
f16Arg	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in header file GFLIB_types.h

3.27.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.27.5 Dependencies

List of all dependent files:

- GFLIB_SgnAsm.h
- GFLIB_types.h

3.27.6 Description

The function **GFLIB_Sgn** calculates the signum of the input argument as depicted in Figure 3-18. For the input values smaller than zero the output is set to 0x8000 and for the values greater than zero the output is set to 0x7FFF. The output is set to zero, if the input argument is equal to zero.

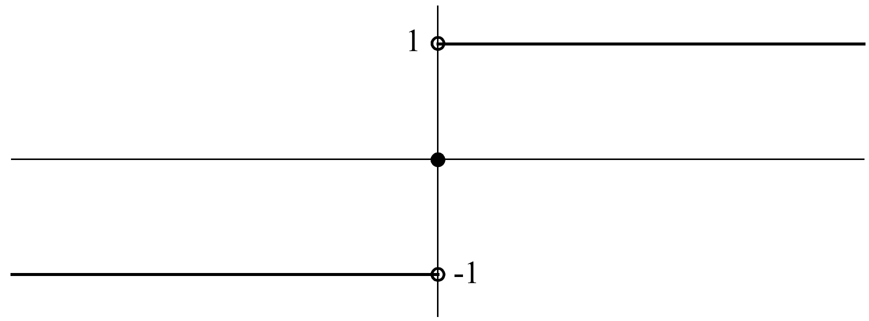

Figure 3-18. Algorithm Transition

3.27.7 Returns

The **GFLIB_Sgn** function returns the sign of the argument.

Table 3-65. Input and Output Returns for the GFLIB_Sgn Function

Input	Output
= 0	0
> 0	1 (32767)
< 0	-1 (-32768)

3.27.8 Range Issues

The input data value is in the range of <-1,1) and the output data values are 0, 1 or -1.

3.27.9 Special Issues

The **GFLIB_Sgn** function calculation is correct regardless of saturation mode.

3.27.10 Implementation

The **GFLIB_Sgn** function is implemented as a function call.

Example 3-27. Implementation Code

```
#include "gflib.h"

static Frac16 mf16Input;
static Frac16 mf16Output;

void main(void)
{
         /* input value 0.6 */
         mf16Input = FRAC16(0.6);

         /* Compute the sgn value */
```

```
        mf16Output = GFLIB_Sgn(mf16Input);
}
```

3.27.11 See Also

See **GFLIB_Sgn2** for more information.

3.27.12 Performance

Table 3-66. Performance of GFLIB_Sgn function

Code Size (words)	12	
Data Size (words)	0	
Execution Clock	Min	23/23 cycles
	Max	23/23 cycles

3.28 GFLIB_Sgn2

The function calculates signum of the input argument with zero being considered as a positive value. The function returns:

0x7FFF if X >= 0

0x8000 if X < 0

3.28.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_Sgn2(Frac16 f16Arg)
```

3.28.2 Prototype

```
asm Frac16 GFLIB_Sgn2FAsm(Frac16 f16Arg)
```

3.28.3 Arguments

Table 3-67. Function Arguments

Name	In/Out	Format	Range	Description
f16Arg	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in header file GFLIB_types.h

3.28.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.28.5 Dependencies

List of all dependent files:

- GFLIB_Sgn2Asm.h
- GFLIB_types.h

3.28.6 Description

The **GFLIB_Sgn2** calculates the signum of the input argument as is depicted in Figure 3-19. For the input values smaller than zero the output is set to 0x8000 and for the values equal or greater than zero the output is set to 0x7FFF.

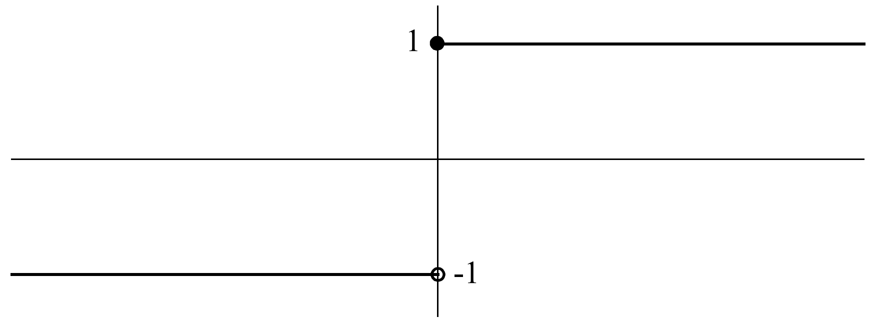

Figure 3-19. Algorithm Transition

3.28.7 Returns

The **GFLIB_Sgn2** function returns the sign of the argument:

Table 3-68.

Input	Output
>= 0	1 (32767)
< 0	-1 (-32768)

3.28.8 Range Issues

The input data value is in the range of <-1,1) and the output data values are 1 (0x7FFF) or -1 (0x8000).

3.28.9 Special Issues

The **GFLIB_Sgn2** function calculation is correct regardless of saturation mode.

3.28.10 Implementation

The **GFLIB_Sgn2** function is implemented as a function call.

Example 3-28. Implementation Code

```
#include "gflib.h"

static Frac16 mf16Input;
static Frac16 mf16Output;

void main(void)
{
        /* input value 0.6 */
        mf16Input = FRAC16(0.6);

        /* Compute the sgn value */
        mf16Output = GFLIB_Sgn2(mf16Input);
```

```
}
```

3.28.11 See Also

See **GFLIB_Sgn** for more information.

3.28.12 Performance

Table 3-69. Performance of GFLIB_Sgn2 function

Code Size (words)	5	
Data Size (words)	0	
Execution Clock	Min	16/17 cycles
	Max	16/17 cycles

3.29 GFLIB_Hyst

The hysteresis function switches output between two predefined values when the input crosses the threshold values.

3.29.1 Synopsis

```
#include "gflib.h"
void GFLIB_Hyst(GFLIB_HYST_T *pudtHystVar)
```

3.29.2 Prototype

```
asm void GFLIB_HystFAsm(GFLIB_HYST_T *pudtHystVar)
```

3.29.3 Arguments

Table 3-70. Function Arguments

Name	In/ Out	Format	Valid Range	Description
*pudtHystVar	in/out	N/A	N/A	pointer to structure containing the input, output values, input thresholds and output constants.

Table 3-71. User type definitions

Typedef	Name	In/ Out	Format	Valid Range	Description
GFLIB_HYST_T	f16In	in	SF16	$8000... $7FFF	input
	f16Out	out	SF16	$8000... $7FFF	hysteresis output
	f16HystOff	in	SF16	$8000... $7FFF	value determining the lower threshold
	f16OutOff	in	SF16	$8000... $7FFF	value of output when the input is lower than f16HystOff
	f16HystOn	in	SF16	$8000... $7FFF	value determining the upper threshold
	f16OutOn	in	SF16	$8000... $7FFF	value of output when the input is higher than f16HystOn

3.29.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.29.5 Dependencies

List of all dependent files:

- GFLIB_HystAsm.h
- gflib.h

3.29.6 Description

The **GFLIB_Hyst** represents a hysteresis (or relay) function. The function switches output between the two predefined values. When the input is higher than upper threshold f16HystOn, the output is high; when the input is below another (lower) threshold f16HystOff, the output is low; when the input is between the two, the output retains its value.

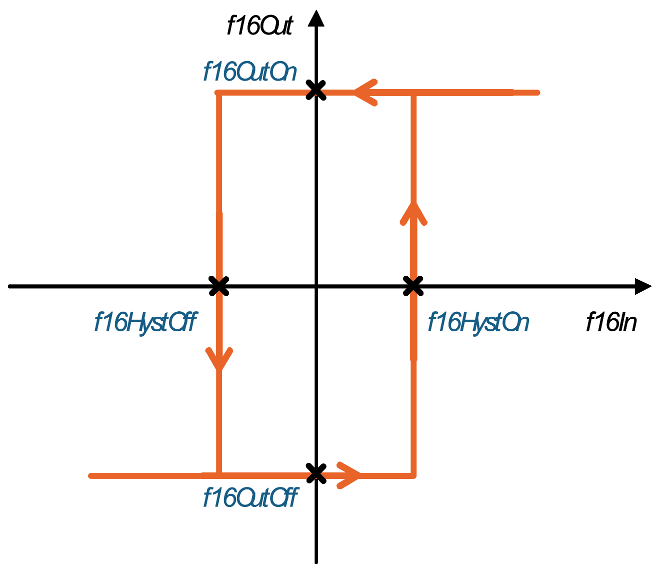

Figure 3-20. Hysteresis function

For correct functionality, f16HystOn must be greater than f16HystOff.

3.29.7 Returns

The calculated output of the function is contained in the function structure pointed to by the pointer *pudtHystVar in the variable f16Out.

3.29.8 Range Issues

The input and output data values are in the fractional range of <-1,1)

3.29.9 Special Issues

The function **GFLIB_Hyst** is the saturation mode independent.

3.29.10 Implementation

Example 3-29. Implementation Code

