General Functions Library

User Reference Manual

Rev. 0
01/2009

freescale™
semiconductor

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

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)
www.freescale.com/support

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

Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

XXXXRM
Rev. 0
01/2009

Chapter 1 License Agreement

Chapter 2 Introduction

Chapter 3 Function API

Chapter 4
Get Upper Word of Argument

Chapter 5
Round Argument

Chapter 6
Deposit a 16-Bit Argument to Upper 16-Bit of 32 Bits

Chapter 7
Arithmetic Shift of a 32-bit Argument by Number of Shifts

Chapter 8
Arithmetic Shift of a 32-Bit Argument by Number of Shifts with Saturation

Chapter 9
Calculating a 16-Bit Fractional Product of Two 16-Bit Fractional Numbers

Chapter 10
Calculating a 32-Bit Fractional Product of Two 16-Bit Fractional Numbers

Chapter 11
Calculating a 32-Bit Fractional Product of a 32-Bit and a 16-Bit Fractional Number

Chapter 12
Calculating the Division of a 32-Bit Fractional Argument by a 16-Bit Fractional Number with a 16-Bit Fractional Result

Chapter 13
Calculating the Division of a 32-Bit Unsigned Argument by a 16-Bit Unsigned Number with a 16-Bit Unsigned Result

Chapter 14
Calculating the Sine Value Using the Lookup Table

Chapter 15
Calculating the Cosine Value Using the Lookup Table

Chapter 16
Calculating the Square Root Value of the Argument

Chapter 17
Calculating a 16-Bit Version of the Up/Down Ramp

Chapter 18
Calculating a 32-Bit Version of the Up/Down Ramp

Chapter 19
Calculating a 16-Bit Scalar Up/Down Limitation of the Input Signal

Chapter 20
Calculating a 32-Bit Scalar Up/Down Limitation of the Input Signal

Chapter 21
Calculating a Parallel Form of the Proportional-Integral Regulator

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 MCF51xx family of microcontrollers. This library contains optimized functions for the ColdFire V1 core. The library is supplied in a binary form, which is unique by its simplicity to integrate with the user application.

2.2 Supported Compilers

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

- CodeWarrior™ Development Studio V6.1 ColdFireV1 AC256 ALPHA Service Pack

The library is delivered in the library module GFLIB_MCF51.lib and is intended for use in small data memory model projects. The interfaces to the algorithms, included in this library, have been combined into a single public interface include file, gflib.h. This was done to simplify 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 General Functions Library tool is installed while the 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 GFLIB_MCF51_rX.X, where X.X denotes the actual release number. This way of installing the library 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, take the following steps:

1. Execute GFLIB_MCF51_rXX.exe.
2. Follow the General Functions Library software installation instructions on your screen.

2.4 Library Integration

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

1. Create a new empty project.
2. Create the GFLIB group in your new open project. Note that this step is not mandatory, it is mentioned here just for the purpose of maintaining the file consistency in the CodeWarrior project window. In the CodeWarrior menu, choose Project > Create Group..., type GFLIB in the dialog window that pops up, and click OK.
3. Reference the GFLIB_MCF51.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 also 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 in the project window. 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.

Arguments

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

Description

This subsection is a description of the function or macro. It explains the 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 the 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 the illustration purposes.

See Also

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

Performance

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

2.6 Data Types

The 16-bit DSC core supports four types of two-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 the 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:

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

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

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

2.6.2 Unsigned Integer (UI)

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

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

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

2.6.3 Signed Fractional (SF)

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

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

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:

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

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 gflib_types.h

Mnemonics	Size - bits	Description
Word8	8	to represent an 8-bit signed variable/value
UWord8	8	to represent an 8-bit signed variable/value
Word16	16	to represent a 16-bit signed variable/value
UWord16	16	to represent a 16-bit signed variable/value
Word32	32	to represent a 32-bit signed variable/value
UWord32	32	to represent a 32-bit signed variable/value
Int8	8	to represent an 8-bit signed variable/value
UInt8	8	to represent an 8-bit signed variable/value
Int16	16	to represent a 16-bit signed variable/value
UInt16	16	to represent a 16-bit signed variable/value
Int32	32	to represent a 32-bit signed variable/value
UInt32	32	to represent a 32-bit signed variable/value
Frac16	16	to represent a 16-bit signed variable/value
Frac32	32	to represent a 32-bit signed variable/value
NULL	constant	represents the NULL pointer
bool	16	boolean variable
false	constant	represents a false value
true	constant	represents a true value
FRAC16()	macro	transforms the float value from <–1, 1) range into the fractional representation <–32768, 32767>
FRAC32()	macro	transforms the float value from <–1, 1) range into the fractional representation <–2147483648, 2147483648>

2.8 Special Issues

All functions in the General Functions Library are implemented without storing any of the volatile registers (D0, D1, D2, A0, A1) used by the respective routine. Only non-volatile registers (D3, D4, D5, D6, D7, A2, A3, A4, A5) are saved by pushing the registers on the stack. Because 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_UpperWordi	Frac32 f32In	Frac16	This function gets the upper 16 bits of the 32-bit argument.
GFLIB_Roundi	Frac32 f32In	Frac16	This function gets the upper 16 bits of the 32-bit argument and rounds it.
GFLIB_DepositHi	Frac16 f16In	Frac32	This function deposits the 16-bit argument to the position of upper 16 bits of a 32-bit variable.
GFLIB_ArShftLftLi	Frac32 f32In Word16 w16Shift	Frac32	This function arithmetically shifts a 32-bit argument by a number of shifts, defined by the argument.
GFLIB_ArShftLftSatL	Frac32 f32In Word16 w16Shift	Frac32	This function arithmetically shifts a 32-bit argument by a number of shifts, defined by the argument. At the same time the overflow is checked, and in case of overflow the output is saturated.
GFLIB_MulsSSSi	Frac16 f16InA Frac16 f16InB	Frac16	This function calculates a 16-bit signed fractional product of two 16-bit signed fractional numbers.
GFLIB_MulsSSLi	Frac16 f16InA Frac16 f16InB	Frac32	This function calculates a 32-bit signed fractional product of two 16-bit signed fractional numbers.
GFLIB_MulsLSLi	Frac32 f32InA Frac16 f16InB	Frac32	This function calculates a 32-bit signed fractional product of a 32-bit signed fractional number and a 16-bit signed fractional number.
GFLIB_DivsLSS	Frac32 f32Dividend Frac16 f16Divisor	Frac16	This function divides a 32-bit signed fractional argument by a 16-bit signed fractional number, and returns a 16-bit signed fractional result.
GFLIB_DivuLSS	UWord32 uw32Dividend UWord16 uw16Divisor	UWord16	This function divides a 32-bit unsigned argument by a 16-bit unsigned number, and returns a 16-bit unsigned result.
GFLIB_SinLut	Frac16 f16Arg	Frac16	This function calculates the sine value of the argument using the lookup table.
GFLIB_CosLut	Frac16 f16Arg	Frac16	This function calculates the sine value of the argument using the lookup table.
GFLIB_SqrtPoly	Frac32 f32Arg	Frac16	This function calculates the square root value of the argument using piece-wise polynomial approximation with post adjustment method.

