NXP Semiconductors
User Manual

Document identifier: CM7FMLIBUG
Rev. 5, 01 November 2021

MLIB User's Guide

ARM® Cortex® M7F

Contents

Chapter 1
Library

1.1 Introduction

1.1.1 Overview

This user's guide describes the Math Library (MLIB) for the family of ARM Cortex M7F core-based microcontrollers. This library contains optimized functions.

1.1.2 Data types

MLIB supports several data types: (un)signed integer, fractional , and accumulator, and floating point. The integer data types are useful for general-purpose computation; they are familiar to the MPU and MCU programmers. The fractional data types enable powerful numeric and digital-signal-processing algorithms to be implemented. The accumulator data type is a combination of both; that means it has the integer and fractional portions. The floating-point data types are capable of storing real numbers in wide dynamic ranges. The type is represented by binary digits and an exponent. The exponent allows scaling the numbers from extremely small to extremely big numbers. Because the exponent takes part of the type, the overall resolution of the number is reduced when compared to the fixed-point type of the same size.

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

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

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

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

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

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

The following list shows the floating-point types defined in the libraries:

- Floating point 32-bit single precision—<-3.40282 · 10^{38} ; 3.40282 · 10^{38}> with the minimum resolution of 2^{-23}

1.1.3 API definition

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

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

where the function is compiled from four parts:

- MLIB—this is the library prefix
- Mac—the function name—Multiply-Accumulate
- F32—the function output type

- Iss—the types of the function inputs; if all the inputs have the same type as the output, the inputs are not marked

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

Table 1. Input/output types

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

1.1.4 Supported compilers

MLIB for the ARM Cortex M7F core is written in C language or assembly language with C-callable interface depending on the specific function. The library is built and tested using the following compilers:

- MCUXpresso IDE
- IAR Embedded Workbench
- Keil µVision

For the MCUXpresso IDE, the library is delivered in the *mlib.a* file.

For the Kinetis Design Studio, the library is delivered in the *mlib.a* file.

For the IAR Embedded Workbench, the library is delivered in the *mlib.a* file.

For the Keil µVision, the library is delivered in the *mlib.lib* file.

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

1.1.5 Library configuration

MLIB for the ARM Cortex M7F core is written in C language or assembly language with C-callable interface depending on the specific function. Some functions from this library are inline type, which are compiled together with project using this library. The optimization level for inline function is usually defined by the specific compiler setting. It can cause an issue especially when high optimization level is set. Therefore the optimization level for all inline assembly written functions is defined by compiler pragmas using macros. The configuration header file *RTCESL_cfg.h* is located in: *specific library folder\MLIB\Include*. The optimization level can be changed by modifying the macro value for specific compiler. In case of any change the library functionality is not guaranteed.

Similarly as optimization level the High-speed functions execution suppport can be enable by defined symbol RAM_RELOCATION in project setting described in the High-speed functions execution suppport cheaper for specific compiler.

1.1.6 Special issues

1. The equations describing the algorithms are symbolic. If there is positive 1, the number is the closest number to 1 that the resolution of the used fractional type allows. If there are maximum or minimum values mentioned, check the range allowed by the type of the particular function version.
2. The library functions that round the result (the API contains Rnd) round to nearest (half up).
3. This RTCESL requires the DSP extension for some saturation functions. If the core does not support the DSP extension feature the assembler code of the RTCESL will not be buildable. For example the core1 of the LPC55s69 has no DSP extension.

1.2 Library integration into project (MCUXpresso IDE)

This section provides a step-by-step guide on how to quickly and easily include MLIB into any MCUXpresso SDK example or new SDK project using MCUXpresso IDE. The SDK based project uses RTCESL from SDK package.

High-speed functions execution suppport

Some RT (or other) platforms contain high-speed functions execution support by relocating all functions from the default Flash memory location to the RAM location for much faster code access. The feature is important especially for devices with a slow Flash interface. This section shows how to turn the RAM optimization feature support on and off.

1. In the MCUXpresso SDK project name node or on the left-hand side, click Properties or select Project > Properties from the menu. A project properties dialog appears.
2. Expand the C/C++ Build node and select Settings. See Figure 1.
3. On the right-hand side, under the MCU C Compiler node, click the Preprocessor node. See Figure 1 .

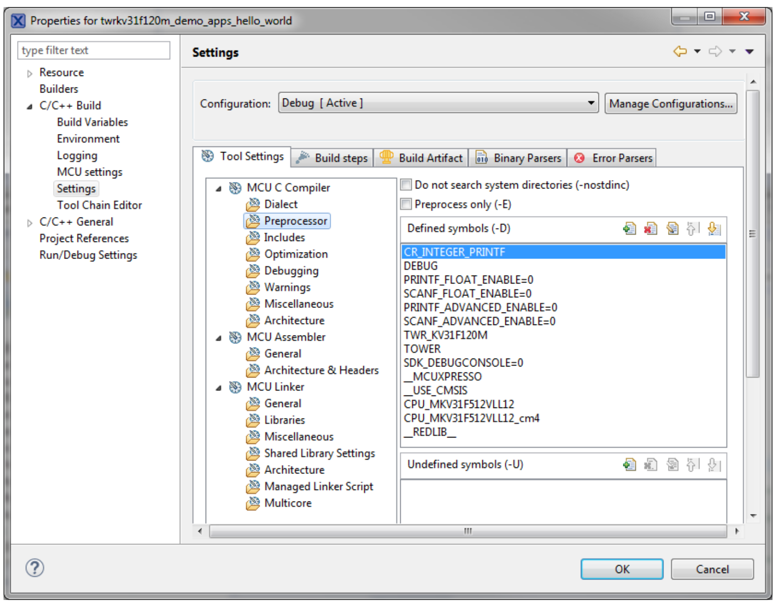

Figure 1. Defined symbols

4. On the right-hand side of the dialog, click the Add... icon located next to the Defined symbols (-D) title.
5. In the dialog that appears (see Figure 2), type the following:
   - RAM_RELOCATION — to turn the RAM optimization feature support on

   If the define is defined, all RTCEL functions are put to the RAM.

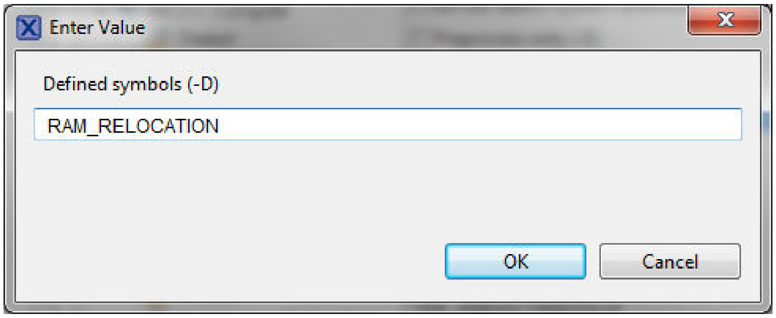

Figure 2. Symbol definition

6. Click OK in the dialog.
7. Click OK in the main dialog.

The RAM_RELOCATION macro places the `__RAMFUNC(RAM)` atribute in front of each function declaration.

Adding RTCESL component to project

The MCUXpresso SDK package is necessary to add any example or new project and RTCESL component. In case the package has not been downloaded go to mcuxpresso.nxp.com, build the final MCUXpresso SDK package for required board and download it.

After package is dowloaded, open the MCUXpresso IDE and drag&drop the SDK package in zip format to the Installed SDK window of the MCUXpresso IDE. After SDK package is dropped the mesage accepting window appears as can be show in following figure.

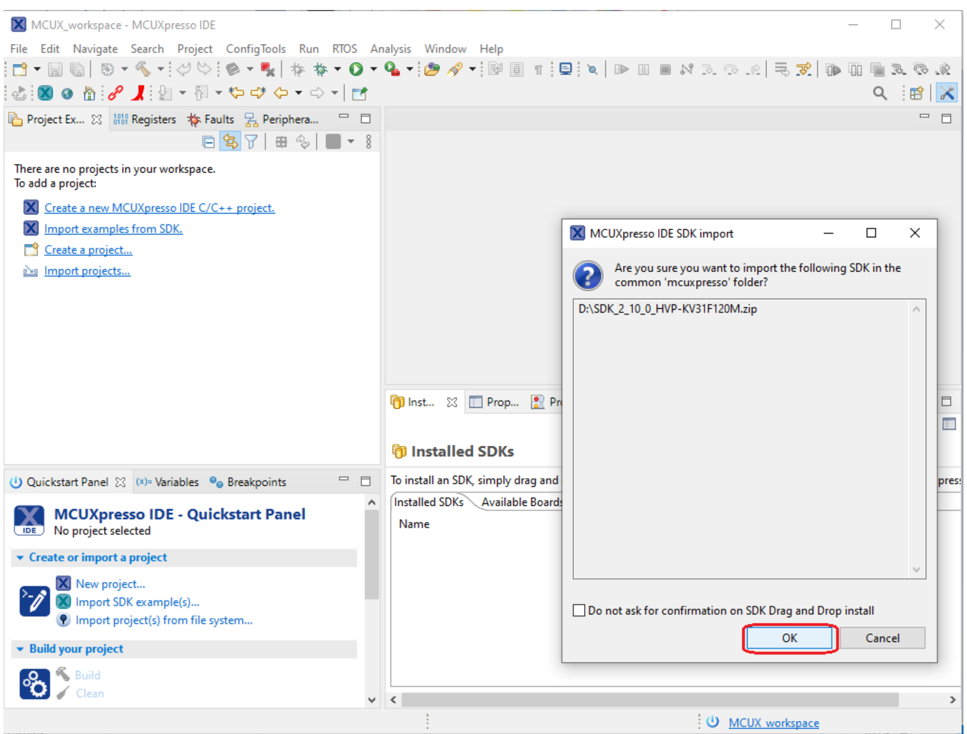

Figure 3. MCUXpresso IDE - imporing the SDK package to MCUXpresso IDE

Click OK to confirm the SDK package import. Find the Quickstart panel in left bottom part of the MCUXpresso IDE and click New project... item or Import SDK example(s)... to add rtcesl component to the project.

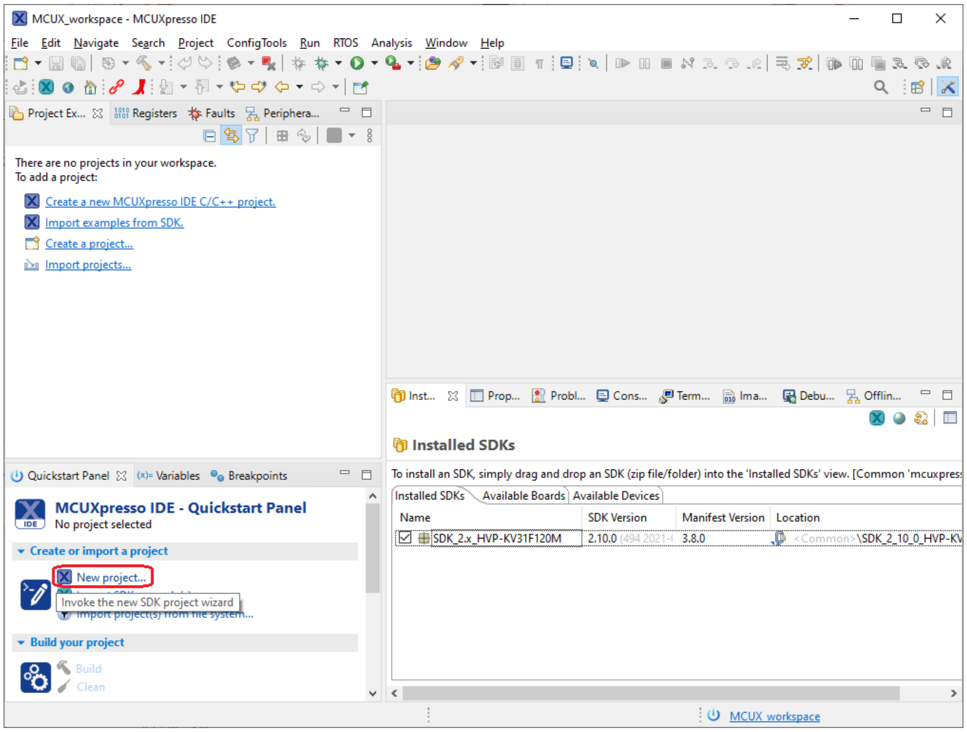

Figure 4. MCUXpresso IDE - create new project or Import SDK example(s)

Then select your board, and clik Next button.