```
#include "gflib.h"

static GFLIB_HYST_T gudtHyst = GFLIB_HYST_DEFAULT;
static Frac16 f16Out;

void main (void)
{
gudtHyst.f16HystOn = FRAC16(0.5);
gudtHyst.f16HystOff = FRAC16(-0.5);
gudtHyst.f16OutOn = FRAC16(0.25);
gudtHyst.f16OutOff = FRAC16(-0.25);

/* Vary the input value to cross the threshold f16HystOn and f16HystOff */
gudtHyst.f16In = FRAC16(-0.75);

GFLIB_Hyst(&gudtHyst);

f16Out = gudtHyst.f16Out;

/* f16Out = 0xE000 (e.g. -0.25) if f16In <= 0xC000 (e.g. -0.5) */
/* f16Out = 0x2000 (e.g. 0.25) if f16In >= 0x4000 (e.g. 0.5)  */

}
```

3.29.11 Performance

Table 3-72. Performance of GFLIB_Hyst function

Code Size (words)	14	
Data Size (words)	0	
Execution Clock	Min	30/31 cycles
	Max	30/31 cycles

3.30 GFLIB_ControllerPIp

This calculates the parallel form of the Proportional-Integral (PI) regulator.

3.30.1 Synopsis

```
#include ”gflib.h”
Frac16 GFLIB_ControllerPIp(Frac16 f16InputErrorK,
GFLIB_CONTROLLER_PI_P_PARAMS_T *pudtPiParams, const Int16 *pi16SatFlag)
```

3.30.2 Prototype

```
asm Frac16 GFLIB_ControllerPIpFAsm(Frac16 f16InputErrorK,
GFLIB_CONTROLLER_PI_P_PARAMS_T *pudtPiParams, const Int16 *pi16SatFlag)
```

3.30.3 Arguments

Table 3-73. Function Arguments

Name	Type	Format	Range	Description
f16InputErrorK	In	SF16	0x8000... 0x7FFF	Input error at step K processed by P and I terms of the PI algorithm
*pudtPiParams	In/Out	N/A	N/A	Pointer to a structure of PI controller parameters; the GFLIB_CONTROLLER_PI_P_PARAMS_T data type is defined in the header file GFLIB_ControllerPIpAsm.h
*pi16SatFlag	In	SI16	0...1	Pointer to a 16-bit integer variable; if the integer variable passed into the function as a pointer is set to 0, then the integral part is limited only by the PI controller limits. If the integer variable is not zero, then the integral part is frozen immediately

Table 3-74. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GFLIB_CONTROLLER_PI_P_PARAMS_T	f16PropGain	In	SF16	0x0... 0x7FFF	Proportional gain
	f16IntegGain	In	SF16	0x0... 0x7FFF	Integral gain
	i16PropGainShift	In	SI16	0...13	Proportional gain shift
	i16IntegGainShift	In	SI16	0...13	Integral gain shift
	f32IntegPartK_1	In/Out	SF32	0x80000000... 0x7FFFFFFF	State variable; integral part at step k-1; can be modified outside of the function.
	f16UpperLimit	In	SF16	0x8000 ... 0x7FFF	Upper limit of the controller; f16UpperLimit > f16LowerLimit
	f16LowerLimit	In	SF16	0x8000 ... 0x7FFF	Lower limit of the controller; f16UpperLimit > f16LowerLimit
	i16LimitFlag	Out	SI16	0 or 1	Limitation flag; if set to 1, the controller output reached f16UpperLimit or f16LowerLimit

3.30.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.30.5 Dependencies

List of all dependent files:

- GFLIB_ControllerPIpAsm.h
- GFLIB_types.h

3.30.6 Description

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 user to define the P and I parameters independently without interaction. The controller output is limited and the limit values (f16UpperLimit and f16LowerLimit) are defined by the user. The PI controller algorithm also returns a limitation flag. This flag named "i16LimitFlag" is the member of the structure of the PI controller parameters (GFLIB_CONTROLLER_PI_P_PARAMS_T). If the PI controller output reaches the upper or lower limit then i16LimitFlag = 1 otherwise i16LimitFlag = 0.

An anti-windup strategy is implemented by limiting the integral portion. There are two ways of limiting the integral portion:

- The integral state is limited by the controller limits, in the same way as the controller output.
- When the variable i16SatFlag set by the user software outside the PI controller function and passed into the function and the pointer pi16SatFlag is not zero, then the integral portion is frozen.

The PI algorithm in the continuous time domain:

$$u(t) = K\left[e(t) + \frac{1}{T_I}\int_0^{\tau} e(\tau)d(\tau)\right] \quad \text{Eqn. 3-45}$$

where

e(t) - input error in the continuous time domain; processed by the P and I terms of the PI algorithm

u(t) - controller output in the continuous time domain

T_I - integral time constant - [s]

Equation 3-45 can be rewritten into the discrete time domain by approximating the integral and derivative terms.

The integral term is approximated by the Backward Euler method, also known as backward rectangular or right - hand approximation as follows.

$$u_I(k) = u_I(k-1) + T \cdot e(k) \quad \text{Eqn. 3-46}$$

The discrete time domain representation of the PI algorithms

$$u(k) = K \cdot e(k) + u_I(k-1) + K_I \cdot e(k) \quad \text{Eqn. 3-47}$$

where

e(k) - input error at step k; processed by the P and I terms

u(k) - controller output at step k

K - proportional gain

K_I - integral gain

T - sampling time/period - [s]

$$K_I = K \cdot \frac{T}{T_I}$$ **Eqn. 3-48**

The discrete time domain representation of the PI algorithm scaled into the fractional range.

$$u_f(k) = K_{sc} \cdot e_f(k) + u_{If}(k-1) + K_{Isc} \cdot e_f(k)$$ **Eqn. 3-49**

where

$$u_f(k) = u(k)/u_{max}$$ **Eqn. 3-50**

$$e_f(k) = e(k)/e_{max}$$ **Eqn. 3-51**

$$K_{sc} = K \cdot \frac{e_{max}}{u_{max}}$$ **Eqn. 3-52**

$$K_{Isc} = K \cdot \frac{T}{T_I} \cdot \frac{e_{max}}{u_{max}} = K_I \cdot \frac{e_{max}}{u_{max}}$$ **Eqn. 3-53**

where

e_{max} - input max range

u_{max} - output max range

Each parameter (e.g. K_{Isc}) of the PI algorithm is represented by two parameters in the processor implementation (e.g. f16IntegGain and i16IntegGainShift).

$$f16PropGain = K_{sc} \cdot 2^{-i16PropGainShift}$$ **Eqn. 3-54**

$$f16IntegGain = K_{Isc} \cdot 2^{-i16IntegGainShift}$$ **Eqn. 3-55**

where

$$0 \le f16PropGain < 1$$ **Eqn. 3-56**

$$0 \le f16IntegGain < 1$$ **Eqn. 3-57**

$$0 \le i16PropGainShift < 14$$ **Eqn. 3-58**

$$0 \le i16IntegGainShift < 14$$ **Eqn. 3-59**

Example

Assumption: K_{Isc}= 2.4

In this case, K_{Isc} cannot be directly interpreted as a fractional value because the range of fractional values is <-1,1) and the range of the parameter f16IntegGain is <0,1). It is necessary to scale the K_{Isc}

parameter using i16IntegGainShift to fit the parameter f16IntegGain into the range <0,1)

Solution:

The most precise scaling approach is to scale down the parameter K_{Isc} to have f16IntegGain in the following range

0.5 <= f16IntegGain < 1

and to calculate the corresponding i16IntegGainShift parameter.

$$\frac{\log(K_{Isc}) - \log(0{,}5)}{\log 2} \geq i16IntegGainShift \qquad \textbf{Eqn. 3-60}$$

$$\frac{\log(2{,}4) - \log(0{,}5)}{\log 2} \geq i16IntegGainShift \qquad \textbf{Eqn. 3-61}$$

$$2{,}26 \geq i16IntegGainShift \qquad \textbf{Eqn. 3-62}$$

$$\frac{\log(K_{Isc}) - \log(1)}{\log 2} < i16IntegGainShift \qquad \textbf{Eqn. 3-63}$$

$$\frac{\log(K_{Isc})}{\log 2} < i16IntegGainShift \qquad \textbf{Eqn. 3-64}$$

$$\frac{\log(2{,}4)}{\log 2} < i16IntegGainShift \qquad \textbf{Eqn. 3-65}$$

$$1{,}26 < i16IntegGainShift \qquad \textbf{Eqn. 3-66}$$

The parameter i16IntegGainShift is in the following range:

$$1{,}26 < i16IntegGainShift \leq 2{,}26 \qquad \textbf{Eqn. 3-67}$$

Because this parameter is an integer value, the result is

$$i16IntegGainShift = 2 \qquad \textbf{Eqn. 3-68}$$

Then

$$f16IntegGain = K_{Isc} \cdot 2^{-i16IntegGainShift} \qquad \textbf{Eqn. 3-69}$$

$$f16IntegGain = 2{,}4 \cdot 2^{(-2)} = 0{,}6 \qquad \textbf{Eqn. 3-70}$$

Result:

f16IntegGain = 0.6

i16IntegGainShift = 2

3.30.7 Returns

The function **GFLIB_ControllerPIp** returns a fractional value as a result of the PI algorithm. The value returned by the algorithm is in the following range:

$$f16LowerLimit \le PIresult \le f16UpperLimit \quad \text{Eqn. 3-71}$$

3.30.8 Range Issues

The PI controller parameters are in the following range

$$0 \le f16PropGain < 1 \quad \text{Eqn. 3-72}$$

$$0 \le f16IntegGain < 1 \quad \text{Eqn. 3-73}$$

$$0 \le i16PropGainShift < 14 \quad \text{Eqn. 3-74}$$

$$0 \le i16IntegGainShift < 14 \quad \text{Eqn. 3-75}$$

3.30.9 Special Issues

The function **GFLIB_ControllerPIp** is the saturation mode independent.

3.30.10 Implementation

Example 3-30. Implementation Code