Table 3-1. API Functions Summary (continued)

GFLIB_Ramp16	Frac16 f16Desired Frac16 f16Actual GFLIB_RAMP_T *pudtParam	Frac16	This function calculates a 16-bit version of up/down ramp with step increment/decrement defined in pParam structure.
GFLIB_Ramp32	Frac32 f32Desired Frac32 f32Actual GFLIB_RAMP32_T *pudtParam	Frac32	This function calculates a 32-bit version of up/down ramp with step increment/decrement defined in pParam structure.
GFLIB_Limit16	Frac16 f16Arg GFLIB_LIMIT_T *pudtLimit	Frac16	This function calculates a 16-bit scalar upper/lower limitation of the input signal.
GFLIB_Limit32	Frac32 f32Arg GFLIB_LIMIT32_T *pudtLimit	Frac32	This function calculates a 32-bit scalar upper/lower limitation of the input signal.
GFLIB_ControllerPIp	Frac16 f16InputErrorK GFLIB_CONTROLLER_PI_P_PARAMS_T *pudtPiParams Int16 *pi16SatFlag	Frac16	This function calculates the parallel form of the proportional-integral (PI) regulator.

Chapter 4
Get Upper Word of Argument

4.1 GFLIB_UpperWordi

This function gets the upper 16 bits of the 32-bit argument.

4.1.1 Synopsis

```
#include ”gflib.h”
Frac16 GFLIB_UpperWordi(Frac32 f32In)
```

4.1.2 Arguments

Table 4-1. Function Arguments

Name	In/Out	Format	Range	Description
f32In	In	SF32	0x80000000... 0x7FFFFFFF	Input argument; the Frac32 data type is defined in the header file GFLIB_types.h.

4.1.3 Availability

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

This library module is targeted at the MCF51xx platform.

4.1.4 Dependencies

List of all dependent files:

- GFLIB_SimpleMathAsm.h
- GFLIB_types.h

4.1.5 Description

The function **GFLIB_UpperWordi** gets the upper 16 bits of the 32-bit argument and returns it as a 16-bit word variable. The function does not round the result.

4.1.6 Returns

The function **GFLIB_UpperWordi** returns the upper 16-bit word of the 32-bit argument without any rounding.

4.1.7 Range Issues

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

4.1.8 Special Issues

None.

4.1.9 Implementation

The **GFLIB_UpperWordi** function is implemented as an inline function.

Example 4-1.

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

#include "gflib.h"

static Frac32 mf32Input;
static Frac16 mf16Output;

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

        /* Get the upper word */
        mf16Output = GFLIB_UpperWordi(mf32Input);
}
```

4.1.10 See Also

See **GFLIB_DepositHi** and **GFLIB_Roundi** for more information.

4.1.11 Performance

Table 4-2. Performance of the GFLIB_UpperWordi Function

Code Size (bytes)	10 bytes
Data Size (bytes)	0 bytes

Table 4-2. Performance of the GFLIB_UpperWordi Function (continued)

Execution Clock	Min	4 cycles
	Max	4 cycles

Chapter 5
Round Argument

5.1 GFLIB_Roundi

This function gets the upper 16 bits of the 32-bit argument and rounds it.

5.1.1 Synopsis

```
#include ”gflib.h”
Frac16 GFLIB_Roundi(Frac32 f32In)
```

5.1.2 Arguments

Table 5-1. Function Arguments

Name	In/Out	Format	Range	Description
f32In	In	SF32	0x80000000... 0x7FFFFFFF	Input argument; the Frac32 data type is defined in the header file GFLIB_types.h.

5.1.3 Availability

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

This library module is targeted for the MCF51xx platform.

5.1.4 Dependencies

List of all dependent files:

- GFLIB_SimpleMathAsm.h
- GFLIB_types.h

5.1.5 Description

The function **GFLIB_Roundi** gets the upper 16 bits of the 32-bit argument, and rounds it according to the highest bit from the lower 16 bits of the argument.

5.1.6 Returns

The function **GFLIB_Roundi** returns the upper 16-bit word of the 32-bit argument with rounding.

5.1.7 Range Issues

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

5.1.8 Special Issues

None.

5.1.9 Implementation

The **GFLIB_Roundi** function is implemented as an inline function.

Example 5-1.

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

#include "gflib.h"

static Frac32 mf32Input;
static Frac16 mf16Output;

void main(void)
{
        /* input value 0.55 */
        mf32Input = FRAC32(0.55);

        /* Round the 32-bit variable to 16 bits */
        mf16Output = GFLIB_Roundi(mf32Input);
}
```

5.1.10 See Also

See **GFLIB_UpperWordi** and **GFLIB_DepositHi** for more information.

5.1.11 Performance

Table 5-2. Performance of the GFLIB_Roundi Function

Code Size (bytes)	24 bytes	
Data Size (bytes)	0 bytes	
Execution Clock	Min	7 cycles
	Max	7 cycles

Chapter 6
Deposit a 16-Bit Argument to Upper 16-Bit of 32 Bits

6.1 GFLIB_DepositHi

This function deposits the 16-bit argument to the position of upper 16 bits of a 32-bit variable.

6.1.1 Synopsis

```
#include ”gflib.h”
Frac32 GFLIB_DepositHi(Frac16 f16In)
```

6.1.2 Arguments

Table 6-1. Function Arguments

Name	In/Out	Format	Range	Description
f16In	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in the header file GFLIB_types.h.

6.1.3 Availability

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

This library module is targeted at the MCF51xx platform.

6.1.4 Dependencies

List of all dependent files:

- GFLIB_SimpleMathAsm.h
- GFLIB_types.h

6.1.5 Description

The function **GFLIB_DepositHi** deposits the 16-bit argument to the position of the upper 16 bits of the 32-bit variable. The lower 16 bits of the results are zeroed.

6.1.6 Returns

The function **GFLIB_DepositHi** returns a 32-bit variable, where the upper 16 bits are formed by the argument and the lower are cleared.

6.1.7 Range Issues

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

6.1.8 Special Issues

None.

6.1.9 Implementation

The **GFLIB_DepositHi** function is implemented as an inline function.

Example 6-1.

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

#include "gflib.h"

static Frac16 mf16Input;
static Frac32 mf32Output;

void main(void)
{
        /* input value 0.55 */
        mf16Input = FRAC16(0.55);

        /* Deposits the 16-bit variable to the 32-bit variable */
        mf32Output = GFLIB_DepositHi(mf16Input);
}
```

6.1.10 See Also

See **GFLIB_UpperWordi** and **GFLIB_Roundi** for more information.

6.1.11 Performance

Table 6-2. Performance of the GFLIB_DepositHi Function

Code Size (bytes)	10 bytes
Data Size (bytes)	0 bytes

Table 6-2. Performance of the GFLIB_DepositHi Function (continued)

Execution Clock	Min	4 cycles
	Max	4 cycles

Chapter 7
Arithmetic Shift of a 32-bit Argument by Number of Shifts

7.1 GFLIB_ArShftLftLi

This function arithmetically shifts a 32-bit argument by a number of shifts, defined by the argument.

7.1.1 Synopsis