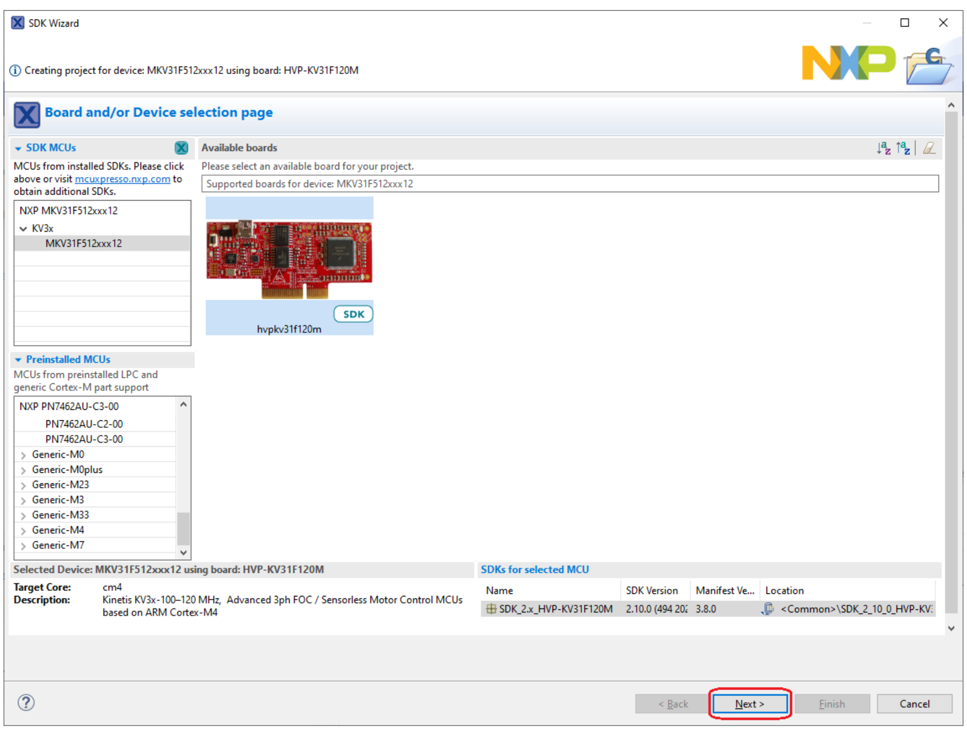

Figure 5. MCUXpresso IDE - selecting the board

Find the Middleware tab in the Components part of the window and click on the checkbox to be the rtcesl component ticked. Last step is to click the Finish button and wait for project creating with all RTCESL libraries and include paths.

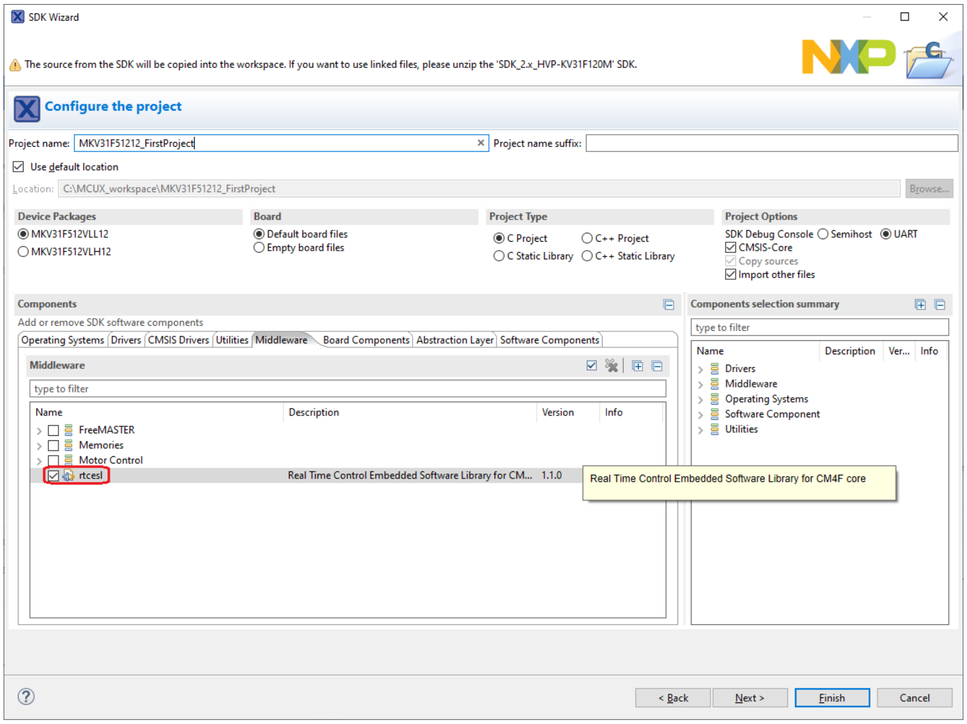

Figure 6. MCUXpresso IDE - selecting rtcesl component

Type the #include syntax into the code where you want to call the library functions. In the left-hand dialog, open the required .c file. After the file opens, include the following line into the #include section:

```
#include "mlib_FP.h"
```

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

1.3 Library integration into project (Keil µVision)

This section provides a step-by-step guide on how to quickly and easily include MLIB into an empty project or any MCUXpresso SDK example or demo application projects using Keil µVision. This example uses the default installation path (C:\NXP\RTCESL\CM7F_RTCESL_4.7_KEIL). If you have a different installation path, use that path instead. If any MCUXpresso SDK project is intended to use (for example hello_world project) go to Linking the files into the project chapter otherwise read next chapter.

NXP pack installation for new project (without MCUXpresso SDK)

This example uses the NXP MKV58F1M0xxx22 part, and the default installation path (C:\NXP\RTCESL\CM7F_RTCESL_4.7_KEIL) is supposed. If the compiler has never been used to create any NXP MCU-based projects before, check whether the NXP MCU pack for the particular device is installed. Follow these steps:

1. Launch Keil µVision.
2. In the main menu, go to Project > Manage > Pack Installer….
3. In the left-hand dialog (under the Devices tab), expand the All Devices > Freescale (NXP) node.
4. Look for a line called "KVxx Series" and click it.
5. In the right-hand dialog (under the Packs tab), expand the Device Specific node.
6. Look for a node called "Keil::Kinetis_KVxx_DFP." If there are the Install or Update options, click the button to install/update the package. See Figure 7.

7. When installed, the button has the "Up to date" title. Now close the Pack Installer.

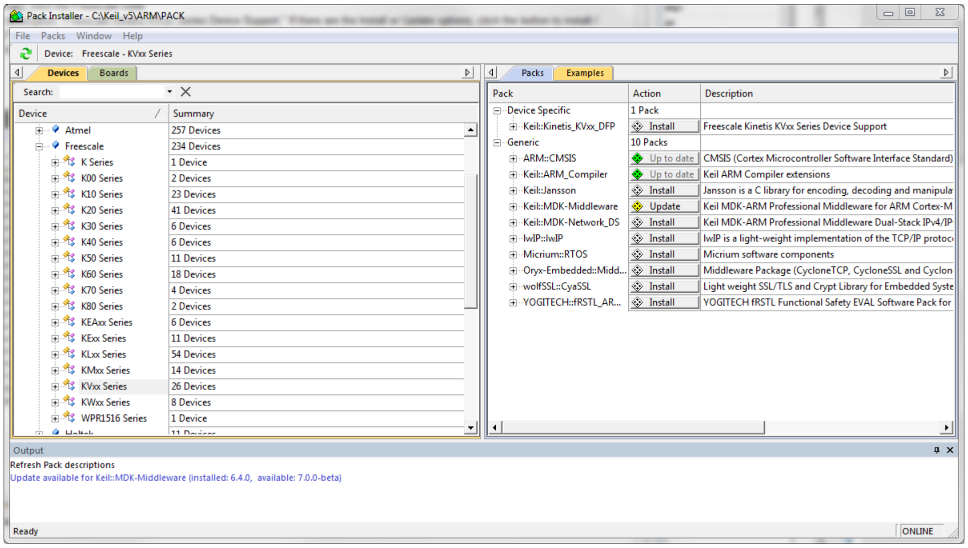

Figure 7. Pack Installer

New project (without MCUXpresso SDK)

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

1. Launch Keil µVision.
2. In the main menu, select Project > New µVision Project…, and the Create New Project dialog appears.
3. Navigate to the folder where you want to create the project, for example C:\KeilProjects\MyProject01. Type the name of the project, for example MyProject01. Click Save. See Figure 8.

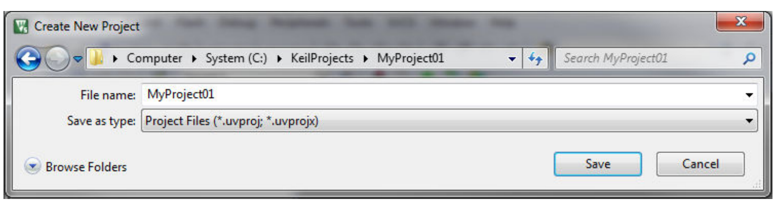

Figure 8. Create New Project dialog

4. In the next dialog, select the Software Packs in the very first box.
5. Type '' into the Search box, so that the device list is reduced to the devices.
6. Expand the node.
7. Click the MKV58F1M0xxx22 node, and then click OK. See Figure 9.

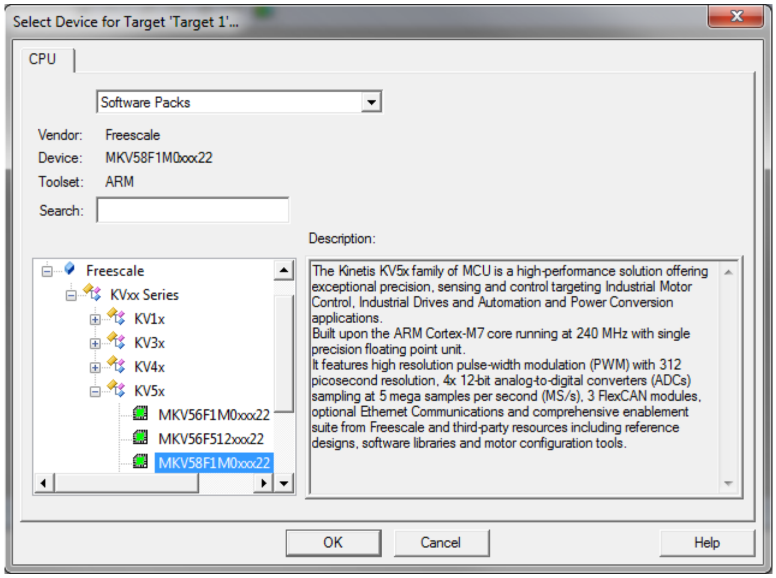

Figure 9. Select Device dialog

8. In the next dialog, expand the Device node, and tick the box next to the Startup node. See Figure 10.
9. Expand the CMSIS node, and tick the box next to the CORE node.

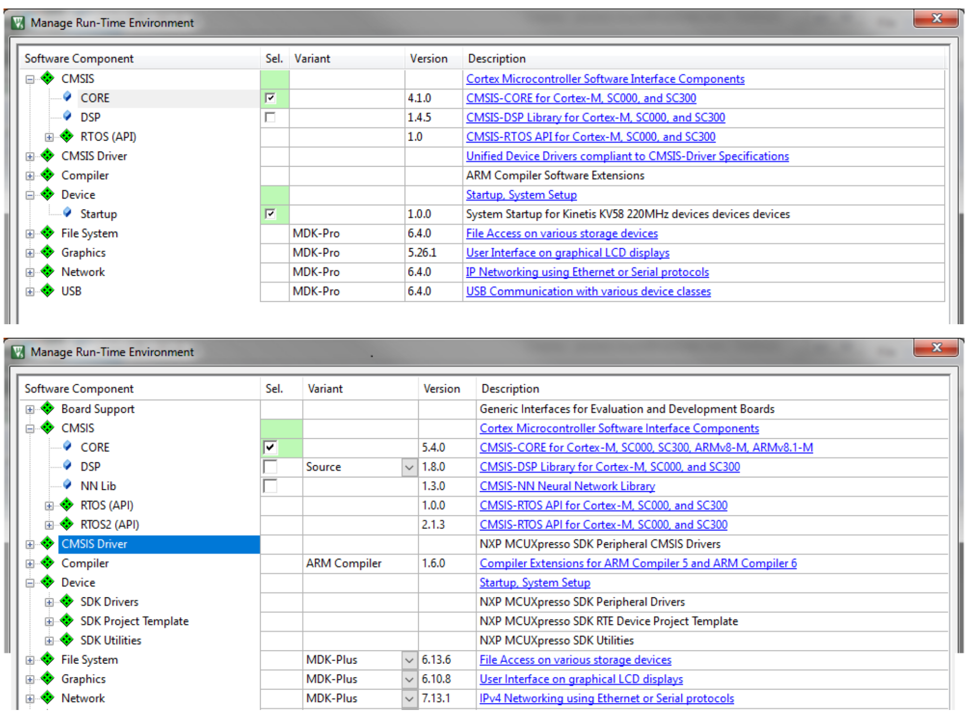

Figure 10. Manage Run-Time Environment dialog

10. Click OK, and a new project is created. The new project is now visible in the left-hand part of Keil µVision. See Figure 11.