```
#include "gflib.h"

static Frac16 mf16DesiredValue;
static Frac16 mf16MeasuredValue;
static Frac16 mf16ErrorK;
static Int16 mi16SatFlag;
static Frac16 mf16ControllerOutput;


/* Controller parameters */
static GFLIB_CONTROLLER_PI_P_PARAMS_T mudtControllerParam;

void Isr(void);

void main(void)
{
        /* Controller parameters initialization */
        mudtControllerParam.f16PropGain = FRAC16(0.5);
        mudtControllerParam.f16IntegGain = FRAC16(0.032);
        mudtControllerParam.i16PropGainShift = 1;
        mudtControllerParam.i16IntegGainShift = 0;
        mudtControllerParam.f32IntegPartK_1 = 0;
        mudtControllerParam.f16UpperLimit = FRAC16(0.8);
        mudtControllerParam.f16LowerLimit = FRAC16(-0.7);
```

```
          /* Desired value initialization */
          mf16DesiredValue = FRAC16(0.5);

          /* Measured value initialization */
          mf16MeasuredValue = 0;

          /* Saturation flag initialization */
          mi16SatFlag = 0;
}

/* Periodical function or interrupt */
void Isr(void)
{
          /* Error calculation */
          mf16ErrorK = mf16DesiredValue - mf16MeasuredValue;

          /* Controller calculation */
          mf16ControllerOutput = GFLIB_ControllerPIp(mf16ErrorK,
&mudtControllerParam, &mi16SatFlag);
}
```

3.30.11 See Also

See **GFLIB_ControllerPIr**, **GFLIB_ControllerPIrLim**, **GFLIB_ControllerPIDp** and **GFLIB_ControllerPIDr** for more information.

3.30.12 Performance

Table 3-75. Performance of GFLIB_ControllerPIp function

Code Size (words)	52	
Data Size (words)	0	
Execution Clock	Min	66/63 cycles
	Max	74/72 cycles

3.31 GFLIB_ControllerPIr

The function calculates the recurrent form of the Proportional-Integral (PI) regulator.

3.31.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_ControllerPIr(Frac16 f16Error,
GFLIB_CONTROLLER_PI_RECURRENT_T * const pudtCtrl)
```

3.31.2 Prototype

```
asm Frac16 GFLIB_ControllerPIRecurrentFAsm(Frac16 f16Err,
GFLIB_CONTROLLER_PI_RECURRENT_T * const pudtCtrl)
```

3.31.3 Arguments

Table 3-76. Function Arguments

Name	In/Out	Format	Range	Description
f16Error	In	SF16	0x8000... 0x7FFF	Error as input argument; the **Frac16** data type is defined in header file GFLIB_types.h
*pudtCtrl	In/out	struct	N/A	Pointer to a controller structure, which contains controller coefficients and integrator delay line; the GFLIB_CONTROLLER_PI_RECURRENT_T data type is defined in header file GFLIB_ControllerPIRecurrentAsm.h

Table 3-77. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GFLIB_CONTROLLER_PI_RECURRENT_T	f32Acc	In/Out	SF32	0x80000000... 0x7FFFFFFF	Internal controller accumulator
	f16ErrorK_1	In/Out	SF16	0x8000 ... 0x7FFF	Input error at step k-1; delayed value of error at step k
	f16CC1Sc	In	SF16	0x8000 ... 0x7FFF	First controller coefficient scaled to fractional format
	f16CC2Sc	In	SF16	0x8000 ... 0x7FFF	Second controller coefficient scaled to fractional format
	ui16NShift	In	UI16	0...15	Scaling shift

3.31.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.31.5 Dependencies

List of all dependent files:

- GFLIB_ControllerPIRecurrentAsm.h
- GFLIB_types.h

3.31.6 Description

The time continuos function of the PI controller is defined as

$$u(t) = K_P \cdot e(t) + K_I \cdot \int_0^t e(t)dt$$ **Eqn. 3-76**

Different techniques are used to convert the continuous PI controller function into the discrete representation. However, the continuous function can only be approximated and the discrete representation can never be exactly equivalent. Different methods can result different controller performances.