```
#include ”gflib.h”
Frac32 GFLIB_ArShftLftLi(Frac32 f32In, Word16 w16Shift)
```

7.1.2 Arguments

Table 7-1. Function Arguments

Name	In/Out	Format	Range	Description
f32In	In	SF32	0x80000000... 0x7FFFFFFF	Input argument; the Frac32 data type is defined in the header file GFLIB_types.h.
w16Shift	In	SI16	-31...31	Input argument; the Word16 data type is defined in the header file GFLIB_types.h.

7.1.3 Availability

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

This library module is targeted at the MCF51xx platform.

7.1.4 Dependencies

List of all dependent files:

- GFLIB_ShiftAsm.h
- GFLIB_types.h

7.1.5 Description

The function **GFLIB_ArShftLftLi** shifts a 32-bit argument arithmetically by the number of shifts, defined by the w16Shift argument. If the number is positive, the shift is performed to the left; if it is positive, the shift is performed to the right.

7.1.6 Returns

The function **GFLIB_ArShftLftLi** returns a 32-bit variable, representing the f32In argument shifted by the number of the w16Shift argument.

7.1.7 Range Issues

The input data value f32In is in the range of <–1, 1), expressed with a 32-bit precision. The input shift number w16Shift is in the valid <–31, 31>, expressed by a 16-bit variable. The output data value is in the range of <–1, 1) expressed with a 32-bit precision.

7.1.8 Special Issues

None.

7.1.9 Implementation

The **GFLIB_ArShftLftLi** function is implemented as an inline function.

Example 7-1.

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

#include "gflib.h"

static Frac32 mf32Input;
static Frac32 mf32Output;
static Word16 mw16Shift;

void main(void)
{
        /* 3 shifts to the left */
        mw16Shift = 3;

        /* input value 0.55 */
        mf32Input = FRAC32(0.55);

        /* Shifts the mf32Input variable by the number of mw16Shift shifts */
        mf32Output = GFLIB_ArShftLftLi(mf32Input, mw16Shift);
}
```

7.1.10 See Also

See **GFLIB_ArShftLftSatL** for more information.

7.1.11 Performance

Table 7-2. Performance of the GFLIB_ArShftLftLi Function

Code Size (bytes)	18 bytes	
Data Size (bytes)	0 bytes	
Execution Clock	Min	7 cycles
	Max	7 cycles

Chapter 8
Arithmetic Shift of a 32-Bit Argument by Number of Shifts with Saturation

8.1 GFLIB_ArShftLftSatL

This function arithmetically shifts a 32-bit argument by a number of shifts, defined by the argument. At the same time the overflow is checked, and in the case of overflow the output is saturated.

8.1.1 Synopsis

```
#include ”gflib.h”
Frac32 GFLIB_ArShftLftSatL(Frac32 f32In, Word16 w16Shift)
Frac32 GFLIB_ArShftLftSatLi(Frac32 f32In, Word16 w16Shift) - inline version
```

8.1.2 Arguments

Table 8-1. Function Arguments

Name	In/Out	Format	Range	Description
f32In	In	SF32	0x80000000... 0x7FFFFFFF	Input argument; the Frac32 data type is defined in the header file GFLIB_types.h.
w16Shift	In	SI16	-31...31	Input argument; the Word16 data type is defined in the header file GFLIB_types.h.

8.1.3 Availability

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

This library module is targeted at the MCF51xx platform.

8.1.4 Dependencies

List of all dependent files:

- GFLIB_ShiftAsm.h
- GFLIB_types.h

8.1.5 Description

The function **GFLIB_ArShftLftSatL** shifts a 32-bit argument arithmetically by the number of shifts, defined by the w16Shift argument. If the number is positive, the shift is performed to the left; if it is positive, the shift is performed to the right. This function checks if the number overflows, and in the case of overflow, the number saturates.

8.1.6 Returns

The function **GFLIB_ArShftLftSatL** returns a 32-bit variable representing the f32In argument shifted by the number of the w16Shift argument.

8.1.7 Range Issues

The input data value f32In is in the range of <–1, 1), expressed with a 32-bit precision. The input shift number w16Shift is in the valid <–31, 31>, expressed by a 16-bit variable. The output data value is in the range of <–1, 1), expressed with a 32-bit precision.

8.1.8 Special Issues

None.

8.1.9 Implementation

The **GFLIB_ArShftLftSatL** function is implemented as a function call, and also an inline function.

Example 8-1.

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

#include "gflib.h"

static Frac32 mf32Input;
static Frac32 mf32Output;
static Word16 mw16Shift;

void main(void)
{
        /* 3 shifts to the left */
        mw16Shift = 3;

        /* input value 0.55 */
        mf32Input = FRAC32(0.55);

        /* Shifts the mf32Input variable by the number of mw16Shift shifts */
        mf32Output = GFLIB_ArShftLftSatLi(mf32Input, mw16Shift);
}
```

8.1.10 See Also

See **GFLIB_ArShftLftLi** for more information.

8.1.11 Performance

Table 8-2. Performance of the GFLIB_ArShftLftSatL Function

Code Size (bytes)	50 (54 inline) bytes	
Data Size (bytes)	0 bytes	
Execution Clock	Min	12 (9 inline) cycles
	Max	21 (17 inline) cycles

Chapter 9
Calculating a 16-Bit Fractional Product of Two 16-Bit Fractional Numbers

9.1 GFLIB_MulsSSSi

This function calculates a 16-bit signed fractional product of two 16-bit signed fractional numbers.

9.1.1 Synopsis

```
#include ”gflib.h”
Frac16 GFLIB_MulsSSSi(Frac16 f16InA, Frac16 f16InB)
```

9.1.2 Arguments

Table 9-1. Function Arguments

Name	In/Out	Format	Range	Description
f16InA	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in the header file GFLIB_types.h.
f16InB	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in the header file GFLIB_types.h.

9.1.3 Availability

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

This library module is targeted at the MCF51xx platform.

9.1.4 Dependencies

List of all dependent files:

- GFLIB_MulAsm.h
- GFLIB_types.h

9.1.5 Description

The function **GFLIB_MulsSSSi** multiplies the two 16-bit arguments; their result is shifted to the left by one. The upper 16 bits are the output of the function. The result does not overflow, but can be saturated. Look at the following equation:

$$\text{Output}(16) = \frac{[\text{f16In}(\text{A}(16) \times \text{f16InB}(16))] \times 2}{65536} \qquad \textit{Eqn. 9-1}$$

9.1.6 Returns

The function **GFLIB_MulsSSSi** returns a 16-bit signed fractional product of two 16-bit signed fractional arguments.

9.1.7 Range Issues

The input data values f16InA and f16InB are in the range of <–1, 1), expressed with a 16-bit precision. The output data value is in the range of <–1, 1), expressed with a 16-bit precision.

9.1.8 Special Issues

None.

9.1.9 Implementation

The **GFLIB_MulsSSSi** function is implemented as an inline function.

Example 9-1.

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

#include "gflib.h"

static Frac16 mf16A;
static Frac16 mf16B;
static Frac16 mf16C;

void main(void)
{
        /* input A value 0.55 */
        mf16A = FRAC16(0.55);

        /* input B value 0.75 */
        mf16B = FRAC16(0.75);

        /* Multiplies the numbers */
        mf16C = GFLIB_MulsSSSi(mf16A, mf16B);
}
```

9.1.10 See Also

See **GFLIB_MulsLSLi** and **GFLIB_MulsSSLi** for more information.

9.1.11 Performance

Table 9-2. Performance of the GFLIB_MulsSSSi Function

Code Size (bytes)	16 bytes	
Data Size (bytes)	0 bytes	
Execution Clock	Min	12 cycles
	Max	12 cycles

Chapter 10
Calculating a 32-Bit Fractional Product of Two 16-Bit Fractional Numbers

10.1 GFLIB_MulsSSLi

This function calculates a 32-bit signed fractional product of two 16-bit signed fractional numbers.