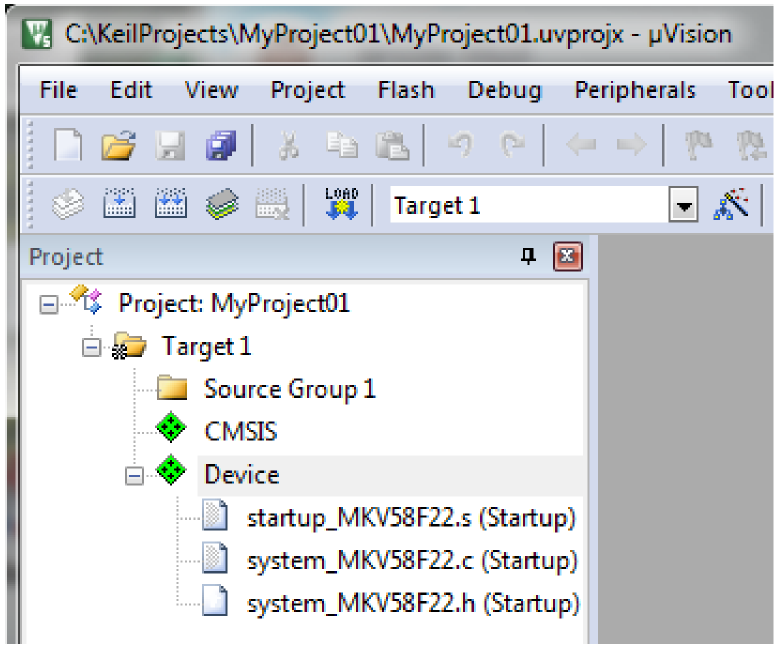

Figure 11. Project

11. In the main menu, go to Project > Options for Target 'Target1'…, and a dialog appears.
12. Select the Target tab.
13. Select Use Single Precision in the Floating Point Hardware option. See Figure 11.

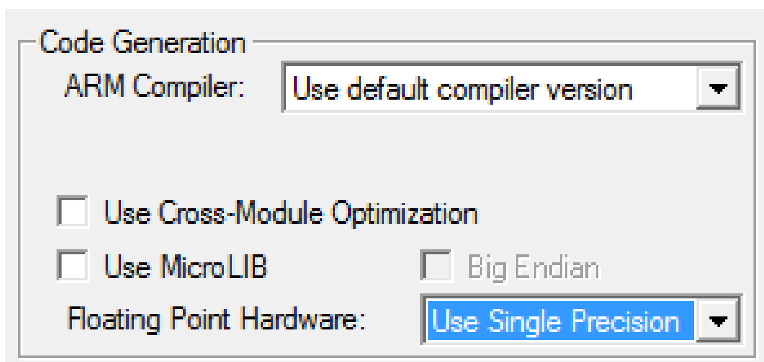

Figure 12. FPU

High-speed functions execution support

Some RT (or other) platforms contain high-speed functions execution support by relocating all functions from the default Flash memory location to the RAM location for much faster code access. The feature is important especially for devices with a slow Flash interface. This section shows how to turn the RAM optimization feature support on and off.

1. In the main menu, go to Project > Options for Target 'Target1'…, and a dialog appears.
2. Select the C/C++ tab. See #unique_19.
3. In the Include Preprocessor Symbols text box, type the following:
   - RAM_RELOCATION — to turn the RAM optimization feature support on

   If the define is defined, all RTCEL functions are put to the RAM.

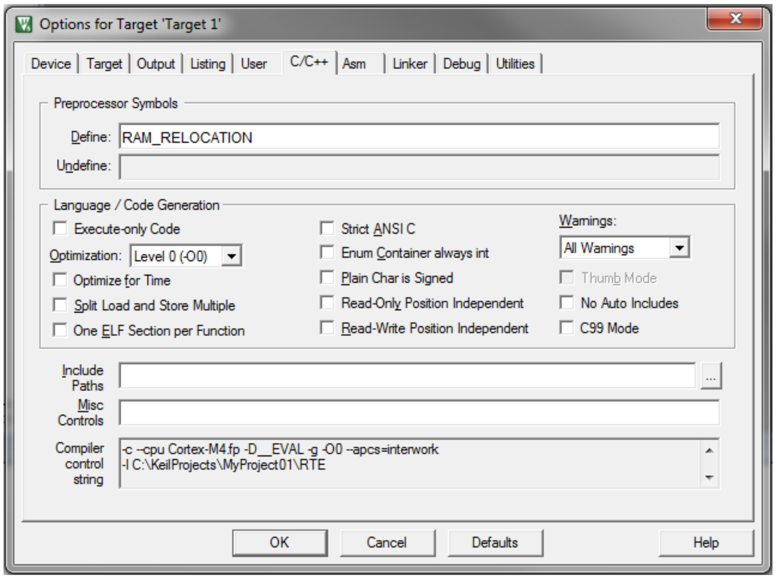

Figure 13. Preprocessor symbols

4. Click OK in the main dialog.

The RAM_RELOCATION macro places the `__attribute__ ((section ("ram")))` atribute in front of each function declaration.

Linking the files into the project

To include the library files in the project, create groups and add them.

1. Right-click the Target 1 node in the left-hand part of the Project tree, and select Add Group… from the menu. A new group with the name New Group is added.
2. Click the newly created group, and press F2 to rename it to RTCESL.
3. Right-click the RTCESL node, and select Add Existing Files to Group 'RTCESL'… from the menu.
4. Navigate into the library installation folder C:\NXP\RTCESL\CM7F_RTCESL_4.7_KEIL\MLIB\Include, and select the *mlib_FP.h* file. If the file does not appear, set the Files of type filter to Text file. Click Add. See Figure 14.

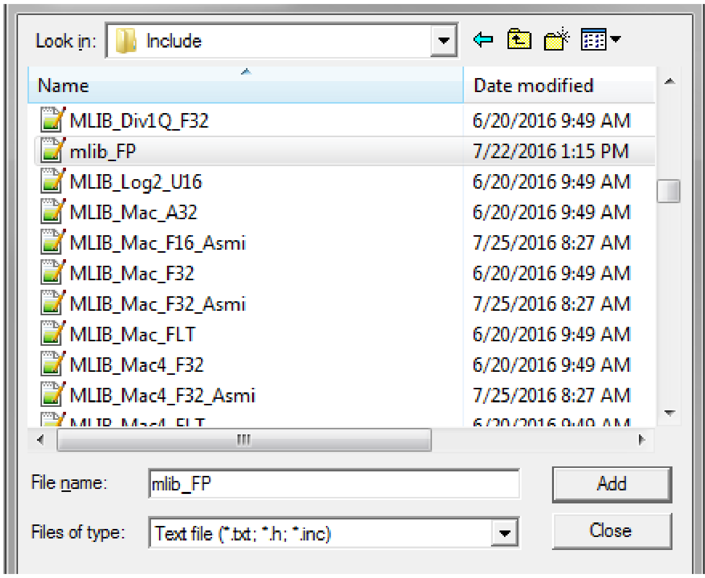

Figure 14. Adding .h files dialog

5. Navigate to the parent folder C:\NXP\RTCESL\CM7F_RTCESL_4.7_KEIL\MLIB, and select the *mlib.lib* file. If the file does not appear, set the Files of type filter to Library file. Click Add. See Figure 15.

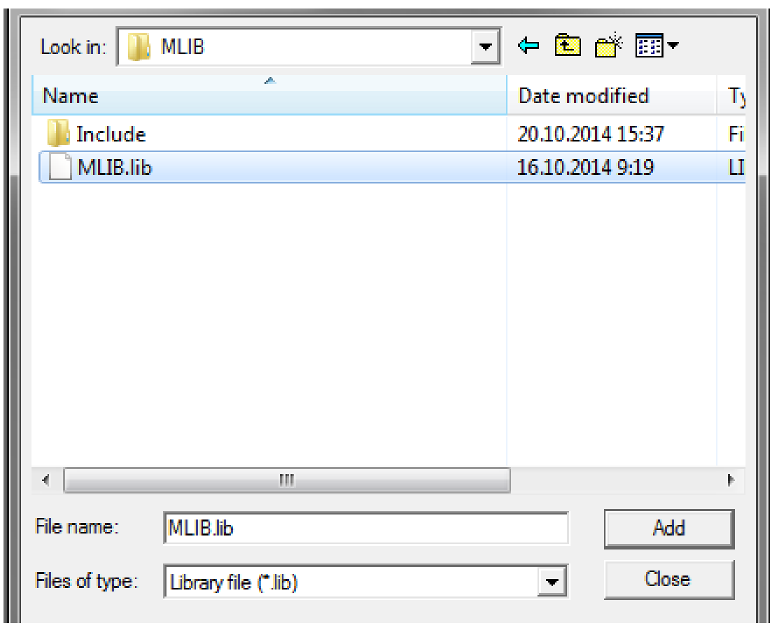

Figure 15. Adding .lib files dialog

6. Now, all necessary files are in the project tree; see Figure 16. Click Close.

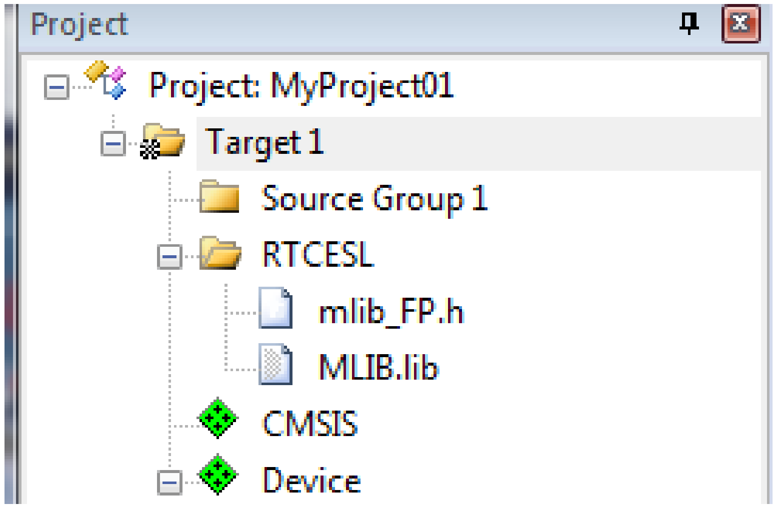

Figure 16. Project workspace

Library path setup

The following steps show the inclusion of all dependent modules.

1. In the main menu, go to Project > Options for Target 'Target1'…, and a dialog appears.
2. Select the C/C++ tab. See Figure 17.
3. In the Include Paths text box, type the following path (if there are more paths, they must be separated by ';') or add it by clicking the … button next to the text box:
   - "C:\NXP\RTCESL\CM7F_RTCESL_4.7_KEIL\MLIB\Include"
4. Click OK.
5. Click OK in the main dialog.

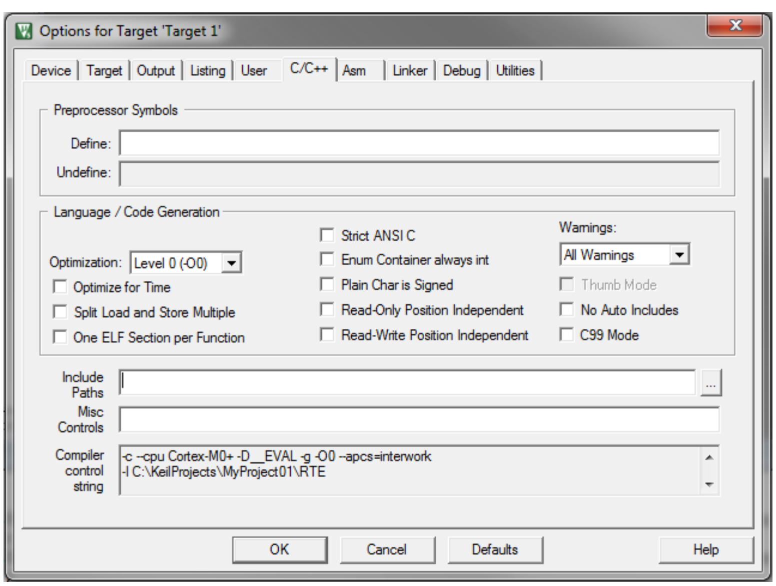

Figure 17. Library path addition

Type the #include syntax into the code. Include the library into a source file. In the new project, it is necessary to create a source file:

1. Right-click the Source Group 1 node, and Add New Item to Group 'Source Group 1'… from the menu.
2. Select the C File (*.c*) option, and type a name of the file into the Name box, for example '*main.c*'. See Figure 18.

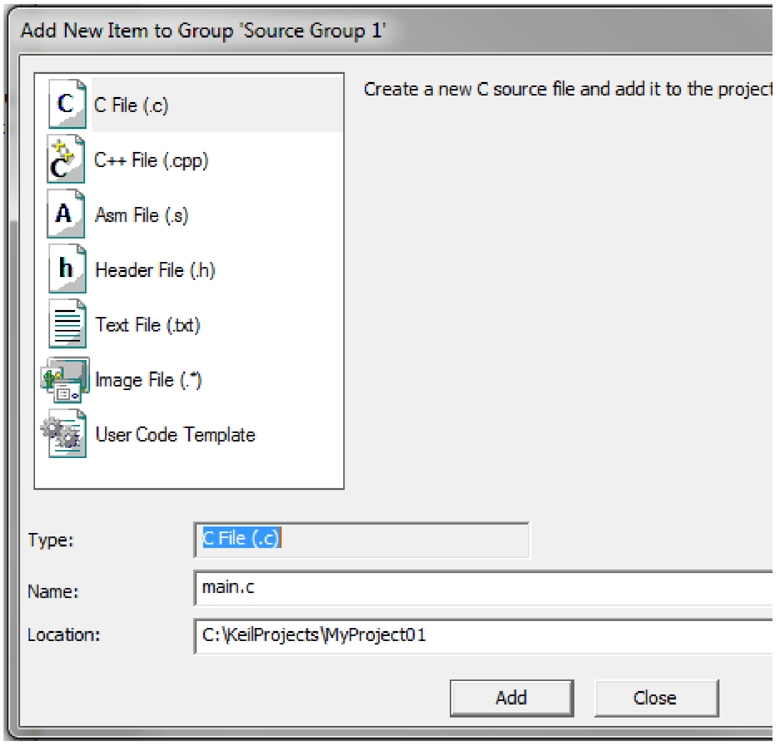

Figure 18. Adding new source file dialog

3. Click Add, and a new source file is created and opened up.
4. In the opened source file, include the following line into the #include section, and create a main function:

```
#include "mlib_FP.h"

int main(void)
{
  while(1);
}
```

When you click the Build (F7) icon, the project will be compiled without errors.