The resulting difference equation derived by the discretization method is in the form as reported below

$$u(k) = u(k-1) + CC1 \cdot e(k) + CC2 \cdot e(k-1)$$ **Eqn. 3-77**

The transition from the continuous to the discrete time domain reveals the following controller coefficients

Table 3-78. Controller Coefficients

Controller Coefficients	Bilinear Transformation	Backward Rectangular	Forward Rectangular
CC1	$K_P + K_I \cdot T_S/2$	$K_P + K_I \cdot T_S$	K_P
CC2	$-K_P + K_I \cdot T_S/2$	$-K_P$	$-K_P + K_I \cdot T_S$

where

- K_P - proportional gain
- K_I - integral gain
- T_S - sampling period

The discrete time domain representation of the recurrent PI algorithm scaled into the fractional range is as

$$f16Uk = f32Acc + f16CC1 \cdot f16Error + f16CC2 \cdot \mathrm{f16ErrorK\_1} \qquad \text{Eqn. 3-78}$$

where f32Acc is the accumulated controller portion over time and is used as the internal variable of this algorithm. The f16CC1 and f16CC2 are recurrent controller coefficients which are adapted as

$$f16CC1 = CC1 \cdot \frac{Emax}{Umax} \qquad \text{Eqn. 3-79}$$

$$f16CC2 = CC2 \cdot \frac{Emax}{Umax} \qquad \text{Eqn. 3-80}$$

The input for the recurrent PI controller is f16Error which is adapted as

$$\mathrm{f16Error} = \mathrm{f16Desired} - \mathrm{f16Actual} \qquad \text{Eqn. 3-81}$$

$$\mathrm{f16Error} = \frac{\mathrm{Error}}{\mathrm{EMax}} \qquad \text{Eqn. 3-82}$$

where f16Error is a fractional number which must be within the fractional range <-1, 0.9999>. The f16Error value is processed within the PI controller algorithm and the delayed value is stored in f16Error_1 for the next calculation. The delayed operation is performed internally by the algorithm itself where any user interaction is not required.

For proper operation of the recurrent PI controller implemented on the 16/32-bit DSC a care must be taken due to the fixed point representation of individual values. All coefficients need to be represented as 16-bit fixed point numbers. The input value f16Error is assumed to be already in the correct 16-bit fixed point number format. Other controller variables as f32Acc, f16Error16K_1 are internally handled by the algorithm of the recurrent PI controller. The controller coefficients f16CC1, f16CC2 must be prepared in the correct 16-bit fixed point number format by user. A scaling shift ui16NShift is introduced to be scaled to the 16-bit fixed point format. Then fractional representation on the DSC of the recurrent PI controller is calculated by the following formula as

$$f16Uk = f16Acc + f16CC1Sc \cdot f16Error + f16CC2Sc \cdot f16ErrorK1 \qquad \text{Eqn. 3-83}$$

where

$$\mathrm{f16CC1Sc} = \mathrm{f16CC1} \cdot 2^{-\mathrm{uiNShift16}} \qquad \text{Eqn. 3-84}$$

$$\mathrm{f16CC2Sc} = \mathrm{f16CC2} \cdot 2^{-\mathrm{uiNShift16}} \qquad \text{Eqn. 3-85}$$

ui16NShift is chosen that the coefficients reside within the common range <-1, 0.9999>. In addition, ui16NShift is chosen as a power of 2, the final de-scaling is a simple shift operation.

$$ui16NShift = \max\left(\mathrm{ceil}\left(\frac{\log(f16CC1)}{\log(2)}\right), \mathrm{ceil}\left(\frac{\log(f16CC2)}{\log(2)}\right)\right) \quad \textbf{Eqn. 3-86}$$

Example:

After the controller coefficients Kp and Ki are determined in the continuous time domain and the suitable sampling time Ts is chosen, then controller coefficients in the discrete domain are calculated as stated in Table 3-79.

Table 3-79. Controller Coefficients:
K_p=125[V/A]; K_i=24000[V/A]; T_s=125e-6[sec]; ErrorMAX=IMAX=8[A]; UMAX=240[V]

Controller Coefficients	Bilinear Transformation	Backward Rectangular	Forward Rectangular
CC1	126.5	128	125
CC2	-123.5	-125	-122
f16CC1	4.216666667	4.266666667	4.166666667
f16CC2	-4.116666667	-4.166666667	-4.066666667
ui16NShuft	3	3	3
f16CC1Sc	0.527083333	0.533333333	0.520833333
f16CC2Sc	-0.514583333	-0.520833333	-0.508333333

3.31.7 Returns

The function returns a 16-bit fractional value as result of the calculation of PI algorithm.

3.31.8 Range Issues

The PI controller parameters are in the following range

$$-1 \le f16ErrorK\_1 \le 0{,}9999 \quad \textbf{Eqn. 3-87}$$

$$-1 \le f16CC1Sc \le 0{,}9999 \quad \textbf{Eqn. 3-88}$$

$$-1 \le f16CC2Sc \le 0{,}9999 \quad \textbf{Eqn. 3-89}$$

$$0 \le ui16NShift < 16 \quad \textbf{Eqn. 3-90}$$

3.31.9 Special Issues

The function **GFLIB_ControllerPIr** is the saturation mode independent.

3.31.10 Implementation

Example 3-31. Implementation Code

```
#include "gflib.h"

static Frac16 mf16DesiredValue;
static Frac16 mf16MeasuredValue;
static Frac16 mf16ErrorK;
static Frac16 mf16ControllerOutput;


/* Controller parameters */
static GFLIB_CONTROLLER_PI_RECURRENT_T mudtControllerParam;

void Isr(void);

void main(void)
{
         /* Controller parameters initialization */
         mudtControllerParam.f16CC1Sc = FRAC16(0.527083333);
         mudtControllerParam.f16CC2Sc = FRAC16(-0.514583333);
         mudtControllerParam.ui16NShift = 3;
         mudtControllerParam.f16ErrorK_1 = 0;
         mudtControllerParam.f32Acc = 0;

         /* Desired value initialization */
         mf16DesiredValue = FRAC16(0.5);

         /* Measured value initialization */
         mf16MeasuredValue = 0;
}

/* Periodical function or interrupt */
void Isr(void)
{
         /* Error calculation */
         mf16ErrorK = mf16DesiredValue - mf16MeasuredValue;

         /* Ramp generation */
         mf16ControllerOutput = GFLIB_ControllerPIr(mf16ErrorK,
&mudtControllerParam);
}
```

3.31.11 See Also

See **GFLIB_ControllerPIp**, **GFLIB_ControllerPIrLim**, **GFLIB_ControllerPIDp** and **GFLIB_ControllerPIDr** for more information.

3.31.12 Performance

Table 3-80. Performance of GFLIB_ControllerPIr function

Code Size (words)	19	
Data Size (words)	0	
Execution Clock	Min	35/34 cycles
	Max	35/34 cycles

3.32 GFLIB_ControllerPIrLim

The function calculates the recurrent form of the Proportional-Integral (PI) regulator with limitation.

3.32.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_ControllerPIrLim(Frac16 f16Error,
GFLIB_CONTROLLER_PI_RECURRENT_LIM_T * const pudtCtrl)
```

3.32.2 Prototype