10.1.1 Synopsis

```
#include ”gflib.h”
Frac32 GFLIB_MulsSSLi(Frac16 f16InA, Frac16 f16InB)
```

10.1.2 Arguments

Table 10-1. Function Arguments

Name	In/Out	Format	Range	Description
f16InA	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in the header file GFLIB_types.h.
f16InB	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in the header file GFLIB_types.h.

10.1.3 Availability

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

This library module is targeted at the MCF51xx platform.

10.1.4 Dependencies

List of the dependent files:

- GFLIB_MulAsm.h
- GFLIB_types.h

10.1.5 Description

The function **GFLIB_MulsSSLi** multiplies the two 16-bit arguments; their result is shifted to the left by one. The 32-bit result is the output of the function. The result does not overflow but can be saturated. See the following equation:

$$\text{Output}(32) = [\text{f16In}(\text{A}(16) \times \text{f16InB}(16))] \times 2 \qquad \textit{Eqn. 10-1}$$

10.1.6 Returns

The function **GFLIB_MulsSSLi** returns a 32-bit signed fractional product of two 16-bit signed fractional arguments.

10.1.7 Range Issues

The input data values f16InA and f16InB are in the range of <–1, 1), expressed with a 16-bit precision. The output data value is in the range of <–1, 1), expressed with a 32-bit precision.

10.1.8 Special Issues

None.

10.1.9 Implementation

The **GFLIB_MulsSSLi** function is implemented as an inline function.

Example 10-1.

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

#include "gflib.h"

static Frac16 mf16A;
static Frac16 mf16B;
static Frac32 mf32C;

void main(void)
{
        /* input A value 0.55 */
        mf16A = FRAC16(0.55);

        /* input B value 0.75 */
        mf16B = FRAC16(0.75);

        /* Multiplies the numbers */
        mf32C = GFLIB_MulsSSLi(mf16A, mf16B);
}
```

10.1.10 See Also

See **GFLIB_MulsSSSi** and **GFLIB_MulsLSLi** for more information.

10.1.11 Performance

Table 10-2. Performance of the GFLIB_MulsSSLi Function

Code Size (bytes)	12 bytes	
Data Size (bytes)	0 bytes	
Execution Clock	Min	11 cycles
	Max	11 cycles

Chapter 11
Calculating a 32-Bit Fractional Product of a 32-Bit and a 16-Bit Fractional Number

11.1 GFLIB_MulsLSLi

This function calculates a 32-bit signed fractional product of a 32-bit signed fractional number and a 16-bit signed fractional number.

11.1.1 Synopsis

```
#include ”gflib.h”
Frac32 GFLIB_MulsLSLi(Frac32 f32InA, Frac16 f16InB)
```

11.1.2 Arguments

Table 11-1. Function Arguments

Name	In/Out	Format	Range	Description
f32InA	In	SF32	0x80000000... 0x7FFFFFFF	Input argument; the Frac32 data type is defined in the header file GFLIB_types.h.
f16InB	In	SF16	0x8000... 0x7FFF	Input argument; the Frac16 data type is defined in the header file GFLIB_types.h.

11.1.3 Availability

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

This library module is targeted at the MCF51xx platform.

11.1.4 Dependencies

List of all dependent files:

- GFLIB_MulAsm.h
- GFLIB_types.h

11.1.5 Description

The function **GFLIB_MulsLSLi** partially multiplies a 32-bit argument with a 16-bit argument; their result is shifted to the left by one to reach 48 bits. The upper 32-bit result is the output of the function. The result does not overflow but can be saturated. Look at the following equation:

$$\text{Output}(32) = \frac{[\text{f32In}(\text{A}(32) \times \text{f16InB}(16))] \times 2}{65536} \qquad \textit{Eqn. 11-1}$$

The detail description of the calculation is as follows:

1. The lower unsigned 16-bits of the 32-bit f32InA argument are multiplicated with the signed 16-bit f16InB argument:

$$\text{Res1}(32) = \text{f32In}(\text{A}(16\text{L}) \times \text{f16InB}(16)) \qquad \textit{Eqn. 11-2}$$

2. The upper signed 16-bits of the 32-bit f32InA argument are multiplicated with the signed 16-bit f16InB argument:

$$\text{Res2}(32) = \text{f32In}(\text{A}(16\text{U}) \times \text{f16InB}(16)) \qquad \textit{Eqn. 11-3}$$

3. The first result, Res1, is shifted to the right by 15, and is added to the second result Res2; the final result is shifted to the left by one:

$$\text{Output}(32) = \left[\text{Res2}(32) + \frac{\text{Res1}(32)}{32768}\right] \times 2 \qquad \textit{Eqn. 11-4}$$

11.1.6 Returns

The function **GFLIB_MulsLSLi** returns a 32-bit signed fractional product of two 16-bit signed fractional arguments.

11.1.7 Range Issues

The input data values f16InA and f16InB are in the range of <–1, 1), expressed with a 16-bit precision. The output data value is in the range of <–1, 1), expressed with a 32-bit precision.

11.1.8 Special Issues

None.

11.1.9 Implementation

The **GFLIB_MulsLSLi** function is implemented as an inline function.

Example 11-1.

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

#include "gflib.h"

static Frac32 mf32A;
```

```
static Frac16 mf16B;
static Frac32 mf32C;

void main(void)
{
        /* input A value 0.55 */
        mf32A = FRAC32(0.55);

        /* input B value 0.75 */
        mf16B = FRAC16(0.75);

        /* Multiplies the numbers */
        mf32C = GFLIB_MulsLSLi(mf32A, mf16B);
}
```

11.1.10 See Also

See **GFLIB_MulsSSSi** and **GFLIB_MulsSSLi** for more information.

11.1.11 Performance

Table 11-2. Performance of the GFLIB_MulsLSLi Function

Code Size (bytes)	28 bytes	
Data Size (bytes)	0 bytes	
Execution Clock	Min	16 cycles
	Max	20 cycles

Chapter 12
Calculating the Division of a 32-Bit Fractional Argument by a 16-Bit Fractional Number with a 16-Bit Fractional Result

12.1 GFLIB_DivsLSS

The function divides a 32-bit signed fractional argument by a 16-bit signed fractional number and returns a 16-bit signed fractional result.

12.1.1 Synopsis

```
#include ”gflib.h”
Frac16 GFLIB_DivsLSS(Frac32 f32Dividend, Frac16 f16Divisor)
```

12.1.2 Arguments

Table 12-1. Function Arguments

Name	In/Out	Format	Range	Description
f32Dividend	In	SF32	0x80000000... 0x7FFFFFFF	Input divident argument; the Frac32 data type is defined in the header file GFLIB_types.h.
f16Divisor	In	SF16	0x8000... 0x7FFF	Input divisor argument; the Frac16 data type is defined in the header file GFLIB_types.h.

12.1.3 Availability

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

This library module is targeted at the MCF51xx platform.

12.1.4 Dependencies

List of all dependent files:

- GFLIB_DivAsm.h
- GFLIB_types.h

12.1.5 Description

The function **GFLIB_DivsLSS** fractionally divides a 32-bit argument by a 16-bit argument. The result is a fractional 16-bit number. If the divisor is zero, the result is the maximum 16-bit fractional number, where the sign is then copied from the dividend, otherwise the result is shifted to the right by one to be fractional.

$$\text{Output}(32) = \frac{\text{f32Dividend}(32)}{\text{f16InDivisor}(16) \times 2} \qquad \textit{Eqn. 12-1}$$

The core does not have any division support, so the division is made by subtraction and shifting and logical addition in a loop.