1.4 Library integration into project (IAR Embedded Workbench)

This section provides a step-by-step guide on how to quickly and easily include the MLIB into an empty project or any MCUXpresso SDK example or demo application projects using IAR Embedded Workbench. This example uses the default installation path (C:\NXP\RTCESL\CM7F_RTCESL_4.7_IAR). If you have a different installation path, use that path instead. If any MCUXpresso SDK project is intended to use (for example hello_world project) go to Linking the files into the project chapter otherwise read next chapter.

New project (without MCUXpresso SDK)

This example uses the NXP MKV58F1M0xxx22 part, and the default installation path (C:\NXP\RTCESL\CM7F_RTCESL_4.7_IAR) is supposed. To start working on an application, create a new project. If the project already exists and is opened, skip to the next section. Perform these steps to create a new project:

1. Launch IAR Embedded Workbench.

2. In the main menu, select Project > Create New Project… so that the "Create New Project" dialog appears. See Figure 19.

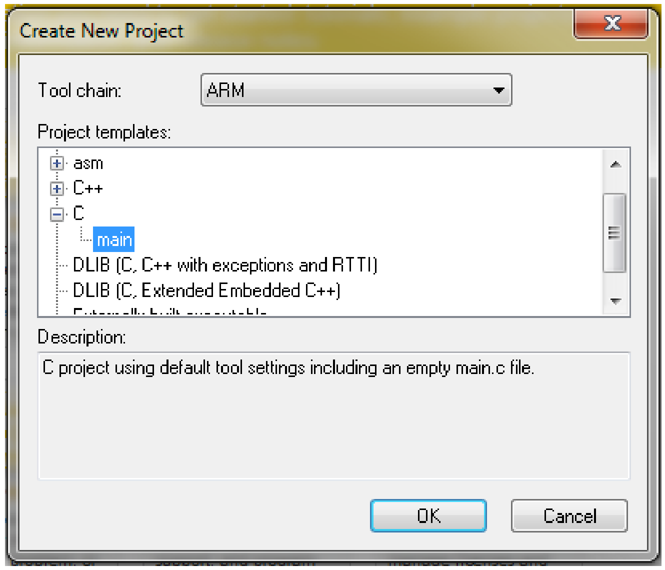

Figure 19. Create New Project dialog

3. Expand the C node in the tree, and select the "main" node. Click OK.
4. Navigate to the folder where you want to create the project, for example, C:\IARProjects\MyProject01. Type the name of the project, for example, MyProject01. Click Save, and a new project is created. The new project is now visible in the left-hand part of IAR Embedded Workbench. See Figure 20.

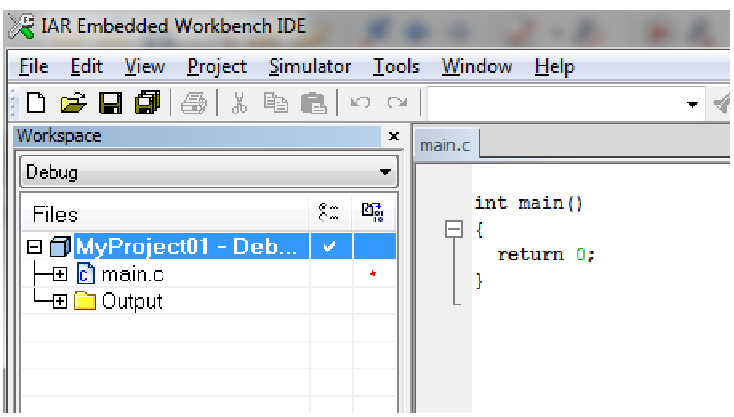

Figure 20. New project

5. In the main menu, go to Project > Options…, and a dialog appears.
6. In the Target tab, select the Device option, and click the button next to the dialog to select the MCU. In this example, select NXP > KV5x > NXP MKV58F1M0xxx22. Select VFPv5 single precision in the FPU option. Click OK. See Figure 21.

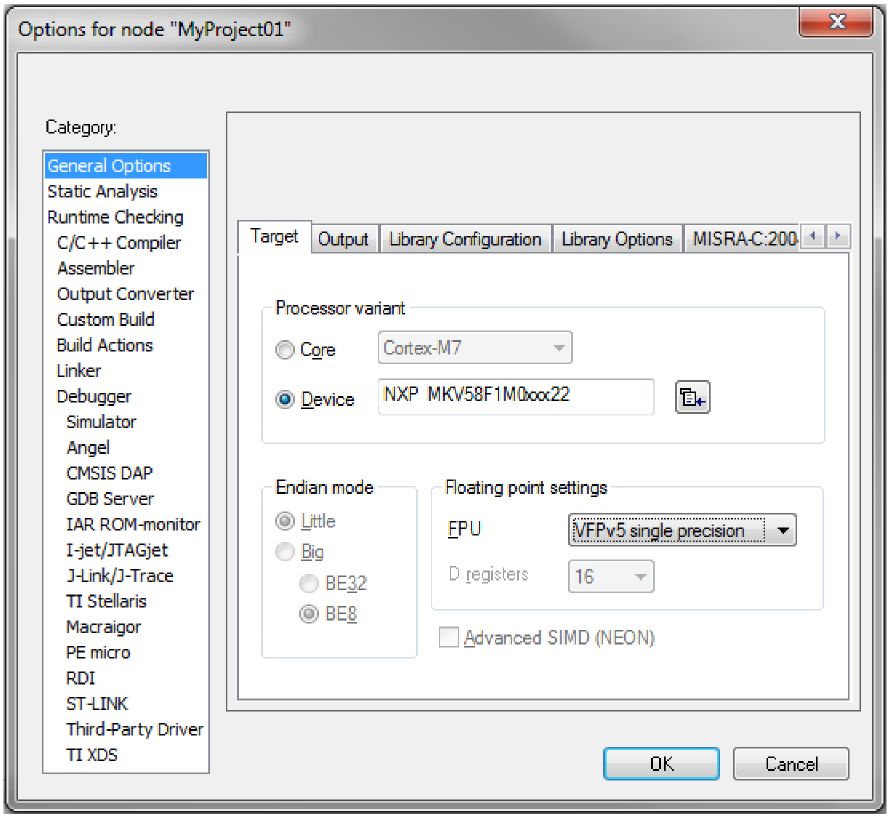

Figure 21. Options dialog

High-speed functions execution suppport

Some RT (or other) platforms contain high-speed functions execution support by relocating all functions from the default Flash memory location to the RAM location for much faster code access. The feature is important especially for devices with a slow Flash interface. This section shows how to turn the RAM optimization feature support on and off.

1. In the main menu, go to Project > Options…, and a dialog appears.
2. In the left-hand side column, select C/C++ Compiler.
3. In the right-hand side of the dialog, click the Preprocessor tab (it can be hidden on the right; use the arrow icons for navigation).
4. In the text box (in Defined symbols: (one per line)), type the following (See Figure 22):
   - RAM_RELOCATION — to turn the RAM optimization feature support on

   If the define is defined, all RTCEL functions are put to the RAM.

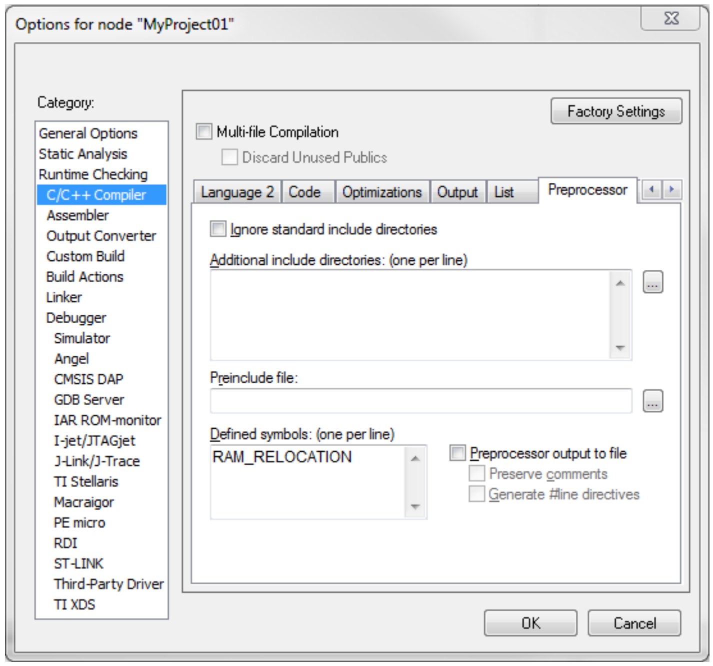

Figure 22. Defined symbols

5. Click OK in the main dialog.

The RAM_RELOCATION macro places the `__ramfunc` atribute in front of each function declaration.

Library path variable

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

1. In the main menu, go to Tools > Configure Custom Argument Variables…, and a dialog appears.
2. Click the New Group button, and another dialog appears. In this dialog, type the name of the group PATH, and click OK. See Figure 23.

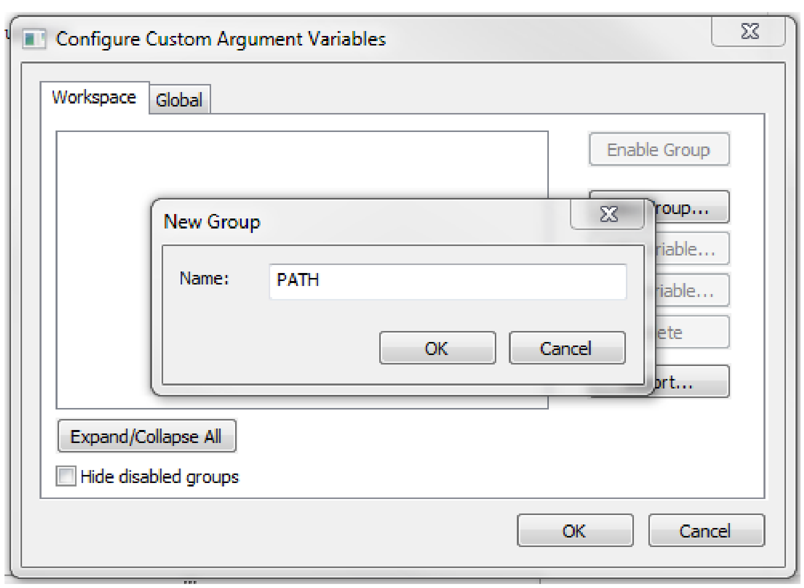

Figure 23. New Group

3. Click on the newly created group, and click the Add Variable button. A dialog appears.
4. Type this name: RTCESL_LOC
5. To set up the value, look for the library by clicking the '…' button, or just type the installation path into the box: C:\NXP\RTCESL\CM7F_RTCESL_4.7_IAR. Click OK.
6. In the main dialog, click OK. See Figure 24.

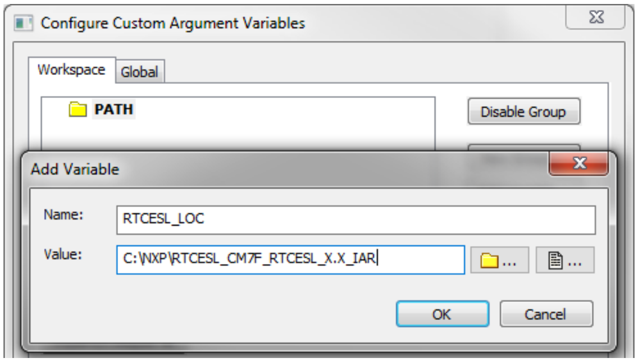

Figure 24. New variable

Linking the files into the project

To include the library files into the project, create groups and add them.

1. Go to the main menu Project > Add Group…
2. Type RTCESL, and click OK.
3. Click on the newly created node RTCESL, go to Project > Add Group…, and create a MLIB subgroup.
4. Click on the newly created node MLIB, and go to the main menu Project > Add Files… See Figure 26.
5. Navigate into the library installation folder C:\NXP\RTCESL\CM7F_RTCESL_4.7_IAR\MLIB\Include, and select the *mlib_FP.h* file. (If the file does not appear, set the file-type filter to Source Files.) Click Open. See Figure 25.
6. Navigate into the library installation folder C:\NXP\RTCESL\CM7F_RTCESL_4.7_IAR\MLIB, and select the *mlib.a* file. If the file does not appear, set the file-type filter to Library / Object files. Click Open.