```
asm Frac16 GFLIB_ControllerPIRecurrentLimFAsm(Frac16 f16Err,
GFLIB_CONTROLLER_PI_RECURRENT_LIM_T * const pudtCtrl)
```

3.32.3 Arguments

Table 3-81. Function Arguments

Name	In/Out	Format	Range	Description
f16Error	In	SF16	0x8000... 0x7FFF	error as input argument; the Frac16 data type is defined in header file GFLIB_types.h
*pudtCtrl	In/Out	struct	N/A	pointer to a controller structure, which contains controller coefficients and integrator delay line; the GFLIB_CONTROLLER_PI_RECURRENT_ASM_T data type is defined in header file GFLIB_ControllerPIRecurrentLimAsm.h

Table 3-82. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GFLIB_CONTROLLER_PI_RECURRENT_LIM_T	f32Acc	In/Out	SF32	0x80000000... 0x7FFFFFFF	Internal controller accumulator
	f16ErrorK_1	In/out	SF16	0x8000 ... 0x7FFF	Input error at step k-1; delayed value of error at step k
	f16CC1Sc	In	SF16	0x8000 ... 0x7FFF	First controller coefficient scaled to fractional format
	f16CC2Sc	In	SF16	0x8000 ... 0x7FFF	Second controller coefficient scaled to fractional format
	ui16NShift	In	UI16	0...15	Scaling shift
	f16UpperLim	In	SF16	—	Upper limit of the controller; f16UpperLimit > f16LowerLimit
	f16LowerLim	In	SF16	—	Lower limit of the controller; f16UpperLimit > f16LowerLimit
	ui16SatFlag	Out	UI16	—	Saturation flag; if set to 1, the controller output reached f16UpperLimit or f16LowerLimit

3.32.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.32.5 Dependencies

List of all dependent files:

1. GFLIB_ControllerPIRecurrentLimAsm.h
2. GFLIB_types.h

3.32.6 Description

The time continuos function of the PI controller is defined as

$$u(t) = K_P \cdot e(t) + K_I \cdot \int_0^t e(t)dt \qquad \text{Eqn. 3-91}$$

Different techniques are used to convert the continuous PI controller function into the discrete representation. However, the continuous function can only be approximated and the discrete representation can never be exactly equivalent. Different methods can result different controller performances.

The resulting difference equation derived by the discretization method is in the form as reported below

$$u(k) = u(k-1) + CC1 \cdot e(k) + CC2 \cdot e(k-1) \qquad \text{Eqn. 3-92}$$

The transition from the continuous to the discrete time domain reveals the following controller coefficients

Table 3-83. Controller Coefficients

Controller Coefficients	Bilinear Transformation	Backward Rectangular	Forward Rectangular
CC1	$K_P + K_I \cdot T_S/2$	$K_P + K_I \cdot T_S$	K_P
CC2	$-K_P + K_I \cdot T_S/2$	$-K_P$	$-K_P + K_I \cdot T_S$

where

- K_P - proportional gain
- K_I - integral gain
- T_S - sampling period

In this controller implementation the actual output variable u(k) is bounded not to exceed the given limit values UpperLimit, LowerLimit. This is the case due to the bounded power of the actuator or to the physical constraints of the plant. The bounds are described by a saturation element equation Equation 3-93 where the calculated variable u(k) obtained from the controller is further limited and the resulting variable acting on the plant is determined as

$$u(k) = \begin{cases} \text{UpperLimit} & \rightarrow u(k) > \text{UpperLimit} \\ u(k) & \rightarrow \text{LowerLimit} \le u(k) \le \text{UpperLimit} \\ Lower\text{Limit} & \rightarrow u(k) < \text{LowerLimit} \end{cases} \qquad \text{Eqn. 3-93}$$

When the bounds are exceeded, the non-linear saturation characteristic takes effect and influences the dynamic behavior. The described limitation is implemented within the PI recurrent controller where the limitation is evaluated during the calculation. 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.

The discrete time domain representation of the recurrent PI algorithm scaled into the fractional range is as

$$f16Uk = f32Acc + f16CC1 \cdot f16Error + f16CC2 \cdot \mathrm{f16ErrorK\_1} \qquad \textbf{Eqn. 3-94}$$

where f32Acc is the accumulated controller portion over time and is used as an internal variable of this algorithm. f16CC1 and f16CC2 are the recurrent controller coefficients which are adapted as

$$f16CC1 = CC1 \cdot \frac{Emax}{Umax} \qquad \textbf{Eqn. 3-95}$$

$$f16CC2 = CC2 \cdot \frac{Emax}{Umax} \qquad \textbf{Eqn. 3-96}$$

The input for the recurrent PI controller is considered f16Error which is adapted as

$$\mathrm{f16Error} = \mathrm{f16Desired} - \mathrm{f16Actual} \qquad \textbf{Eqn. 3-97}$$

$$\mathrm{f16Error} = \frac{\mathrm{Error}}{\mathrm{Emax}} \qquad \textbf{Eqn. 3-98}$$

where f16Error is a fractional number which must be within the fractional range <-1, 0.9999>. The f16Error value is processed within the PI controller algorithm and the delayed value is stored in f16Error_1 for the next calculation. The delayed operation is performed internally by the algorithm itself where any user interaction is not required.

For proper operation of the recurrent PI controller implemented on the 16/32-bit DSC a care must be taken due to fixed point representation of the individual values. All coefficients need to be represented as 16-bit fixed point numbers. The input value f16Error is assumed to be already in the correct 16-bit fixed point number format. Other controller variables as f32Acc, f16Error16K_1 are internally handled by the algorithm of the recurrent PI controller. The controller coefficients f16CC1, f16CC2 must be prepared in the correct 16-bit fixed point number format by user. A scaling shift ui16NShift is introduced to be scaled to the 16-bit fixed point format. Then the fractional representation on the DSC of the recurrent PI controller is calculated by the following formula as

$$f16Uk = f16Acc + f16CC1Sc \cdot f16Error + f16CC2Sc \cdot f16ErrorK1 \qquad \textbf{Eqn. 3-99}$$

where

$$\mathrm{f16CC1Sc} = \mathrm{f16CC1} \cdot 2^{-\mathrm{uiNShift16}} \qquad \textbf{Eqn. 3-100}$$

$$\mathrm{f16CC2Sc} = \mathrm{f16CC2} \cdot 2^{-\mathrm{uiNShift16}} \qquad \textbf{Eqn. 3-101}$$

ui16NShift is chosen that the coefficients reside within the common range <-1, 0.9999>. In addition, ui16NShift is chosen as a power of 2, the final de-scaling is a simple shift operation.

$$ui16NShift = \max\left(\text{ceil}\left(\frac{\log(f16CC1)}{\log(2)}\right), \text{ceil}\left(\frac{\log(f16CC2)}{\log(2)}\right)\right)$$ **Eqn. 3-102**

Example:

After the controller coefficients Kp, Ki are determined in the continuous time domain and the suitable sampling time Ts is chosen, then controller coefficients in the discrete domain are calculated as stated in Table 3-84.

Table 3-84. Controller Coefficients: Kp=125[V/A]; Ki=24000[V/A]; Ts=125e-6[sec]; ErrorMAX=IMAX=8[A]; UMAX=240[V]

Controller Coefficients	Bilinear Transformation	Backward Rectangular	Forward Rectangular
CC1	126.5	128	125
CC2	-123.5	-125	-122
f16CC1	4.216666667	4.266666667	4.166666667
f16CC2	-4.116666667	-4.166666667	-4.066666667
ui16NShuft	3	3	3
f16CC1Sc	0.527083333	0.533333333	0.520833333
f16CC2Sc	-0.514583333	-0.520833333	-0.508333333

3.32.7 Returns

The function returns a 16-bit fractional value as result of calculation of the PI algorithm.

3.32.8 Range Issues

The PI controller parameters are in the following range

$$-1 \le f16ErrorK\_1 \le 0{,}9999$$ **Eqn. 3-103**

$$-1 \le f16CC1Sc \le 0{,}9999$$ **Eqn. 3-104**

$$-1 \le f16CC2Sc \le 0{,}9999$$ **Eqn. 3-105**

$$0 \le ui16NShift < 16$$ **Eqn. 3-106**

3.32.9 Special Issues

The function **GFLIB_ControllerPIrLim** is the saturation mode independent.

3.32.10 Implementation

Example 3-32. Implementation Code

```
#include "gflib.h"

static Frac16 mf16DesiredValue;
static Frac16 mf16MeasuredValue;
static Frac16 mf16ErrorK;
static Frac16 mf16ControllerOutput;


/* Controller parameters */
static GFLIB_CONTROLLER_PI_RECURRENT_LIM_T mudtControllerParam;

void Isr(void);

void main(void)
{
        /* Controller parameters initialization */
        mudtControllerParam.f16CC1Sc = FRAC16(0.527083333);
        mudtControllerParam.f16CC2Sc = FRAC16(-0.514583333);
        mudtControllerParam.ui16NShift = 3;
        mudtControllerParam.f16UpperLim = FRAC16(0.5);
        mudtControllerParam.f16LowerLim = FRAC16(-0.5);
        mudtControllerParam.f16ErrorK_1 = 0;
        mudtControllerParam.f32Acc = 0;