The principle is the following:

1. If the divisor is zero, the number is set to the maximum signed 16-bit fractional value with the sign of the dividend, and returns from the function.
2. Negative arguments are negated to be positive and the information is stored, if their signs are different from each other in order for the result to be negated at the end.
3. The shifter is set to 0b10000.
4. The difference in the number of leading bits between the divisor and dividend is calculated. The shifter 0b10000 and the divisor are shifted to the left by this number; in other words in the case of the negative difference, they are shifted to the right.
5. The intermediate result is cleared and the shifter is shifted to the right by one. If the shifter is zero, it jumps to the step number eight.
6. The dividend is compared to the divisor. If the dividend is greater than or equal to the divisor, the shifter valued is ORed to the intermediate result and the divisor is subtracted from the dividend.
7. The shifter and the divisor are shifted to the right by one, and if the shifter is greater than zero, it jumps to and repeats the cycle from the step number six.
8. If the dividend and the divisor have different signs, the intermediate result is negated.
9. If the intermediate result is greater than the maximum 16-bit number, it is set to the maximum number. If it is smaller than the minimum 16-bit number, it is set to the minimum number.

12.1.6 Returns

The function **GFLIB_DivsLSS** returns a 16-bit signed fractional result of a division of the 32-bit signed fractional argument by a 16-bit fractional argument.

12.1.7 Range Issues

The input data value f32Dividend is in the range of <–1, 1), expressed with a 32-bit precision. The input data value f16Divisor is in the range of <–1, 1), expressed with a 16-bit precision. The output data value is in the range of <–1, 1), expressed with a 16-bit precision.

12.1.8 Special Issues

None.

12.1.9 Implementation

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

Example 12-1.

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

#include "gflib.h"

static Frac32 mf32A;
static Frac16 mf16B;
static Frac16 mf16C;

void main(void)
{
        /* input A value 0.55 */
        mf32A = FRAC32(0.55);

        /* input B value 0.75 */
        mf16B = FRAC16(0.75);

        /* Divides mf32A by mf16B */
        mf16C = GFLIB_DivsLSS(mf32A, mf16B);
}
```

12.1.10 See Also

See **GFLIB_DivuLSS** for more information.

12.1.11 Performance

Table 12-2. Performance of the GFLIB_DivsLSS Function

Code Size (bytes)	156 bytes	
Data Size (bytes)	0 bytes	
Execution Clock	Min	15 cycles
	Max	166 cycles

Chapter 13
Calculating the Division of a 32-Bit Unsigned Argument by a 16-Bit Unsigned Number with a 16-Bit Unsigned Result

13.1 GFLIB_DivuLSS

This function divides a 32-bit unsigned argument by a 16-bit unsigned number, and returns a 16-bit unsigned result.

13.1.1 Synopsis

```
#include ”gflib.h”
UWord16 GFLIB_DivuLSS(UWord32 uw32Dividend, UWord16 uw16Divisor)
```

13.1.2 Arguments

Table 13-1. Function Arguments

Name	In/Out	Format	Range	Description
uw32Dividend	In	UI32	0x0... 0xFFFFFFFF	Input divident argument; the UWord32 data type is defined in the header file GFLIB_types.h.
uw16Divisor	In	UI16	0x0... 0xFFFF	Input divisor argument; the UWord16 data type is defined in the header file GFLIB_types.h.

13.1.3 Availability

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

This library module is targeted at the MCF51xx platform.

13.1.4 Dependencies

List of all dependent files:

- GFLIB_DivAsm.h
- GFLIB_types.h

13.1.5 Description

The function **GFLIB_DivuLSS** divides an unsigned 32-bit integer argument by an unsigned 16-bit argument. The result is an unsigned 16-bit number. If the divisor is zero, the result is the maximum unsigned 16-bit number.

$$\text{Output}(32) = \frac{\text{uw32Dividend}(32)}{\text{uw16Divisor}(16)} \qquad \textit{Eqn. 13-1}$$

The core does not have any division support, so the division is made by subtraction, shifting, and logical addition in a loop.

The principle is the following:

1. If the divisor is zero, the number is set to the maximum unsigned 16-bit integer value, and returned from the function.
2. The shifter is set to 0b10000.
3. The difference in the number of leading bits between the divisor and dividend is calculated. The shifter 0b10000 and the divisor are shifted to the left by this number; in other words in the case of the negative difference, they are shifted to the right.
4. The intermediate result is cleared. If the shifter is zero, it jumps to the step number seven.
5. The dividend is compared to the divisor. If the dividend is greater than or equal to the divisor, the shifter value is ORed to the intermediate result and the divisor is subtracted from the dividend.
6. The shifter and the divisor are shifted to the right by one, and if the shifter is greater than zero, it jumps to and repeats the cycle from the step number five.
7. If the inermediate result is greater than the maximum 16-bit number, it is set to the maximum number.

13.1.6 Returns

The function **GFLIB_DivuLSS** returns a 16-bit unsigned integer result of a division of the 32-bit unsigned integer argument by a 16-bit unsigned integer argument.

13.1.7 Range Issues

The input data value uw32Dividend is in the range of <0, 4294967295). The input data value uw16Divisor is in the range of <0, 65535). The output data value is in the range of <0, 65535).

13.1.8 Special Issues

None.

13.1.9 Implementation

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

Example 13-1.

```
#include <hidef.h> /* for EnableInterrupts macro */
```

```
#include "derivative.h" /* include peripheral declarations */

#include "gflib.h"

static UWord32 muw32A;
static UWord16 muw16B;
static UWord16 muw16C;

void main(void)
{
        /* input A value */
        muw32A = 0x12345678;

        /* input B value */
        muw16B = 0x5555;

        /* Divides muw32A by muw16B */
        muw16C = GFLIB_DivuLSS(muw32A, muw16B);
}
```

13.1.10 See Also

See **GFLIB_DivsLSS** for more information.

13.1.11 Performance

Table 13-2. Performance of the GFLIB_DivuLSS Function

Code Size (bytes)	94 bytes	
Data Size (bytes)	0 bytes	
Execution Clock	Min	13 cycles
	Max	144 cycles

Chapter 14
Calculating the Sine Value Using the Lookup Table

14.1 GFLIB_SinLut

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

14.1.1 Synopsis