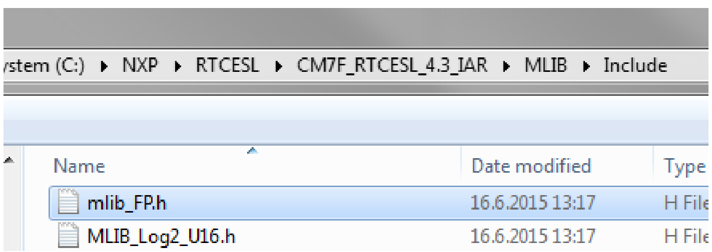

Figure 25. Add Files dialog

7. Now you will see the files added in the workspace. See Figure 26.

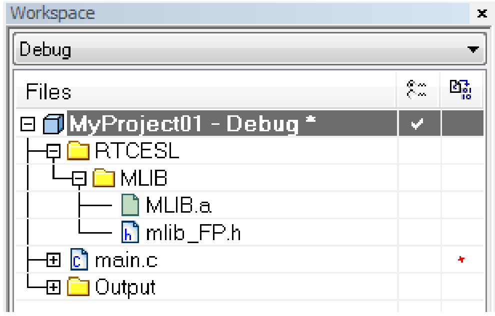

Figure 26. Project workspace

Library path setup

1. In the main menu, go to Project > Options…, and a dialog appears.
2. In the left-hand column, select C/C++ Compiler.
3. In the right-hand part of the dialog, click on the Preprocessor tab (it can be hidden in the right; use the arrow icons for navigation).
4. In the text box (at the Additional include directories title), type the following folder (using the created variable):
   - $RTCESL_LOC$\MLIB\Include
5. Click OK in the main dialog. See Figure 27.

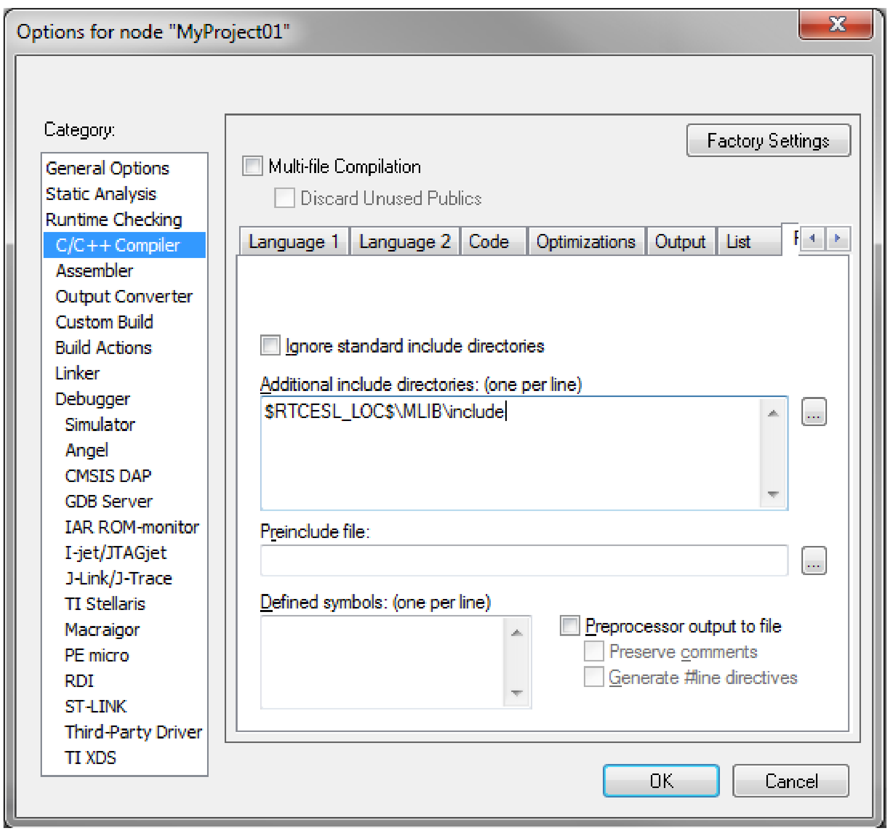

Figure 27. Library path adition

Type the #include syntax into the code. Include the library included into the *main.c* file. In the workspace tree, double-click the *main.c* file. After the *main.c* file opens up, include the following line into the #include section:

```
#include "mlib_FP.h"
```

When you click the Make icon, the project will be compiled without errors.

Chapter 2
Algorithms in detail

2.1 MLIB_Abs

The MLIB_Abs functions return the absolute value of the input. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_Abs}(x) = |x|$$

Figure 28. Algorithm formula

2.1.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The result may overflow.
- Floating-point output - the output is a floating-point number; the result is a non-negative value.

The available versions of the MLIB_Abs function are shown in the following table.

Table 2. Function versions

Function name	Input type	Result type	Description
MLIB_Abs_F16	frac16_t	frac16_t	Absolute value of a 16-bit fractional value. The output is within the range <-1 ; 1).
MLIB_Abs_F32	frac32_t	frac32_t	Absolute value of a 32-bit fractional value. The output is within the range <-1 ; 1).
MLIB_Abs_FLT	float_t	float_t	Absolute value of a 32-bit single precision floating-point value. The output is a non-negative value.

2.1.2 Declaration

The available MLIB_Abs functions have the following declarations:

```
frac16_t MLIB_Abs_F16(frac16_t f16Val)
frac32_t MLIB_Abs_F32(frac32_t f32Val)
float_t MLIB_Abs_FLT(float_t fltVal)
```

2.1.3 Function use

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

Fixed-point version:

```
#include "mlib.h"

static frac32_t f32Result;
static frac32_t f32Val;

void main(void)
{
```

```
   f32Val = FRAC32(-0.354);         /* f32Val = -0.354 */

   /* f32Result = |f32Val| */
   f32Result = MLIB_Abs_F32(f32Val);
}
```

Floating-point version:

```
#include "mlib.h"

static float_t fltResult;
static float_t fltVal;

void main(void)
{
   fltVal = -0.354F;         /* fltVal = -0.354 */

   /* fltResult = |fltVal| */
   fltResult = MLIB_Abs_FLT(fltVal);
}
```

2.2 MLIB_AbsSat

The MLIB_AbsSat functions return the absolute value of the input. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_AbsSat}(x) = |x|$$

Figure 29. Algorithm formula

2.2.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <0 ; 1). The result may saturate.

The available versions of the MLIB_AbsSat function are shown in the following table.

Table 3. Function versions

Function name	Input type	Result type	Description
MLIB_AbsSat_F16	frac16_t	frac16_t	Absolute value of a 16-bit fractional value. The output is within the range <0 ; 1).
MLIB_AbsSat_F32	frac32_t	frac32_t	Absolute value of a 32-bit fractional value. The output is within the range <0 ; 1).

2.2.2 Declaration

The available MLIB_AbsSat functions have the following declarations:

```
frac16_t MLIB_AbsSat_F16(frac16_t f16Val)
frac32_t MLIB_AbsSat_F32(frac32_t f32Val)
```

2.2.3 Function use

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

```
#include "mlib.h"

static frac16_t f16Val, f16Result;

void main(void)
{
  f16Val = FRAC16(-0.835);        /* f16Val = -0.835 */

  /*   f16Result = sat(|f16Val|)        */
  f16Result = MLIB_AbsSat_F16(f16Val);
}
```

2.3 MLIB_Add

The MLIB_Add functions return the sum of two addends. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_Add}(a, b) = a + b$$

Figure 30. Algorithm formula

2.3.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The result may overflow.
- Accumulator output with fractional inputs - the output is the accumulator type, where the result can be out of the range <-1 ; 1). The inputs are the fractional values only.
- Accumulator output with mixed inputs - the output is the accumulator type, where the result can be out of the range <-1 ; 1). The inputs are the accumulator and fractional values. The result may overflow.
- Floating-point output - the output is a floating-point number; the result is within the full range.

The available versions of the MLIB_Add function are shown in the following table.

Table 4. Function versions

Function name	Input type		Result type	Description
	Addend 1	Addend 2		
MLIB_Add_F16	frac16_t	frac16_t	frac16_t	Addition of two 16-bit fractional addends. The output is within the range <-1 ; 1).
MLIB_Add_F32	frac32_t	frac32_t	frac32_t	Addition of two 32-bit fractional addends. The output is within the range <-1 ; 1).
MLIB_Add_A32ss	frac16_t	frac16_t	acc32_t	Addition of two 16-bit fractional addends; the result is a 32-bit accumulator. The output may be out of the range <-2 ; 2>.
MLIB_Add_A32as	acc32_t	frac16_t	acc32_t	A 16-bit fractional addend is added to a 32-bit accumulator. The output may be out of the range <-2 ; 2>.

Table continues on the next page...

Table 4. Function versions (continued)

Function name	Input type		Result type	Description
	Addend 1	Addend 2		
MLIB_Add_FLT	float_t	float_t	float_t	Addition of two 32-bit single precision floating-point addends. The output is within the full range.

2.3.2 Declaration

The available MLIB_Add functions have the following declarations:

```
frac16_t MLIB_Add_F16(frac16_t f16Add1, frac16_t f16Add2)
frac32_t MLIB_Add_F32(frac32_t f32Add1, frac32_t f32Add2)
acc32_t MLIB_Add_A32ss(frac16_t f16Add1, frac16_t f16Add2)
acc32_t MLIB_Add_A32as(acc32_t a32Accum, frac16_t f16Add)
float_t MLIB_Add_FLT(float_t fltAdd1, float_t fltAdd2)
```

2.3.3 Function use

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

Fixed-point version:

```
#include "mlib.h"

static acc32_t a32Result;
static frac16_t f16Add1, f16Add2;

void main(void)
{
  f16Add1 = FRAC16(-0.8);        /* f16Add1 = -0.8 */
  f16Add2 = FRAC16(-0.5);        /* f16Add2 = -0.5 */

  /* a32Result = f16Add1 + f16Add2 */
  a32Result = MLIB_Add_A32ss(f16Add1, f16Add2);
}
```

Floating-point version:

```
#include "mlib.h"

static float_t fltResult;
static float_t fltAdd1, fltAdd2;

void main(void)
{
  fltAdd1 = -0.8F;        /* fltAdd1 = -0.8 */
  fltAdd2 = -0.5F;        /* fltAdd2 = -0.5 */

  /* fltResult = fltAdd1 + fltAdd2 */
  fltResult = MLIB_Add_FLT(fltAdd1, fltAdd2);
}
```

2.4 MLIB_AddSat

The MLIB_AddSat functions return the sum of two addends. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_AddSat}(a, b) = \begin{cases} 1, & a+b > 1 \\ -1, & a+b < -1 \\ a+b, & \text{else} \end{cases}$$

Figure 31. Algorithm formula

2.4.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_AddSat function are shown in the following table.

Table 5. Function versions

Function name	Input type		Result type	Description
	Addend 1	Addend 2		
MLIB_AddSat_F16	frac16_t	frac16_t	frac16_t	Addition of two 16-bit fractional addends. The output is within the range <-1 ; 1).
MLIB_AddSat_F32	frac32_t	frac32_t	frac32_t	Addition of two 32-bit fractional addends. The output is within the range <-1 ; 1).

2.4.2 Declaration

The available MLIB_AddSat functions have the following declarations:

```
frac16_t MLIB_Add_F16(frac16_t f16Add1, frac16_t f16Add2)
frac32_t MLIB_Add_F32(frac32_t f32Add1, frac32_t f32Add2)
```

2.4.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Add1, f32Add2, f32Result;

void main(void)
{
  f32Add1 = FRAC32(-0.8);         /* f32Add1 = -0.8 */
  f32Add2 = FRAC32(-0.5);         /* f32Add2 = -0.5 */

  /* f32Result = sat(f32Add1 + f32Add2) */
  f32Result = MLIB_AddSat_F32(f32Add1, f32Add2);
}
```

2.5 MLIB_Add4

The MLIB_Add4 functions return the sum of four addends. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_Add4}(a, b, c, d) = a + b + c + d$$

Figure 32. Algorithm formula

2.5.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The result may overflow.
- Floating-point output - the output is a floating-point number; the result is within the full range.

The available versions of the MLIB_Add4 function are shown in the following table.

Table 6. Function versions

Function name	Input type				Result type	Description
	Add. 1	Add. 2	Add. 3	Add. 4		
MLIB_Add4_F16	frac16_t	frac16_t	frac16_t	frac16_t	frac16_t	Addition of four 16-bit fractional addends. The output is within the range <-1 ; 1).
MLIB_Add4_F32	frac32_t	frac32_t	frac32_t	frac32_t	frac32_t	Addition of four 32-bit fractional addends. The output is within the range <-1 ; 1).
MLIB_Add4_FLT	float_t	float_t	float_t	float_t	float_t	Addition of four 32-bit single precision floating-point addends. The output is within the full range.

2.5.2 Declaration

The available MLIB_Add4 functions have the following declarations:

```
frac16_t MLIB_Add4_F16(frac16_t f16Add1, frac16_t f16Add2, frac16_t f16Add3, frac16_t f16Add4)

frac32_t MLIB_Add4_F32(frac32_t f32Add1, frac32_t f32Add2, frac32_t f32Add3, frac32_t f32Add4)


float_t MLIB_Add4_FLT(float_t fltAdd1, float_t fltAdd2, float_t fltAdd3, float_t fltAdd4)
```

2.5.3 Function use

The use of the MLIB_Add4 function is shown in the following examples:

Fixed-point version:

```
#include "mlib.h"

static frac32_t f32Result;
static frac32_t f32Add1, f32Add2, f32Add3, f32Add4;

void main(void)
```

```
{
   f32Add1 = FRAC32(-0.3);        /* f32Add1 = -0.3 */
   f32Add2 = FRAC32(0.5);         /* f32Add2 = 0.5 */
   f32Add3 = FRAC32(-0.2);        /* f32Add3 = -0.2 */
   f32Add4 = FRAC32(-0.4);        /* f32Add4 = -0.4 */

   /* f32Result = f32Add1 + f32Add2 + f32Add3 + f32Add4 */
   f32Result = MLIB_Add4_F32(f32Add1, f32Add2, f32Add3, f32Add4);
}
```

Floating-point version:

```
#include "mlib.h"

static float_t fltResult;
static float_t fltAdd1, fltAdd2, fltAdd3, fltAdd4;

void main(void)
{
   fltAdd1 = -0.3F;        /* fltAdd1 = -0.3 */
   fltAdd2 = 0.5F;         /* fltAdd2 = 0.5 */
   fltAdd3 = -0.2F;        /* fltAdd3 = -0.2 */
   fltAdd4 = -0.4F;        /* fltAdd4 = -0.4 */

   /* fltResult = fltAdd1 + fltAdd2 + fltAdd3 + fltAdd4 */
   fltResult = MLIB_Add4_FLT(fltAdd1, fltAdd2, fltAdd3, fltAdd4);
}
```

2.6 MLIB_Add4Sat

The MLIB_Add4Sat functions return the sum of four addends. The function saturates the output. See the following equation:

$$\text{MLIB\_Add4Sat}(a, b, c, d) = \begin{cases} 1, & a+b+c+d > 1 \\ -1, & a+b+c+d < -1 \\ a+b+c+d, & \text{else} \end{cases}$$

Figure 33. Algorithm formula

2.6.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_Add4Sat function are shown in the following table.

Table 7. Function versions

Function name	Input type				Result type	Description
	Add. 1	Add. 2	Add. 3	Add. 4		
MLIB_Add4Sat_F16	frac16_t	frac16_t	frac16_t	frac16_t	frac16_t	Addition of four 16-bit fractional addends. The output is within the range <-1 ; 1).

Table continues on the next page...

Table 7. Function versions (continued)

Function name	Input type				Result type	Description
	Add. 1	Add. 2	Add. 3	Add. 4		
MLIB_Add4Sat_F32	frac32_t	frac32_t	frac32_t	frac32_t	frac32_t	Addition of four 32-bit fractional addends. The output is within the range <-1 ; 1).

2.6.2 Declaration

The available MLIB_Add4Sat functions have the following declarations:

```
frac16_t MLIB_Add4Sat_F16(frac16_t f16Add1, frac16_t f16Add2, frac16_t f16Add3, frac16_t f16Add4)

frac32_t MLIB_Add4Sat_F32(frac32_t f32Add1, frac32_t f32Add2, frac32_t f32Add3, frac32_t f32Add4)
```

2.6.3 Function use

The use of the MLIB_Add4Sat function is shown in the following example:

```
#include "mlib.h"

static frac16_t f16Result, f16Add1, f16Add2, f16Add3, f16Add4;

void main(void)
{
   f16Add1 = FRAC16(-0.7);       /* f16Add1 = -0.7 */
   f16Add2 = FRAC16(0.9);        /* f16Add2 = 0.9 */
   f16Add3 = FRAC16(0.4);        /* f16Add3 = 0.4 */
   f16Add4 = FRAC16(0.7);        /* f16Add4 = 0.7 */

   /* f16Result = sat(f16Add1 + f16Add2 + f16Add3 + f16Add4) */
   f16Result = MLIB_Add4Sat_F16(f16Add1, f16Add2, f16Add3, f16Add4);
}
```

2.7 MLIB_Clb

The MLIB_Clb functions return the number of leading bits of the input. If the input is 0, it returns the size of the type minus one.

2.7.1 Available versions

This function is available in the following versions:

- Integer output with fractional input - the output is the unsigned integer value when the input is fractional; the result is greater than or equal to 0.

The available versions of the MLIB_Clb function are shown in the following table.

Table 8. Function versions

Function name	Input type	Result type	Description
MLIB_Clb_U16s	frac16_t	uint16_t	Counts the leading bits of a 16-bit fractional value. The output is within the range <0 ; 15>.
MLIB_Clb_U16l	frac32_t	uint16_t	Counts the leading bits of a 32-bit fractional value. The output is within the range <0 ; 31>.

2.7.2 Declaration

The available MLIB_Clb functions have the following declarations:

```
uint16_t MLIB_Clb_U16s(frac16_t f16Val)
uint16_t MLIB_Clb_U16l(frac32_t f32Val)
```

2.7.3 Function use

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

```
#include "mlib.h"

static uint16_t u16Result;
static frac32_t f32Val;

void main(void)
{
   f32Val = FRAC32(0.00000452);         /* f32Val = 0.00000452 */

   /* u16Result = clb(f32Val) */
   u16Result = MLIB_Clb_U16l(f32Val);
}
```

2.8 MLIB_Conv

The MLIB_Conv functions return the input value, converted to the output type.

2.8.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1).
- Accumulator output - the output is the accumulator type, where the result may be out of the range <-1 ; 1).
- Floating-point output - the output is a floating-point number; the result is within the range <-1 ; 1)

The available versions of the MLIB_Conv function are shown in the following table.

Table 9. Function versions

Function name	Input type	Result type	Description
MLIB_Conv_F16l	frac32_t	frac16_t	Conversion of a 32-bit fractional value to a 16-bit fractional value. The output is within the range <-1 ; 1).

Table continues on the next page...

Table 9. Function versions (continued)

Function name	Input type	Result type	Description
MLIB_Conv_F16f	float_t	frac16_t	Conversion of a 32-bit single precision floating-point value to a 16-bit fractional value. The output is within the range <-1 ; 1). If the result is out of this range, it is saturated.
MLIB_Conv_F32s	frac16_t	frac32_t	Conversion of a 16-bit fractional value to a 32-bit fractional value. The output is within the range <-1 ; 1).
MLIB_Conv_F32f	float_t	frac32_t	Conversion of a 32-bit single precision floating-point value to a 32-bit fractional value. The output is within the range <-1 ; 1). If the result is out of this range, it is saturated.
MLIB_Conv_A32f	float_t	acc32_t	Conversion of a 32-bit single precision floating-point value to a 32-bit accumulator value. The output is within the range <-65536.0 ; 65536.0). If the result is out of this range, it is saturated.
MLIB_Conv_FLTs	frac16_t	float_t	Conversion of a 16-bit fractional value to a 32-bit single precision floating-point value. The output is within the range <-1 ; 1).
MLIB_Conv_FLTl	frac32_t	float_t	Conversion of a 32-bit fractional value to a 32-bit single precision floating-point value. The output is within the range <-1 ; 1).
MLIB_Conv_FLTa	acc32_t	float_t	Conversion of a 32-bit accumulator value to a 32-bit single precision floating-point value. The output is within the range <-65536.0 ; 65536.0).

2.8.2 Declaration

The available MLIB_Conv functions have the following declarations:

```
frac16_t MLIB_Conv_F16l(frac32_t f32Val)
frac16_t MLIB_Conv_F16f(float_t fltVal)
frac32_t MLIB_Conv_F32s(frac16_t f16Val)
frac32_t MLIB_Conv_F32f(float_t fltVal)
acc32_t MLIB_Conv_A32f(float_t fltVal)
float_t MLIB_Conv_FLTs(frac16_t f16Val)
float_t MLIB_Conv_FLTl(frac32_t f32Val)
float_t MLIB_Conv_FLTa(acc32_t a32Val)
```

2.8.3 Function use

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

Fixed-point version:

```
#include "mlib.h"

static frac32_t f32Result;
static frac16_t f16Val;

void main(void)
{
   f16Val = FRAC16(-0.5);          /* f16Val = -0.5 */

   /* f32Result = (frac32_t)f16Val << 16 */
```

```
    f32Result = MLIB_Conv_F32s(f16Val);
}
```

Floating-point version:

```
#include "mlib.h"

static float_t fltResult;
static frac16_t f16Val;

void main(void)
{
    f16Val = FRAC16(0.3);          /* f16Val = 0.3 */

    /* fltResult = (float_t)(f16Val/2^15) */
    fltResult = MLIB_Conv_FLTs(f16Val);
}
```

2.9 MLIB_Div

The MLIB_Div functions return the fractional division of the numerator and denominator. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_Div}(a, b) = \begin{cases} \max, & a \geq 0 \ \wedge \ b = 0 \ \vee \ a \leq -0 \ \wedge \ b = -0 \\ \min, & a \leq -0 \ \wedge \ b = 0 \ \vee \ a \geq 0 \ \wedge \ b = -0 \\ \frac{a}{b}, & \text{else} \end{cases}$$

Figure 34. Algorithm formula

2.9.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The function is only defined for: |nominator| < |denominator|. The function returns undefined results out of this condition.
- Accumulator output - the output is the accumulator type, where the result may be out of the range <-1 ; 1).
- Floating-point output - the output is a floating-point number; the result is within the full range.

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

Table 10. Function versions

Function name	Input type		Result type	Description
	Num.	Denom.		
MLIB_Div_F16	frac16_t	frac16_t	frac16_t	Division of a 16-bit fractional numerator and denominator. The output is within the range <-1 ; 1).
MLIB_Div_F16ls	frac32_t	frac16_t	frac16_t	Division of a 32-bit fractional numerator by a 16-bit fractional denominator; the output is a 16-bit fractional result. The output is within the range <-1 ; 1).

Table continues on the next page...

Table 10. Function versions (continued)

Function name	Input type		Result type	Description
	Num.	Denom.		
MLIB_Div_F16ll	frac32_t	frac32_t	frac16_t	Division of a 32-bit fractional numerator and denominator; the output is a 16-bit fractional result. The output is within the range <-1 ; 1).
MLIB_Div_F32ls	frac32_t	frac16_t	frac32_t	Division of a 32-bit fractional numerator by a 16-bit fractional denominator; the output is a 32-bit fractional result. The output is within the range <-1 ; 1).
MLIB_Div_F32	frac32_t	frac32_t	frac32_t	Division of a 32-bit fractional numerator and denominator. The output is within the range <-1 ; 1).
MLIB_Div_A32ss	frac16_t	frac16_t	acc32_t	Division of a 16-bit fractional numerator and denominator; the output is a 32-bit accumulator result. The output may be out of the range <-65536 ; 65536).
MLIB_Div_A32ls	frac32_t	frac16_t	acc32_t	Division of a 32-bit fractional numerator by a 16-bit fractional denominator; the output is a 32-bit accumulator result. The output may be out of the range <-65536 ; 65536).
MLIB_Div_A32ll	frac32_t	frac32_t	acc32_t	Division of a 32-bit fractional numerator and denominator; the output is a 32-bit accumulator result. The output may be out of the range <-65536 ; 65536).
MLIB_Div_A32as	acc32_t	frac16_t	acc32_t	Division of a 32-bit accumulator numerator by a 16-bit fractional denominator; the output is a 32-bit accumulator result. The output may be out of the range <-65536 ; 65536).
MLIB_Div_FLT	float_t	float_t	float_t	Division of a 32-bit single precision floating-point numerator and denominator. The output is within the full range.

2.9.2 Declaration

The available MLIB_Div functions have the following declarations:

```
frac16_t MLIB_Div_F16(frac16_t f16Num, frac16_t f16Denom)
frac16_t MLIB_Div_F16ls(frac32_t f32Num, frac16_t f16Denom)
frac16_t MLIB_Div_F16ll(frac32_t f32Num, frac32_t f32Denom)
frac32_t MLIB_Div_F32ls(frac32_t f32Num, frac16_t f16Denom)
frac32_t MLIB_Div_F32(frac32_t f32Num, frac32_t f32Denom)
acc32_t MLIB_Div_A32ss(frac16_t f16Num, frac16_t f16Denom)
acc32_t MLIB_Div_A32ls(frac32_t f32Num, frac16_t f16Denom)
acc32_t MLIB_Div_A32ll(frac32_t f32Num, frac32_t f32Denom)
acc32_t MLIB_Div_A32as(acc32_t a32Num, frac16_t f16Denom)
float_t MLIB_Div_FLT(float_t fltNum, float_t fltDenom)
```

2.9.3 Function use

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

Fixed-point version:

```
#include "mlib.h"

static frac32_t f32Num, f32Result;
```

```
static frac16_t f16Denom;

void main(void)
{
  f32Num = FRAC32(0.2);          /* f32Num = 0.2 */
  f16Denom = FRAC16(-0.495);     /* f16Denom = -0.495 */

  /* f32Result = f32Num / f16Denom */
  f32Result = MLIB_Div_F32ls(f32Num, f16Denom);
}
```

Floating-point version:

```
#include "mlib.h"

static float_t fltNum, fltResult;
static float_t fltDenom;

void main(void)
{
  fltNum = 0.2F;          /* fltNum = 0.2 */
  fltDenom = -0.495F;     /* fltDenom = -0.495 */