        /* Desired value initialization */
        mf16DesiredValue = FRAC16(0.5);

        /* Measured value initialization */
        mf16MeasuredValue = 0;
}

/* Periodical function or interrupt */
void Isr(void)
{
        /* Error calculation */
        mf16ErrorK = mf16DesiredValue - mf16MeasuredValue;

        /* Controller calculation */
        mf16ControllerOutput = GFLIB_ControllerPIrLim(mf16ErrorK,
&mudtControllerParam);
}
```

3.32.11 See Also

See **GFLIB_ControllerPIp**, **GFLIB_ControllerPIr**, **GFLIB_ControllerPIDp** and **GFLIB_ControllerPIDr** for more information.

3.32.12 Performance

Table 3-85. Performance of GFLIB_ControllerPIrLim Function

Code Size (words)	26	
Data Size (words)	0	
Execution Clock	Min	43/41 cycles
	Max	43/41 cycles

3.33 GFLIB_ControllerPIDp

The function calculates the parallel form of the Proportional-Integral-Derivative (PID) regulator.

3.33.1 Synopsis

```
#include ”gflib.h”
Frac16 GFLIB_ControllerPIDp(Frac16 f16InputErrorK, Frac16
f16InputDErrorK, GFLIB_CONTROLLER_PID_P_PARAMS_T *pudtPidParams, const
Int16 *pi16SatFlag, Frac16 *pf16InputDErrorK_1)
```

3.33.2 Prototype

```
asm Frac16 GFLIB_ControllerPIDpFAsm(Frac16 f16InputErrorK, Frac16
f16InputDErrorK, GFLIB_CONTROLLER_PID_P_PARAMS_T *pudtPidParams, const
Int16 *pi16SatFlag, Frac16 *pf16InputDErrorK_1)
```

3.33.3 Arguments

Table 3-86. Function Arguments

Name	Type	Format	Range	Description
f16InputErrorK	In	SF16	0x8000 ... 0x7FFF	Input error at step K processed by P and I terms of the PID algorithm
f16InputDErrorK	In	SF16	0x8000 ... 0x7FFF	Input error at step K processed by the D term of the PID algorithm
*pudtPidParams	In/Out	N/A	N/A	Pointer to a structure of PID controller parameters; the GFLIB_CONTROLLER_PID_P_PARAMS_T data type is defined in header file GFLIB_ControllerPIDpAsm.h
*pi16SatFlag	In	N/A	N/A	Pointer to a 16-bit integer variable; if the integer variable passed into the function as a pointer is set to 0, then the integral part is limited by the PID controller limits. If the integer variable is not zero, then the integral part is frozen
*pf16InputDErrorK_1	In/Out	N/A	N/A	Pointer to a 16-bit fractional variable; input error at step K-1processed only by the derivative term of the PID algorithm. It is the state variable modified by the function. The variable can also be modified outside of the function. The purpose can be the initialization of the variable

Table 3-87. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GFLIB_CONTROLLER_PID_P_PARAMS_T	f16PropGain	In	SF16	0x0... 0x7FFF	Proportional gain
	f16IntegGain	In	SF16	0x0... 0x7FFF	Integral gain
	f16DerGain	In	SF16	0x0... 0x7FFF	Derivative gain
	i16PropGainShift	In	SI16	0...13	Proportional gain shift
	i16IntegGainShift	In	SI16	0...13	Integral gain shift
	i16DerGainShift	In	SI16	0...13	Derivative gain shift
	f32IntegPartK_1	In/Out	SF32	0x80000000... 0x7FFFFFFF	State variable; integral part at step k-1; can be modified outside of the function.
	f16UpperLimit	In	SF16	0x8000 ... 0x7FFF	Upper limit of the controller; f16UpperLimit > f16LowerLimit
	f16LowerLimit	In	SF16	0x8000 ... 0x7FFF	Lower limit of the controller; f16UpperLimit > f16LowerLimit
	i16LimitFlag	Out	SI16	0 or 1	Limitation flag; if set to 1, the controller output reached f16UpperLimit or f16LowerLimit

3.33.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.33.5 Dependencies

List of all dependent files:

1. GFLIB_ControllerPIDpAsm.h
2. GFLIB_types.h

3.33.6 Description

The **GFLIB_ControllerPIDp** function calculates the Proportional-Integral-Derivative (PID) algorithm according to the equations below. The PID algorithm is implemented in the parallel (non-interacting) form allowing the user to define the P, I and D parameters independently without interaction. The controller output is limited and the limit values (f16UpperLimit and f16LowerLimit) are defined by user. The PID controller algorithm also returns the limitation flag. This flag named "i16LimitFlag" is the member of the structure of the PID controller parameters (GFLIB_CONTROLLER_PID_P_PARAMS_T). If the PID controller output reaches the upper or lower limit then i16LimitFlag = 1 otherwise i16LimitFlag = 0.

An anti-windup strategy is implemented by limiting the integral portion. There are two ways of limiting the integral part:

- The integral state is limited by the controller limits, in the same way as the controller output.
- When the variable satFlag set by the user software outside the PID controller function and passed into the function and the pointer pi16SatFlag is not zero, then the integral portion is frozen.

The PID algorithm in the continuous time domain:

$$u(t) = K\left[e(t) + \frac{1}{T_I}\int_0^{\tau} e(\tau)d(\tau) + T_D \cdot \frac{d}{dt}e_D(t)\right]$$ **Eqn. 3-107**

where

$e(t)$ - input error in the continuous time domain; processed by the P and I terms of the PID algorithm

$e_D(t)$ - input error in the continuous time domain; processed by the D term of the PID algorithm

$u(t)$ - controller output in the continuous time domain

T_I - integral time constant - [s]

T_D - derivative time constant -[s]

Equation 3-107 can be rewritten into the discrete time domain by approximating the integral and derivative terms.

The integral term is approximated by the Backward Euler method, also known as backward rectangular or right - hand or backward difference approximation as follows.

$$u_I(k) = u_I(k-1) + T \cdot e(k)$$ **Eqn. 3-108**

The derivative term is approximated in the same way as the integral term (backward difference).

$$\frac{d}{dt}e_D(t) = \frac{e_D(k) - e_D(k-1)}{T}$$ **Eqn. 3-109**

The discrete time domain representation of the PID algorithm

$$u(k)= K \cdot e(k) + u_I(k-1) + K_I \cdot e(k) + K_D \cdot (e_D(k) - e_D(k-1)) \quad \text{Eqn. 3-110}$$

where

$e(k)$ - input error at step k; processed by the P and I terms
$e_D(k)$ - input error at step k; processed by the D term
$e_D(k-1)$ - input error at step k-1; processed by the D term
$u(k)$ - controller output at the step k
K - proportional gain
K_I - integral gain
K_D - derivative gain
T - sampling time/period - [s]

$$K_I = K \cdot \frac{T}{T_I} \quad \text{Eqn. 3-111}$$

$$K_D = K \cdot \frac{T_D}{T} \quad \text{Eqn. 3-112}$$

The discrete time domain representation of the PID algorithm scaled into the fractional range.

$$u_f(k)= K_{sc} \cdot e_f(k) + u_{If}(k-1) + K_{Isc} \cdot e_f(k) + K_{Dsc} \cdot (e_{Df}(k) - e_{Df}(k-1)) \quad \text{Eqn. 3-113}$$

where

$$u_f(k) = u(k)/u_{max} \quad \text{Eqn. 3-114}$$

$$e_f(k) = e(k)/e_{max} \quad \text{Eqn. 3-115}$$

$$e_{Df}(k) = e_D(k)/e_{max} \quad \text{Eqn. 3-116}$$

$$K_{sc} = K \cdot \frac{e_{max}}{u_{max}} \quad \text{Eqn. 3-117}$$

$$K_{Isc} = K \cdot \frac{T}{T_I} \cdot \frac{e_{max}}{u_{max}} = K_I \cdot \frac{e_{max}}{u_{max}} \quad \text{Eqn. 3-118}$$

$$K_{Dsc} = K \cdot \frac{T_D}{T} \cdot \frac{e_{max}}{u_{max}} = K_D \cdot \frac{e_{max}}{u_{max}} \quad \text{Eqn. 3-119}$$

where

e_{max} - input max range
u_{max} - output max range

Each parameter (e.g. K_{sc}) of the PID algorithm is represented by two parameters in the processor implementation (e.g. f16PropGain and i16PropGainShift).

$$f16PropGain = K_{sc} \cdot 2^{-i16PropGainShift}$$ **Eqn. 3-120**

$$f16IntegGain = K_{Isc} \cdot 2^{-i16IntegGainShift}$$ **Eqn. 3-121**

$$f16DerGain = K_{Dsc} \cdot 2^{-i16DerGainShift}$$ **Eqn. 3-122**

where

$$0 \le f16PropGain < 1$$ **Eqn. 3-123**

$$0 \le f16IntegGain < 1$$ **Eqn. 3-124**

$$0 \le f16DerGain < 1$$ **Eqn. 3-125**

$$0 \le i16PropGainShift < 14$$ **Eqn. 3-126**

$$0 \le i16IntegGainShift < 14$$ **Eqn. 3-127**

$$0 \le i16DerGainShift < 14$$ **Eqn. 3-128**

Example

Assumption: K_{sc}=1.8

In this case K_{sc} cannot be directly interpreted as a fractional value because the range of the fractional values is <-1,1) and the range of the parameter f16PropGain is <0,1). It is necessary to scale the K_{sc} parameter using i16PropGainShift to fit the parameter f16PropGain into the range <0,1)

Solution:

The most precise scaling approach is to scale down the parameter K_{sc} to have the f16PropGain in the following range

0.5 <= f16PropGain < 1

and to calculate the corresponding i16PropGainShift parameter.

$$\frac{\log(K_{sc}) - \log(0{,}5)}{\log 2} \ge i16PropGainShift$$ **Eqn. 3-129**

$$\frac{\log(1{,}8) - \log(0{,}5)}{\log 2} \ge i16PropGainShift$$ **Eqn. 3-130**

$$1{,}8 \ge i16PropGainShift$$ **Eqn. 3-131**

$$\frac{\log(K_{sc}) - \log(1)}{\log 2} < i16PropGainShift \qquad \textbf{Eqn. 3-132}$$

$$\frac{\log(K_{sc})}{\log 2} < i16PropGainShift \qquad \textbf{Eqn. 3-133}$$

$$0{,}8 < i16PropGainShift \qquad \textbf{Eqn. 3-134}$$

The parameter i16PropGainShift is in the following range:

$$0{,}8 < i16PropGainShift \le 1{,}8 \qquad \textbf{Eqn. 3-135}$$

Because this parameter is an integer value, the result is

$$i16PropGainShift = 1 \qquad \textbf{Eqn. 3-136}$$

Then

$$f16PropGain = K_{sc} \cdot 2^{-i16PropGainShift} \qquad \textbf{Eqn. 3-137}$$

$$f16PropGain = 1{,}8 \cdot 2^{(-1)} = 0{,}9 \qquad \textbf{Eqn. 3-138}$$

Result:

f16PropGain = 0.9

i16PropGainShift = 1

Note:

Ti > 4*Td

3.33.7 Returns

The function **GFLIB_ControllerPIDp** returns a fractional value as result of the PID algorithm. The value returned by the algorithm is in the following range:

$$f16LowerLimit \le PIDresult \le f16UpperLimit$$

3.33.8 Range Issues

The PID controller parameters are in the following range

$$0 \le f16PropGain < 1$$

$$0 \le f16IntegGain < 1$$

$$0 \le f16DerGain < 1$$

$$0 \le i16PropGainShift < 14$$

$$0 \le i16IntegGainShift < 14$$

$$0 \le i16DerGainShift < 14$$

3.33.9 Special Issues

The function **GFLIB_ControllerPIDp** is the saturation mode independent.

3.33.10 Implementation

Example 3-33. Implementation Code