```
#include ”gflib.h”
Frac16 GFLIB_SinLut(Frac16 f16Arg)
Frac16 GFLIB_SinLutFAsm(Frac16 f16Arg, Frac16 *pudtSinTable, UWord16 uw16TableSize)[1]
```

14.1.2 Arguments

Table 14-1. Function Arguments

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

14.1.3 Availability

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

This library module is targeted at the MCF51xx platform.

14.1.4 Dependencies

List of all dependent files:

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

1. optional function API allowing modification of values and size of the Look-up table

14.1.5 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 range from one to 15. The table contains $2^n + 1$ points.

The function finds the two nearest pre-calculated points of the input argument, and using the linear interpolation between these two points, it calculates the output value.

The sin function is asymmetrical along the defined interval, therefore the table contains pre-calculated values just for the range $-\pi/2$ to zero. For the values outside this interval, the function transforms the input value to the $-\pi/2$ to zero interval, and calculates as if it was in this interval. In the end, if the input was in the interval of zero to π, the output is negated.

Figure 14-1 shows the function that has nine table points, in other words $2^3 + 1$, therefore the table size is 3.

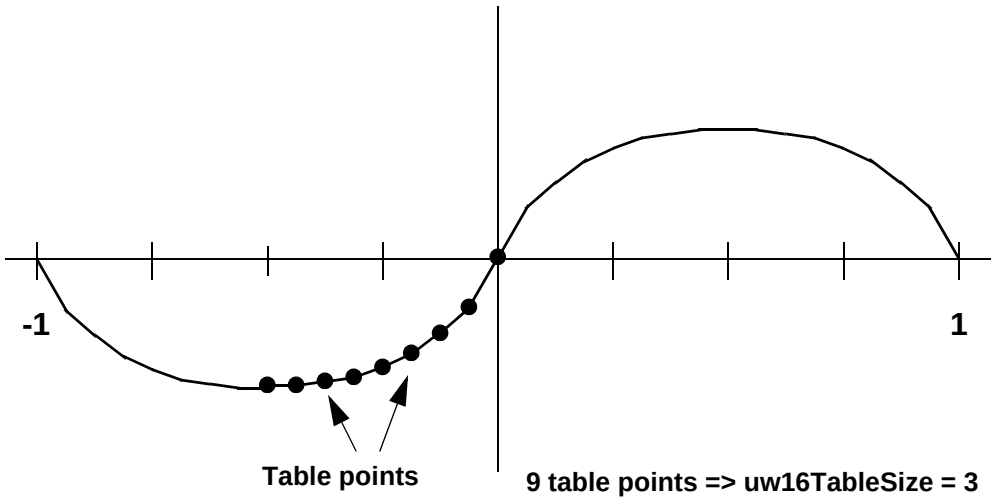

Figure 14-1. Algorithm Diagram

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

14.1.6 Returns

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

14.1.7 Range Issues

The input data value is in the range of <–1, 1), which corresponds to the angle in the range of <–π, π). The output data value is in the range of <–1, 1). It means that with the input value of zero, it will have the output result of zero. Similarly if the input value is 0.5, the output will be one.

14.1.8 Special Issues

None.

14.1.9 Implementation

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

Example 14-1.

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

#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_SinLut(mf16Input);
}
```

14.1.10 Performance

Table 14-2. Performance of the GFLIB_SinLut Function

Code Size (bytes)	90 bytes	
Data Size (bytes)	516 bytes	
Execution Clock	Min	44 cycles
	Max	45 cycles

Chapter 15
Calculating the Cosine Value Using the Lookup Table

15.1 GFLIB_CosLut

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

15.1.1 Synopsis

```
#include ”gflib.h”
Frac16 GFLIB_CosLut(Frac16 f16Arg)
Frac16 GFLIB_CosLutFAsm(Frac16 f16Arg, Frac16 *pudtSinTable, UWord16 uw16TableSize)[1]
```

15.1.2 Arguments

Table 15-1. Function Arguments

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

15.1.3 Availability

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

This library module is targeted at the MCF51xx platform.

15.1.4 Dependencies

List of all dependent files:

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

1. optional function API allowing to modify the values and size of the lookup table

15.1.5 Description

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

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

The cos function is symmetrical along the defined interval, therefore the table contains pre-calculated values just for the range from $-\pi$ to $-\pi/2$. This interval is used to share the sine function table in order 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 15-1 shows the function that has nine table points, in other words $2^3 + 1$, therefore the table size is 3.

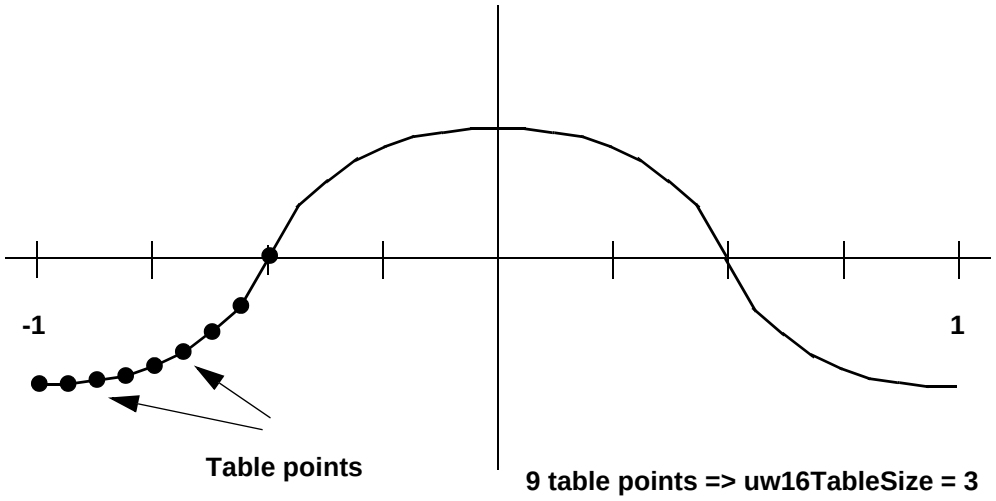

Figure 15-1. Algorithm Diagram

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

15.1.6 Returns

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

15.1.7 Range Issues

The input data value is in the range of <–1, 1), which corresponds to the angle in the range of <–π, π), and the output data value is in the range of <–1, 1). It means that with the input value of zero, it will have the output result of one. Similarly if the input value is ±1, the output will be –1.

15.1.8 Special Issues

None.

15.1.9 Implementation

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

Example 15-1.

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

#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_CosLut(mf16Input);
}
```

15.1.10 Performance

Table 15-2. Performance of the GFLIB_CosLut Function

Code Size (bytes)	96 bytes	
Data Size (bytes)	0 bytes	
Execution Clock	Min	47 cycles
	Max	50 cycles

Chapter 16
Calculating the Square Root Value of the Argument

16.1 GFLIB_SqrtPoly

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

16.1.1 Synopsis

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

16.1.2 Arguments

Table 16-1. Function Arguments

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

16.1.3 Availability

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

This library module is targeted at the MCF51xx platform.

16.1.4 Dependencies

List of all dependent files:

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

16.1.5 Description

The function **GFLIB_SqrtPoly** calculates and 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 fourth order with three intervals. Then the raw result is iterated in three steps to get a more precise result.

16.1.6 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.

16.1.7 Range Issues

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

16.1.8 Special Issues

None.

16.1.9 Implementation

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

Example 16-1.

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

#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);
}
```

16.1.10 Performance

Table 16-2. Performance of the GFLIB_SqrtPoly Function

Code Size (bytes)	172 bytes	
Data Size (bytes)	72 bytes	
Execution Clock	Min	84 cycles
	Max	101 cycles

Chapter 17
Calculating a 16-Bit Version of the Up/Down Ramp

17.1 GFLIB_Ramp16

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

17.1.1 Synopsis

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

17.1.2 Arguments

Table 17-1. Function Arguments

Name	In/Out	Format	Range	Description
f16Desired	In	SF16	0x8000... 0x7FFF	Desired value; the Frac16 data type is defined in the header file GFLIB_types.h.
f16Actual	In	SF16	0x8000... 0x7FFF	Actual value; the Frac16 data type is defined in the header file GFLIB_types.h.
*pudtParam	In	UI16	—	Pointer to a structure, containing the ramp-up and ramp-down increments.