  /* fltResult = fltNum / fltDenom */
  fltResult = MLIB_Div_FLT(fltNum, fltDenom);
}
```

2.10 MLIB_DivSat

The MLIB_DivSat functions return the fractional division of the numerator and denominator. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_DivSat}(a,b) = \begin{cases} max, & \frac{a}{b} > max \ \vee \ a \geq 0 \ \wedge \ b = 0 \\ min, & \frac{a}{b} < min \ \vee \ a < 0 \ \wedge \ b = 0 \\ \frac{a}{b}, & \text{else} \end{cases}$$

Figure 35. Algorithm formula

2.10.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The result may saturate.
- Accumulator output - the output is the accumulator type, where the result may be out of the range <-65536 ; 65536).

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

Table 11. Function versions

Function name	Input type		Result type	Description
	Num.	Denom.		
MLIB_DivSat_F16	frac16_t	frac16_t	frac16_t	Division of a 16-bit fractional numerator and denominator. The output is within the range <-1 ; 1).
MLIB_DivSat_F16ls	frac32_t	frac16_t	frac16_t	Division of a 32-bit fractional numerator by a 16-bit fractional denominator; the output is a 16-bit fractional result. The output is within the range <-1 ; 1).
MLIB_DivSat_F16ll	frac32_t	frac32_t	frac16_t	Division of a 32-bit fractional numerator and denominator; the output is a 16-bit fractional result. The output is within the range <-1 ; 1).
MLIB_DivSat_F32ls	frac32_t	frac16_t	frac32_t	Division of a 32-bit fractional numerator by a 16-bit fractional denominator; the output is a 32-bit fractional result. The output is within the range <-1 ; 1).
MLIB_DivSat_F32	frac32_t	frac32_t	frac32_t	Division of a 32-bit fractional numerator and denominator. The output is within the range <-1 ; 1).
MLIB_DivSat_A32as	acc32_t	frac16_t	acc32_t	Division of a 32-bit accumulator numerator by a 16-bit fractional denominator; the output is a 32-bit accumulator result. The output may be out of the range <-65536 ; 65536).

2.10.2 Declaration

The available MLIB_DivSat functions have the following declarations:

```
frac16_t MLIB_DivSat_F16(frac16_t f16Num, frac16_t f16Denom)
frac16_t MLIB_DivSat_F16ls(frac32_t f32Num, frac16_t f16Denom)
frac16_t MLIB_DivSat_F16ll(frac32_t f32Num, frac32_t f32Denom)
frac32_t MLIB_DivSat_F32ls(frac32_t f32Num, frac16_t f16Denom)
frac32_t MLIB_DivSat_F32(frac32_t f32Num, frac32_t f32Denom)
acc32_t MLIB_DivSat_A32as(acc32_t a32Num, frac16_t f16Denom)
```

2.10.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Num, f32Denom, f32Result;

void main(void)
{
  f32Num = FRAC32(0.4);          /* f32Num = 0.4 */
  f32Denom = FRAC32(-0.02);      /* f32Denom = -0.02 */

  /* f32Result = f32Num / f32Denom */
  f32Result = MLIB_DivSat_F32(f32Num, f32Denom);
}
```

2.11 MLIB_Div1Q

The MLIB_Div1Q functions return the single-quadrant fractional division of the numerator and denominator. The numerator and denominator must be non-negative numbers, otherwise the function returns undefined results. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_Div1Q}(a, b) = \begin{cases} \max, & a \geq 0 \ \wedge \ b = 0 \\ \frac{a}{b}, & a \geq 0 \ \wedge \ b > 0 \end{cases}$$

Figure 36. Algorithm formula

2.11.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <0 ; 1). The function is only defined for: nominator < denominator, and both are non-negative. The function returns undefined results out of this condition.
- Accumulator output - the output is the accumulator type, where the result is greater than or equal to 0.

The available versions of the MLIB_Div1Q function are shown in the following table:

Table 12. Function versions

Function name	Input type		Result type	Description
	Num.	Denom.		
MLIB_Div1Q_F16	frac16_t	frac16_t	frac16_t	Division of a non-negative 16-bit fractional numerator and denominator. The output is within the range <0 ; 1).
MLIB_Div1Q_F16ls	frac32_t	frac16_t	frac16_t	Division of a non-negative 32-bit fractional numerator by a non-negative 16-bit fractional denominator; the output is a non-negative 16-bit fractional result. The output is within the range <0 ; 1).
MLIB_Div1Q_F16ll	frac32_t	frac32_t	frac16_t	Division of a non-negative 32-bit fractional numerator and denominator; the output is a non-negative 16-bit fractional result. The output is within the range <0 ; 1).
MLIB_Div1Q_F32ls	frac32_t	frac16_t	frac32_t	Division of a non-negative 32-bit fractional numerator by a non-negative 16-bit fractional denominator; the output is a non-negative 32-bit fractional result. The output is within the range <0 ; 1).
MLIB_Div1Q_F32	frac32_t	frac32_t	frac32_t	Division of a non-negative 32-bit fractional numerator and denominator. The output is within the range <0 ; 1).
MLIB_Div1Q_A32ss	frac16_t	frac16_t	acc32_t	Division of a non-negative 16-bit fractional numerator and denominator; the output is a non-negative 32-bit accumulator result. The output is greater than or equal to 0.
MLIB_Div1Q_A32ls	frac32_t	frac16_t	acc32_t	Division of a non-negative 32-bit fractional numerator by a non-negative 16-bit fractional denominator; the output is a non-negative 32-bit accumulator result. The output is greater than or equal to 0.
MLIB_Div1Q_A32ll	frac32_t	frac32_t	acc32_t	Division of a non-negative 32-bit fractional numerator and denominator; the output is a non-negative 32-bit accumulator result. The output is greater than or equal to 0.
MLIB_Div1Q_A32as	acc32_t	frac16_t	acc32_t	Division of a non-negative 32-bit accumulator numerator by a non-negative 16-bit fractional denominator; the output is a 32-bit accumulator result. The output is greater than or equal to 0.

2.11.2 Declaration

The available MLIB_Div1Q functions have the following declarations:

```
frac16_t MLIB_Div1Q_F16(frac16_t f16Num, frac16_t f16Denom)
frac16_t MLIB_Div1Q_F16ls(frac32_t f32Num, frac16_t f16Denom)
frac16_t MLIB_Div1Q_F16ll(frac32_t f32Num, frac32_t f32Denom)
frac32_t MLIB_Div1Q_F32ls(frac32_t f32Num, frac16_t f16Denom)
frac32_t MLIB_Div1Q_F32(frac32_t f32Num, frac32_t f32Denom)
acc32_t MLIB_Div1Q_A32ss(frac16_t f16Num, frac16_t f16Denom)
acc32_t MLIB_Div1Q_A32ls(frac32_t f32Num, frac16_t f16Denom)
acc32_t MLIB_Div1Q_A32ll(frac32_t f32Num, frac32_t f32Denom)
acc32_t MLIB_Div1Q_A32as(acc32_t a32Num, frac16_t f16Denom)
```

2.11.3 Function use

The use of the MLIB_Div1Q function is shown in the following example:

```
#include "mlib.h"

static frac32_t f32Num, f32Denom, f32Result;

void main(void)
{
  f32Num = FRAC32(0.2);           /* f32Num = 0.2 */
  f32Denom = FRAC32(0.865);       /* f32Denom = 0.865 */

  /* f32Result = f32Num / f32Denom */
  f32Result = MLIB_Div1Q_F32(f32Num, f32Denom);
}
```

2.12 MLIB_Div1QSat

The MLIB_Div1QSat functions return the fractional division of the numerator and denominator. The numerator and denominator must be non-negative numbers. The function saturates the output. See the following equation:

$$\text{MLIB\_Div1QSat}(a, b) = \begin{cases} max, & \frac{a}{b} > max \ \wedge \ a \geq 0 \ \wedge \ b \geq 0 \\ \frac{a}{b}, & a \geq 0 \ \wedge \ b > 0 \end{cases}$$

Figure 37. Algorithm formula

2.12.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <0 ; 1). The result may saturate.
- Accumulator output - the output is the accumulator type, where the result is greater than or equal to 0.

The available versions of the MLIB_Div1QSat function are shown in the following table:

Table 13. Function versions

Function name	Input type		Result type	Description
	Num.	Denom.		
MLIB_Div1QSat_F16	frac16_t	frac16_t	frac16_t	Division of a non-negative 16-bit fractional numerator and denominator. The output is within the range <0 ; 1).
MLIB_Div1QSat_F16ls	frac32_t	frac16_t	frac16_t	Division of a non-negative 32-bit fractional numerator by a non-negative 16-bit fractional denominator; the output is a non-negative 16-bit fractional result. The output is within the range <0 ; 1).
MLIB_Div1QSat_F16ll	frac32_t	frac32_t	frac16_t	Division of a non-negative 32-bit fractional numerator and denominator; the output is a non-negative 16-bit fractional result. The output is within the range <0 ; 1).
MLIB_Div1QSat_F32ls	frac32_t	frac16_t	frac32_t	Division of a non-negative 32-bit fractional numerator by a non-negative 16-bit fractional denominator; the output is a non-negative 32-bit fractional result. The output is within the range <0 ; 1).
MLIB_Div1QSat_F32	frac32_t	frac32_t	frac32_t	Division of a non-negative 32-bit fractional numerator and denominator. The output is within the range <0 ; 1).
MLIB_Div1QSat_A32as	acc32_t	frac16_t	acc32_t	Division of a non-negative 32-bit accumulator numerator by a non-negative 16-bit fractional denominator; the output is a 32-bit accumulator result. The output is greater than or equal to 0.

2.12.2 Declaration

The available MLIB_Div1QSat functions have the following declarations:

```
frac16_t MLIB_Div1QSat_F16(frac16_t f16Num, frac16_t f16Denom)
frac16_t MLIB_Div1QSat_F16ls(frac32_t f32Num, frac16_t f16Denom)
frac16_t MLIB_Div1QSat_F16ll(frac32_t f32Num, frac32_t f32Denom)
frac32_t MLIB_Div1QSat_F32ls(frac32_t f32Num, frac16_t f16Denom)
frac32_t MLIB_Div1QSat_F32(frac32_t f32Num, frac32_t f32Denom)
acc32_t MLIB_Div1QSat_A32as(acc32_t a32Num, frac16_t f16Denom)
```

2.12.3 Function use

The use of the MLIB_Div1QSat function is shown in the following example:

```
#include "mlib.h"

static frac32_t f32Num, f32Result;
static frac16_t f16Denom;

void main(void)
{
  f32Num = FRAC32(0.02);         /* f32Num = 0.02 */
  f16Denom = FRAC16(0.4);        /* f16Denom = 0.4 */

  /* f32Result = f32Num / f16Denom */
```

```
  f32Result = MLIB_Div1QSat_F32ls(f32Num, f16Denom);
}
```

2.13 MLIB_Log2

The MLIB_Log2 functions return the binary logarithm of the input. See the following equation:

$$\mathrm{MLIB\_Log2}(x) = \begin{cases} 0, & x \le 1 \\ \mathrm{Log}_2(x), & \text{else} \end{cases}$$

Figure 38. Algorithm formula

2.13.1 Available versions

This function is available in the following versions:

- Unsigned integer output - the output is the unsigned integer result.

The available versions of the MLIB_Log2 function are shown in the following table.

Table 14. Function versions

Function name	Input type	Result type	Description
MLIB_Log2_U16	uint16_t	uint16_t	Binary logarithm of a 16-bit unsigned integer value. The output is greater than or equal to 0.

2.13.2 Declaration

The available MLIB_Log2 functions have the following declarations:

```
uint16_t MLIB_Log2_U16(uint16_t u16Val)
```

2.13.3 Function use

The use of the MLIB_Log2 function is shown in the following example:

```
#include "mlib.h"

static uint16_t u16Result, u16Val;

void main(void)
{
   u16Val = 5;             /* u16Val = 5 */

   /* u16Result = log2(u16Val) */
    u16Result = MLIB_Log2_U16(u16Val);
}
```

2.14 MLIB_Mac

The MLIB_Mac functions return the sum of the input accumulator, and the fractional product of two multiplicands. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_Mac}(a, b, c) = a + b \cdot c$$

Figure 39. Algorithm formula

2.14.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The result may overflow.
- Accumulator output with mixed inputs - the output is the accumulator type, where the result can be out of the range <-1 ; 1). The accumulator is the accumulator type, the multiplicands are the fractional types. The result may overflow.
- Floating-point output - the output is a floating-point number; the result is within the full range.

The available versions of the MLIB_Mac function are shown in the following table.

Table 15. Function versions

Function name	Input type			Result type	Description
	Accum.	Mult. 1	Mult. 2		
MLIB_Mac_F16	frac16_t	frac16_t	frac16_t	frac16_t	The upper 16-bit portion [16..31] of the fractional product (of two 16-bit fractional multiplicands) is added to a 16-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_Mac_F32lss	frac32_t	frac16_t	frac16_t	frac32_t	The 32-bit fractional product (of two 16-bit fractional multiplicands) is added to a 32-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_Mac_F32	frac32_t	frac32_t	frac32_t	frac32_t	The upper 32-bit portion [32..63] of the fractional product (of two 32-bit fractional multiplicands) is added to a 32-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_Mac_A32ass	acc32_t	frac16_t	frac16_t	acc32_t	The upper 16-bit portion [16..31] of the fractional product (of two 16-bit fractional multiplicands) is added to a 32-bit accumulator. The output may be out of the range <-65536 ; 65536).
MLIB_Mac_FLT	float_t	float_t	float_t	float_t	The product (of two 32-bit single-point floating-point multiplicands) is added to a single-point floating-point accumulator. The output is within the full range.