```
#include "gflib.h"

static Frac16 mf16DesiredValue;
static Frac16 mf16MeasuredValue;
static Frac16 mf16ErrorK;
static Frac16 mf16DErrorK;
static Frac16 mf16DErrorK_1;
static Int16 mi16SatFlag;
static Frac16 mf16ControllerOutput;

/* Controller parameters */
static GFLIB_CONTROLLER_PID_P_PARAMS_T mudtControllerParam;

void Isr(void);

void main(void)
{
        /* Controller parameters initialization */
        mudtControllerParam.f16PropGain = FRAC16(0.5);
        mudtControllerParam.f16IntegGain = FRAC16(0.032);
        mudtControllerParam.f16DerGain = FRAC16(0.01);
        mudtControllerParam.i16PropGainShift = 1;
        mudtControllerParam.i16IntegGainShift = 0;
        mudtControllerParam.i16DerGainShift = 0;
        mudtControllerParam.f32IntegPartK_1 = 0;
        mudtControllerParam.f16UpperLimit = FRAC16(0.8);
        mudtControllerParam.f16LowerLimit = FRAC16(-0.7);
        mf16DErrorK_1 = 0;

        /* Desired value initialization */
        mf16DesiredValue = FRAC16(0.5);

        /* Measured value initialization */
        mf16MeasuredValue = 0;

        /* Saturation flag initialization */
        mi16SatFlag = 0;
}

/* Periodical function or interrupt */
void Isr(void)
{
        /* Error calculation */
        mf16ErrorK = mf16DesiredValue - mf16MeasuredValue;
        mf16ErrorK = mf16DErrorK;

        /* Controller calculation */
        mf16ControllerOutput = GFLIB_ControllerPIDp(mf16ErrorK,
mf16ErrorK, &mudtControllerParam, &mi16SatFlag, &mf16DErrorK_1);
```

```
}
```

3.33.11 See Also

See **GFLIB_ControllerPIp**, **GFLIB_ControllerPIr**, **GFLIB_ControllerPIrLim** and **GFLIB_ControllerPIDr** for more information.

3.33.12 Performance

Table 3-88. Performance of GFLIB_ControllerPIDp Function

Code Size (words)	74	
Data Size (words)	0	
Execution Clock	Min	93/89 cycles
	Max	101/98 cycles

3.34 GFLIB_ControllerPIDr

The function calculates the recurrent form of the Proportional-Integral-Derivative (PID) regulator.

3.34.1 Synopsis

```
#include "gflib.h"
Frac16 GFLIB_ControllerPIDr(Frac16 f16Error,
GFLIB_CONTROLLER_PID_RECURRENT_T * const pudtCtrl)
```

3.34.2 Prototype