Table 17-2. 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

17.1.3 Availability

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

This library module is targeted at the MCF51xx platform.

17.1.4 Dependencies

List of all dependent files:

- GFLIB_RampAsm.h
- GFLIB_types.h

17.1.5 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.

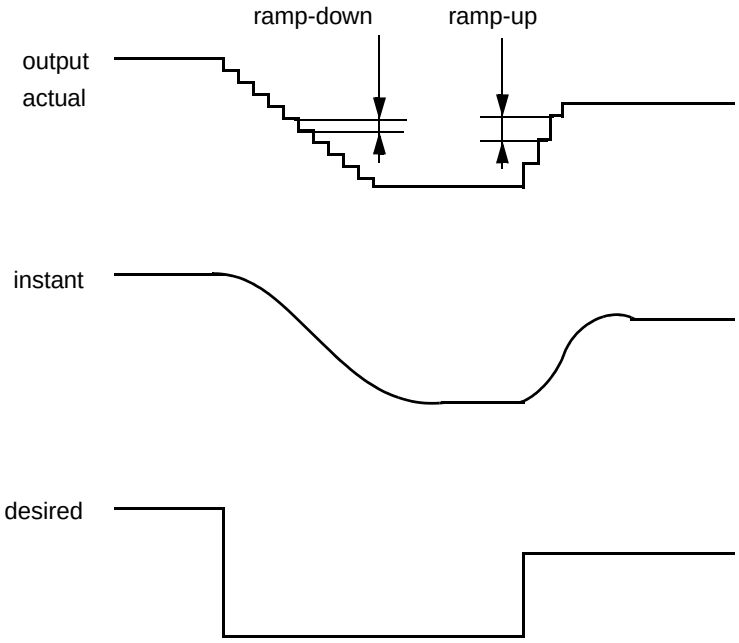

Figure 17-1. Algorithm Diagram

17.1.6 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.

17.1.7 Range Issues

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

17.1.8 Special Issues

None.

17.1.9 Implementation

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

Example 17-1.

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

#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);
}
```

17.1.10 Performance

Table 17-3. Performance of the GFLIB_Ramp16 Function

Code Size (bytes)	36 bytes	
Data Size (bytes)	0 bytes	
Execution Clock	Min	20 cycles
	Max	21 cycles

Chapter 18
Calculating a 32-Bit Version of the Up/Down Ramp

18.1 GFLIB_Ramp32

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

18.1.1 Synopsis

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

18.1.2 Arguments

Table 18-1. Function Arguments

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

Table 18-2. 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

18.1.3 Availability

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

This library module is targeted at the MCF51xx platform.

18.1.4 Dependencies

List of all dependent files:

- GFLIB_RampAsm.h
- GFLIB_types.h

18.1.5 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.

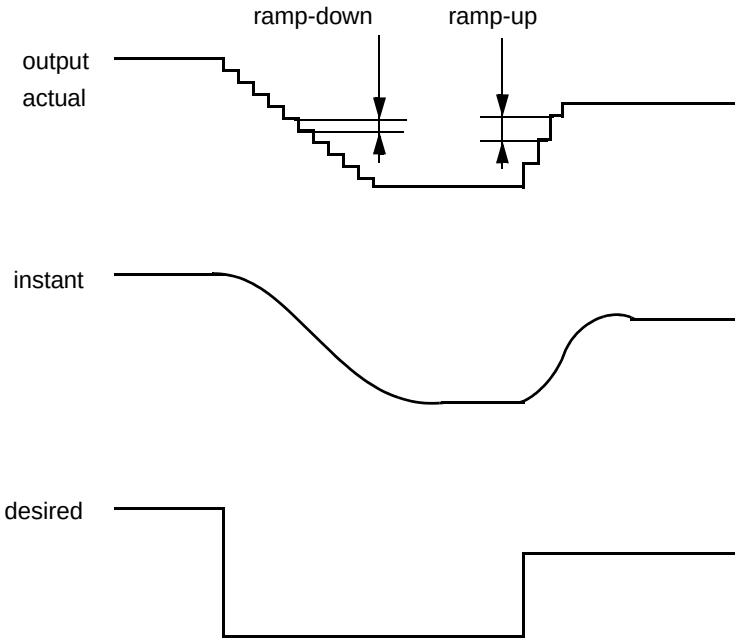

Figure 18-1. Algorithm Diagram

18.1.6 Returns

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

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

18.1.7 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.

18.1.8 Special Issues

None.

18.1.9 Implementation

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

Example 18-1.

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

#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);
}
```

18.1.10 Performance

Table 18-3. Performance of the GFLIB_Ramp32 Function

Code Size (bytes)	34 bytes	
Data Size (bytes)	0 bytes	
Execution Clock	Min	19 cycles
	Max	21 cycles

Chapter 19
Calculating a 16-Bit Scalar Up/Down Limitation of the Input Signal

19.1 GFLIB_Limit16

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

19.1.1 Synopsis

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

19.1.2 Arguments

Table 19-1. 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 a structure containing the upper and lower limits

Table 19-2. 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

19.1.3 Availability

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

This library module is targeted at the MCF51xx platform.

19.1.4 Dependencies

List of all dependent files:

- GFLIB_LimitAsm.h
- GFLIB_types.h

19.1.5 Description

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

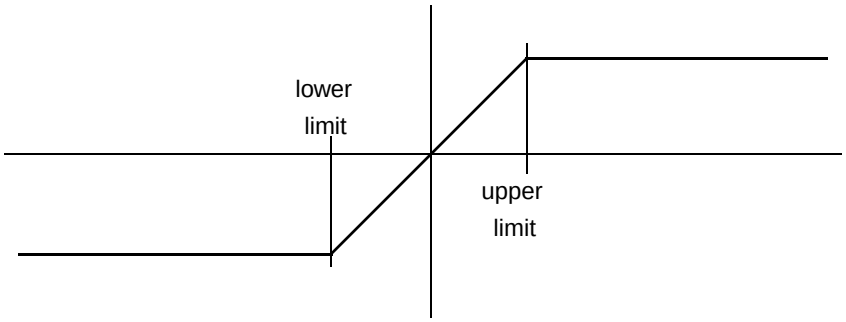

Figure 19-1. Algorithm Transition

19.1.6 Returns

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

19.1.7 Range Issues

The input data value is in the range of <–1, 1) and the output data values are in the range <upperLimit, lowerLimit>.

19.1.8 Special Issues

None.

19.1.9 Implementation

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

Example 19-1.