```
asm Frac16 GFLIB_ControllerPIDRecurrentFAsm(Frac16 f16Err,
GFLIB_CONTROLLER_PID_RECURRENT_T * const pudtCtrl)
```

3.34.3 Arguments

Table 3-89. Function Arguments

Name	In/Out	Format	Range	Description
f16Error	In	SF16	0x8000 ... 0x7FFF	Error as input argument; the Frac16 data type is defined in header file GFLIB_types.h
*pudtCtrl	In/Out	struct	N/A	Pointer to a controller structure, which contains controller coefficients and integrator delay line; the GFLIB_CONTROLLER_PID_RECURRENT_T data type is defined in header file GFLIB_ControllerPIDRecurrentAsm.h

Table 3-90. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GFLIB_CONTROLLER_PID_RECURRENT_T	f32Acc	In/Out	SF32	0x80000000... 0x7FFFFFFF	Internal controller accumulator
	f16ErrorK_1	In/Out	SF16	0x8000 ... 0x7FFF	Input error at step k-1; delayed value of error at step k
	f16CC1Sc	In	SF16	0x8000 ... 0x7FFF	First controller coefficient scaled to fractional format
	f16CC2Sc	In	SF16	0x8000 ... 0x7FFF	Second controller coefficient scaled to fractional format
	f16CC3Sc	In	SF16	0x8000 ... 0x7FFF	Third controller coefficient scaled to fractional format
	ui16NShift	In	UI16	0...15	Scaling shift

3.34.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.34.5 Dependencies

List of all dependent files:

- GFLIB_ControllerPIDRecurrentAsm.h
- GFLIB_types.h

3.34.6 Description

The time continuos function of the PID controller is defined as

$$u(t) = K_P \cdot e(t) + K_I \cdot \int_0^t e(t)dt + K_D \cdot \frac{de(t)}{t} \qquad \textbf{Eqn. 3-139}$$

Different techniques can be used to convert the continuous PID controller function into the discrete representation. However, the continuous function can only be approximated and the discrete representation can never be exactly equivalent. Different methods can result different controller performances.

The resulting difference equation derived by the discretization method is in the form as reported below

$$u(k) = u(k-1) + CC1 \cdot e(k) + CC2 \cdot e(k-1) + CC3 \cdot e(k-2) \qquad \textbf{Eqn. 3-140}$$

The transition from the continuous to the discrete time domain reveals the following controller coefficients

Table 3-91. Controller Coefficients

Controller Coefficients	Integration with Bilinear Transformation Derivation with Backward Rectangular	Integration with Backward Rectangular Derivation with Backward Rectangular
CC1	$K_P + K_I \cdot T_S/2 + K_D/T_S$	$K_P + K_I \cdot T_S + K_D/T_S$
CC2	$-K_P + K_I \cdot T_S/2 - 2K_D/T_S$	$-K_P - 2K_D/T_S$
CC3	K_D/T_S	K_D/T_S

where

- K_P - proportional gain
- K_I - integral gain
- K_D - derivation gain

- T_S - sampling period

The discrete time domain representation of the recurrent PI algorithm scaled into the fractional range as

$$f16Uk = f32Acc + f16CC1 \cdot f16Error + f16CC2 \cdot \mathrm{f16ErrorK\_1} + \mathrm{f16CC3} \cdot \mathrm{f16ErrorK\_2}$$

Eqn. 3-141

where f32Acc is the accumulated controller portion over time and is used as an internal variable of this algorithm. f16CC1, f16CC2 and f16CC3 are the recurrent controller coefficients which are adapted as

$$f16CC1 = CC1 \cdot \frac{Emax}{Umax}$$ **Eqn. 3-142**

$$f16CC2 = CC2 \cdot \frac{Emax}{Umax}$$ **Eqn. 3-143**

$$f16CC3 = CC3 \cdot \frac{Emax}{Umax}$$ **Eqn. 3-144**

The input for the recurrent PI controller is considered f16Error which is adapted as

$$\mathrm{f16Error} = \mathrm{f16Desired} - \mathrm{f16Actual}$$ **Eqn. 3-145**

$$\mathrm{f16Error} = \frac{\mathrm{Error}}{\mathrm{EMax}}$$ **Eqn. 3-146**

where f16Error is a fractional number which must be within the fractional range <-1, 0.9999>. The f16Error value is processed within the PID controller algorithm and the delayed values are stored in f16Error_1, f16Error_2 for the next calculation. The operation of the delaying error values is performed internally by the algorithm itself and any user action is not required.

For proper operation of the recurrent PID controller implemented on the 16/32-bit DSC a care must be taken due to fixed point representation of individual values. All coefficients need to be represented as 16-bit fixed point numbers. The input value f16Error is assumed to be already in the correct 16-bit fixed point number format. Other controller variables as f32Acc, f16Error16K_1, f16Error16K_2 are internally handled by the algorithm of the recurrent PID controller. The controller coefficients f16CC1, f16CC2, and f16CC3 must be prepared in the correct 16-bit fixed point number format by the user. A scaling shift ui16NShift is introduced to be scaled to the 16-bit fixed point format. Then the fractional representation on the DSC of the recurrent PID controller is calculated by the following formula as

Eqn. 3-147

$$f16Uk = f16Acc + f16CC1Sc \cdot f16Error + f16CC2Sc \cdot f16ErrorK1 + f16CC3Sc \cdot f16ErrorK2$$

where

$$f16CC1Sc = f16CC1 \cdot 2^{-ui16NShift}$$ **Eqn. 3-148**

$$f16CC2Sc = f16CC2 \cdot 2^{-ui16NShift}$$ **Eqn. 3-149**

$$f16CC3Sc = f16CC3 \cdot 2^{-ui16NShift}$$ **Eqn. 3-150**

ui16NShift is chosen that the coefficients reside within the common range <-1, 0.9999>.

$$ui16NShift = \max\left(\mathrm{ceil}\left(\frac{\log(f16CC1)}{\log(2)}\right), \mathrm{ceil}\left(\frac{\log(f16CC2)}{\log(2)}\right), \mathrm{ceil}\left(\frac{\log(f16CC3)}{\log(2)}\right)\right)$$ **Eqn. 3-151**

In addition, ui16NShift is chosen as a power of 2, then the final de-scaling is a simple shift operation.

3.34.7 Returns

The function returns a 16-bit fractional value as result of calculation of the PID algorithm.

3.34.8 Range Issues

The PID controller parameters are in the following range

$$-1 \le f16ErrorK\_1 \le 0{,}9999$$ **Eqn. 3-152**

$$-1 \le f16ErrorK\_2 \le 0{,}9999$$ **Eqn. 3-153**

$$-1 \le f16CC1Sc \le 0{,}9999$$ **Eqn. 3-154**

$$-1 \le f16CC2Sc \le 0{,}9999$$ **Eqn. 3-155**

$$-1 \le f16CC3Sc \le 0{,}9999$$ **Eqn. 3-156**

$$0 \le ui16NShift < 16$$ **Eqn. 3-157**

3.34.9 Special Issues

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

3.34.10 Implementation

Example 3-34. Implementation Code

```
#include "gflib.h"
```

```
static Frac16 mf16DesiredValue;
static Frac16 mf16MeasuredValue;
static Frac16 mf16ErrorK;
static Frac16 mf16ControllerOutput;


/* Controller parameters */
static GFLIB_CONTROLLER_PID_RECURRENT_T mudtControllerParam;

void Isr(void);

void main(void)
{
        /* Controller parameters initialization */
        mudtControllerParam.f16CC1Sc = FRAC16(0.527083333);
        mudtControllerParam.f16CC2Sc = FRAC16(-0.514583333);
        mudtControllerParam.f16CC3Sc = FRAC16(-0.514583333);
        mudtControllerParam.ui16NShift = 3;
        mudtControllerParam.f16ErrorK_1 = 0;
        mudtControllerParam.f16ErrorK_2 = 0;
        mudtControllerParam.f32Acc = 0;

        /* Desired value initialization */
        mf16DesiredValue = FRAC16(0.5);

        /* Measured value initialization */
        mf16MeasuredValue = 0;
}

/* Periodical function or interrupt */
void Isr(void)
{
        /* Error calculation */
        mf16ErrorK = mf16DesiredValue - mf16MeasuredValue;

        /* Controller calculation */
        mf16ControllerOutput = GFLIB_ControllerPIDr(mf16ErrorK,
&mudtControllerParam);
}
```

3.34.11 See Also

See **GFLIB_ControllerPIp**, **GFLIB_ControllerPIr**, **GFLIB_ControllerPIrLim** and **GFLIB_ControllerPIDp** for more information.

3.34.12 Performance

Table 3-92. Performance of GFLIB_ControllerPIDr function

Code Size (words)	22	
Data Size (words)	0	
Execution Clock	Min	38/36 cycles
	Max	38/36 cycles

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics of their non-RoHS-compliant and/or non-Pb-free counterparts. For further information, see http://www.freescale.com or contact your Freescale sales representative.

For information on Freescale's Environmental Products program, go to http://www.freescale.com/epp.

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.
© Freescale Semiconductor, Inc. 2009. All rights reserved.

56800E_GFLIB
Rev. 3, 06/2011