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

#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);
}
```

19.1.10 See Also

See **GFLIB_Limit32** for more information.

19.1.11 Performance

Table 19-3. Performance of the GFLIB_Limit16 Function

Code Size (bytes)	20 bytes	
Data Size (bytes)	0 bytes	
Execution Clock	Min	12 cycles
	Max	17 cycles

Chapter 20
Calculating a 32-Bit Scalar Up/Down Limitation of the Input Signal

20.1 GFLIB_Limit32

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

20.1.1 Synopsis

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

20.1.2 Arguments

Table 20-1. 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 a structure containing the upper and lower limits

Table 20-2. 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

20.1.3 Availability

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

This library module is targeted at the MCF51xx platform.

20.1.4 Dependencies

List of all dependent files:

- GFLIB_LimitAsm.h
- GFLIB_types.h

20.1.5 Description

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

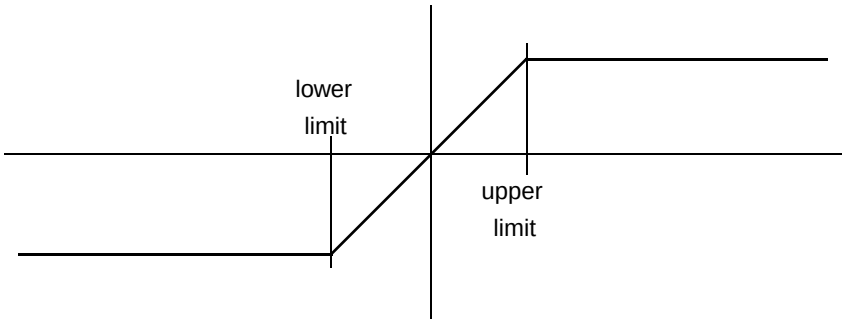

Figure 20-1. Algorithm Transition

20.1.6 Returns

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

20.1.7 Range Issues

The input data value is in the range of <–1, 1) and the output data values are in the range <upperLimit, lowerLimit>.

20.1.8 Special Issues

None.

20.1.9 Implementation

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

Example 20-1.

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

#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);
}
```

20.1.10 See Also

See **GFLIB_Limit16** for more information.

20.1.11 Performance

Table 20-3. Performance of the GFLIB_Limit32 Function

Code Size (bytes)	22 bytes	
Data Size (bytes)	0 bytes	
Execution Clock	Min	12 cycles
	Max	17 cycles

Chapter 21
Calculating a Parallel Form of the Proportional-Integral Regulator

21.1 GFLIB_ControllerPIp

This function calculates the parallel form of the proportional-integral (PI) regulator.

21.1.1 Synopsis

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

21.1.2 Arguments

Table 21-1. 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	—	—	Pointer to a structure of the PI controller parameters; the GFLIB_CONTROLLER_PI_P_PARAMS_T data type is defined in 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 zero, 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 21-2. 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 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 one, the controller output reaches either f16UpperLimit or f16LowerLimit.

21.1.3 Availability

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

This library module is targeted at the MCF51xx platform.

21.1.4 Dependencies

List of all dependent files:

- GFLIB_ControllerPIpAsm.h
- GFLIB_types.h

21.1.5 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 the 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 satFlag, set by the user software outside the PI controller function and passed into the function and the pointer pSatFlag is not zero, then the integral portion is frozen.

The PI algorithm in the continuous time domain is as follows:

$$u(t) = K\left[e(t) + \frac{1}{T_I}\int_0^{\tau} e(\tau)d(\tau)\right] \qquad \textbf{Eqn. 21-1}$$

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 21-1 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:

$$u_I(k) = u_I(k-1) + T \times e(k) \qquad \textbf{Eqn. 21-2}$$

The discrete time domain representation of the PI algorithms is as follows:

$$u(k) = K \times e(k) + u_I(k-1) + K_I \times e(k) \qquad \textbf{Eqn. 21-3}$$

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]

$$K_I = K \times \frac{T}{T_I} \qquad \textbf{Eqn. 21-4}$$

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

$$u_f(k) = K_{sc} \times e_f(k) + u_{If}(k-1) + K_{Isc} \times e_f(k) \qquad \textbf{Eqn. 21-5}$$

where:

$$u_f(k) = u(k)/u_{max} \quad \textit{Eqn. 21-6}$$

$$e_f(k) = e(k)/e_{max} \quad \textit{Eqn. 21-7}$$

$$K_{sc} = K \times \frac{e_{max}}{u_{max}} \quad \textit{Eqn. 21-8}$$

$$K_{Isc} = K \times \frac{T}{T_I} \times \frac{e_{max}}{u_{max}} = K_I \times \frac{e_{max}}{u_{max}} \quad \textit{Eqn. 21-9}$$

where

e_{max} — input maximum range

u_{max} — output maximum range

Each parameter (for example K_{Isc}) of the PI algorithm is represented by two parameters in the processor implementation (for example f16IntegGain and i16IntegGainShift).

$$f16PropGain = K_{sc} \times 2^{-i16PropGainShift} \quad \textit{Eqn. 21-10}$$

$$f16IntegGain = K_{Isc} \times 2^{-i16IntegGainShift} \quad \textit{Eqn. 21-11}$$

where:

$$0 \le f16PropGain < 1 \quad \textit{Eqn. 21-12}$$

$$0 \le f16IntegGain < 1 \quad \textit{Eqn. 21-13}$$

$$0 \le i16PropGainShift < 14 \quad \textit{Eqn. 21-14}$$

$$0 \le i16IntegGainShift < 14 \quad \textit{Eqn. 21-15}$$

Example 21-1.

Assumption: K_{Isc}= 2.4

In this case the K_{Isc} cannot be directly interpreted as a fractional value, since 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

Then calculate the corresponding i16IntegGainShift parameter.

$$\frac{\log(K_{Isc}) - \log(0.5)}{\log 2} \geq i16IntegGainShift \quad \textit{Eqn. 21-16}$$

$$\frac{\log(2.4) - \log(0.5)}{\log 2} \geq i16IntegGainShift \quad \textit{Eqn. 21-17}$$

$$2.26 \geq i16IntegGainShift \quad \textit{Eqn. 21-18}$$

$$\frac{\log(K_{Isc}) - \log(1)}{\log 2} < i16IntegGainShift \quad \textit{Eqn. 21-19}$$

$$\frac{\log(K_{Isc})}{\log 2} < i16IntegGainShift \quad \textit{Eqn. 21-20}$$

$$\frac{\log(2.4)}{\log 2} < i16IntegGainShift \quad \textit{Eqn. 21-21}$$

$$1.26 < i16IntegGainShift \quad \textit{Eqn. 21-22}$$

The parameter i16IntegGainShift is in the following range:

$$1.26 < i16IntegGainShift \leq 2.26 \quad \textit{Eqn. 21-23}$$

Since this parameter is an integer value, the result is:

$$i16IntegGainShift = 2 \quad \textit{Eqn. 21-24}$$

Then:

$$f16IntegGain = K_{Isc} \times 2^{-i16IntegGainShift} \quad \textit{Eqn. 21-25}$$

$$f16IntegGain = 2.4 \times 2^{(-2)} = 0.6 \quad \textit{Eqn. 21-26}$$

Result:
f16IntegGain = 0.6
i16IntegGainShift = 2

21.1.6 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:

$$f16LowerLimit \leq PIresult \leq f16UpperLimit \quad \textit{Eqn. 21-27}$$

21.1.7 Range Issues

The PI controller parameters are in the following range:

$$0 \leq f16PropGain < 1 \qquad \textit{Eqn. 21-28}$$

$$0 \leq f16IntegGain < 1 \qquad \textit{Eqn. 21-29}$$

$$0 \leq i16PropGainShift < 14 \qquad \textit{Eqn. 21-30}$$

$$0 \leq i16IntegGainShift < 14 \qquad \textit{Eqn. 21-31}$$

21.1.8 Special Issues

None.

21.1.9 Implementation

Example 21-2.

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

#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_ControllerPIpAsm(mf16ErrorK, &mudtControllerParam,
&mi16SatFlag);
}
```

21.1.10 Performance

Table 21-3. Performance of the GFLIB_ControllerPIp Function

Code Size (bytes)	196 bytes	
Data Size (bytes)	0 bytes	
Execution Clock	Min	48 cycles
	Max	77 cycles