2.14.2 Declaration

The available MLIB_Mac functions have the following declarations:

```
frac16_t MLIB_Mac_F16(frac16_t f16Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_Mac_F32lss(frac32_t f32Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_Mac_F32(frac32_t f32Accum, frac32_t f32Mult1, frac32_t f32Mult2)
acc32_t MLIB_Mac_A32ass(acc32_t a32Accum, frac16_t f16Mult1, frac16_t f16Mult2)
float_t MLIB_Mac_FLT(float_t fltAccum, float_t fltMult1, float_t fltMult2)
```

2.14.3 Function use

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

Fixed-point version:

```
#include "mlib.h"

static frac32_t f32Accum, f32Result;
static frac16_t f16Mult1, f16Mult2;

void main(void)
{
  f32Accum = FRAC32(0.3);               /* f32Accum = 0.3 */
  f16Mult1 = FRAC16(0.1);               /* f16Mult1 = 0.1 */
  f16Mult2 = FRAC16(-0.2);              /* f16Mult2 = -0.2 */

  /* f32Result = f32Accum + f16Mult1 * f16Mult2 */
  f32Result = MLIB_Mac_F32lss(f32Accum, f16Mult1, f16Mult2);
}
```

Floating-point version:

```
#include "mlib.h"

static float_t fltAccum, fltResult;
static float_t fltMult1, fltMult2;

void main(void)
{
  fltAccum = 0.3F;               /* fltAccum = 0.3 */
  fltMult1 = 0.1F;               /* fltMult1 = 0.1 */
  fltMult2 = -0.2F;              /* fltMult2 = -0.2 */

  /* fltResult = fltAccum + fltMult1 * fltMult2 */
  fltResult = MLIB_Mac_FLT(fltAccum, fltMult1, fltMult2);
}
```

2.15 MLIB_MacSat

The MLIB_MacSat functions return the sum of the input accumulator and the fractional product of two multiplicands. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_MacSat}(a, b, c) = \begin{cases} 1, & a + b \bullet c > 1 \\ -1, & a + b \bullet c < -1 \\ a + b \bullet c, & \text{else} \end{cases}$$

Figure 40. Algorithm formula

2.15.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_MacSat function are shown in the following table.

Table 16. Function versions

Function name	Input type			Result type	Description
	Accum.	Mult. 1	Mult. 2		
MLIB_MacSat_F16	frac16_t	frac16_t	frac16_t	frac16_t	The upper 16-bit portion [16..31] of the fractional product (of two 16-bit fractional multiplicands) is added to a 16-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_MacSat_F32lss	frac32_t	frac16_t	frac16_t	frac32_t	The 32-bit fractional product (of two 16-bit fractional multiplicands) is added to a 32-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_MacSat_F32	frac32_t	frac32_t	frac32_t	frac32_t	The upper 32-bit portion [32..63] of the fractional product (of two 32-bit fractional multiplicands) is added to a 32-bit fractional accumulator. The output is within the range <-1 ; 1).

2.15.2 Declaration

The available MLIB_MacSat functions have the following declarations:

```
frac16_t MLIB_MacSat_F16(frac16_t f16Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_MacSat_F32lss(frac32_t f32Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_MacSat_F32(frac32_t f32Accum, frac32_t f32Mult1, frac32_t f32Mult2)
```

2.15.3 Function use

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

```
#include "mlib.h"

static frac16_t f16Mult1, f16Mult2;
static frac32_t f32Accum, f32Result;

void main(void)
{
   f32Accum = FRAC32(-0.7);                /* f32Accum = -0.7 */
   f16Mult1 = FRAC16(-1.0);                /* f16Mult1 = -1.0 */
   f16Mult2 = FRAC16(0.8);                 /* f16Mult2 = 0.8 */

   /* f32Result = sat(f32Accum + f16Mult1 * f16Mult2) */
   f32Result = MLIB_MacSat_F32lss(f32Accum, f16Mult1, f16Mult2);
}
```

2.16 MLIB_MacRnd

The MLIB_MacRnd functions return the sum of the input accumulator and the rounded fractional product of two multiplicands. The round method is the round to nearest. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_MacRnd}(a, b, c) = a + \mathrm{round}(b \cdot c)$$

Figure 41. Algorithm formula

2.16.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The result may overflow.
- Accumulator output with mixed inputs - the output is the accumulator type where the result can be out of the range <-1 ; 1). The accumulator is the accumulator type, the multiplicands are the fractional types. The result may overflow.

The available versions of the MLIB_MacRnd function are shown in the following table.

Table 17. Function versions

Function name	Input type			Result type	Description
	Accum.	Mult. 1	Mult. 2		
MLIB_MacRnd_F16	frac16_t	frac16_t	frac16_t	frac16_t	The fractional product (of two 16-bit fractional multiplicands), rounded to the upper 16 bits, is added to a 16-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_MacRnd_F32lls	frac32_t	frac32_t	frac16_t	frac32_t	The fractional product (of a 32-bit and 16-bit fractional multiplicand), rounded to the upper 32 bits [16..48], is added to a 32-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_MacRnd_F32	frac32_t	frac32_t	frac32_t	frac32_t	The fractional product (of two 32-bit fractional multiplicands), rounded to the upper 32 bits [32..63], is added to a 32-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_MacRnd_A32ass	acc32_t	frac16_t	frac16_t	acc32_t	The fractional product (of two 16-bit fractional multiplicands), rounded to the upper 16 bits [16..31], is added to a 32-bit accumulator. The output may be out of the range <-65536 ; 65536).

2.16.2 Declaration

The available MLIB_MacRnd functions have the following declarations:

```
frac16_t MLIB_MacRnd_F16(frac16_t f16Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_MacRnd_F32lls(frac32_t f32Accum, frac32_t f32Mult1, frac16_t f16Mult2)
frac32_t MLIB_MacRnd_F32(frac32_t f32Accum, frac32_t f32Mult1, frac32_t f32Mult2)
acc32_t MLIB_MacRnd_A32ass(acc32_t a32Accum, frac16_t f16Mult1, frac16_t f16Mult2)
```

2.16.3 Function use

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

```
#include "mlib.h"

static frac16_t f16Accum, f16Mult1, f16Mult2, f16Result;

void main(void)
{
   f16Accum = FRAC16(0.3);                /* f16Accum = 0.3 */
   f16Mult1 = FRAC16(0.1);                /* f16Mult1 = 0.1 */
   f16Mult2 = FRAC16(-0.2);               /* f16Mult2 = -0.2 */
```

```
    /* f16Result = round(f16Accum + f16Mult1 * f16Mult2) */
    f16Result = MLIB_MacRnd_F16(f16Accum, f16Mult1, f16Mult2);
}
```

2.17 MLIB_MacRndSat

The MLIB_MacRndSat functions return the sum of the input accumulator and the rounded fractional product of two multiplicands. The round method is the round to nearest. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_MacRndSat}(a, b, c) = \begin{cases} 1, & a + \mathrm{round}(b \cdot c) > 1 \\ -1, & a + \mathrm{round}(b \cdot c) < -1 \\ a + \mathrm{round}(b \cdot c), & \text{else} \end{cases}$$

Figure 42. Algorithm formula

2.17.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_MacRndSat function are shown in the following table.

Table 18. Function versions

Function name	Input type			Result type	Description
	Accum.	Mult. 1	Mult. 2		
MLIB_MacRndSat_F16	frac16_t	frac16_t	frac16_t	frac16_t	The fractional product (of two 16-bit fractional multiplicands), rounded to the upper 16 bits, is added to a 16-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_MacRndSat_F32lls	frac32_t	frac32_t	frac16_t	frac32_t	The fractional product (of a 32-bit and 16-bit fractional multiplicands), rounded to the upper 32 bits [16..48], is added to a 32-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_MacRndSat_F32	frac32_t	frac32_t	frac32_t	frac32_t	The fractional product (of two 32-bit fractional multiplicands), rounded to the upper 32 bits [32..63], is added to a 32-bit fractional accumulator. The output is within the range <-1 ; 1).

2.17.2 Declaration

The available MLIB_MacRndSat functions have the following declarations:

```
frac16_t MLIB_MacRndSat_F16(frac16_t f16Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_MacRndSat_F32lls(frac32_t f32Accum, frac32_t f32Mult1, frac16_t f16Mult2)
frac32_t MLIB_MacRndSat_F32(frac32_t f32Accum, frac32_t f32Mult1, frac32_t f32Mult2)
```

2.17.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Accum, f32Mult1, f32Mult2, f32Result;

void main(void)
{
   f32Accum = FRAC32(-0.7);              /* f32Accum = -0.7 */
   f32Mult1 = FRAC32(-1.0);              /* f32Mult1 = -1.0 */
   f32Mult2 = FRAC32(0.8);               /* f32Mult2 = 0.8 */

   /* f32Result = sat(round(f32Accum + f32Mult1 * f32Mult2)) */
   f32Result = MLIB_MacRndSat_F32(f32Accum, f32Mult1, f32Mult2);
}
```

2.18 MLIB_Mac4

The MLIB_Mac4 functions return the sum of two products of two pairs of multiplicands. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_Mac4}(a, b, c, d) = a \cdot b + c \cdot d$$

Figure 43. Algorithm formula

2.18.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The result may overflow.
- Floating-point output - the output is a floating-point number; the result is within the full range.

The available versions of the MLIB_Mac4 function are shown in the following table.

Table 19. Function versions

Function name	Input type				Result type	Description
	Product 1		Product 2			
	Mult. 1	Mult. 2	Mult. 1	Mult. 2		
MLIB_Mac4_F32ssss	frac16_t	frac16_t	frac16_t	frac16_t	frac32_t	Addition of two 32-bit fractional products (of two 16-bit fractional multiplicands). The output is within the range <-1 ; 1).
MLIB_Mac4_FLT	float_t	float_t	float_t	float_t	float_t	Addition of two 32-bit single-point floating-point products (of two 32-bit single-point floating-point multiplicands). The output is within the full range.

2.18.2 Declaration

The available MLIB_Mac4 functions have the following declarations:

```
frac32_t MLIB_Mac4_F32ssss(frac16_t f16Add1Mult1, frac16_t f16Add1Mult2, frac16_t f16Add2Mult1, 
frac16_t f16Add2Mult2)

float_t MLIB_Mac4_FLT(float_t fltAdd1Mult1, float_t fltAdd1Mult2, float_t fltAdd2Mult1, 
float_t fltAdd2Mult2)
```

2.18.3 Function use

The use of the MLIB_Mac4 function is shown in the following examples:

Fixed-point version:

```
#include "mlib.h"

static frac32_t f32Result;
static frac16_t f16Add1Mult1, f16Add1Mult2, f16Add2Mult1, f16Add2Mult2;

void main(void)
{
   f16Add1Mult1 = FRAC16(0.2);          /* f16Add1Mult1 = 0.2 */
   f16Add1Mult2 = FRAC16(-0.7);         /* f16Add1Mult2 = -0.7 */
   f16Add2Mult1 = FRAC16(0.3);          /* f16Add2Mult1 = 0.3 */
   f16Add2Mult2 = FRAC16(-0.25);        /* f16Add2Mult2 = -0.25 */

   /* f32Result = f16Add1Mult1 * f16Add1Mult2 + f16Add2Mult1 * f16Add2Mult2*/
   f32Result = MLIB_Mac4_F32ssss(f16Add1Mult1, f16Add1Mult2, f16Add2Mult1, 
f16Add2Mult2);
}
```

Floating-point version:

```
#include "mlib.h"

static float_t fltResult;
static float_t fltAdd1Mult1, fltAdd1Mult2, fltAdd2Mult1, fltAdd2Mult2;

void main(void)
{
   fltAdd1Mult1 = 0.2F;          /* fltAdd1Mult1 = 0.2 */
   fltAdd1Mult2 = -0.7F;         /* fltAdd1Mult2 = -0.7 */
   fltAdd2Mult1 = 0.3F;          /* fltAdd2Mult1 = 0.3 */
   fltAdd2Mult2 = -0.25F;        /* fltAdd2Mult2 = -0.25 */

   /* fltResult = fltAdd1Mult1 * fltAdd1Mult2 + fltAdd2Mult1 * fltAdd2Mult2*/
   fltResult = MLIB_Mac4_FLT(fltAdd1Mult1, fltAdd1Mult2, fltAdd2Mult1, fltAdd2Mult2);
}
```

2.19 MLIB_Mac4Sat

The MLIB_Mac4Sat functions return the sum of two products of two pairs of multiplicands. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_Mac4Sat}(a, b, c, d) = \begin{cases} 1, & a \cdot b + c \cdot d > 1 \\ -1, & a \cdot b + c \cdot d < -1 \\ a \cdot b + c \cdot d, & \text{else} \end{cases}$$

Figure 44. Algorithm formula

2.19.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_Mac4Sat function are shown in the following table.

Table 20. Function versions

Function name	Input type				Result type	Description
	Product 1		Product 2			
	Mult. 1	Mult. 2	Mult. 1	Mult. 2		
MLIB_Mac4Sat_F32ssss	frac16_t	frac16_t	frac16_t	frac16_t	frac32_t	Addition of two 32-bit fractional products (of two 16-bit fractional multiplicands). The output is within the range <-1 ; 1).

2.19.2 Declaration

The available MLIB_Mac4Sat functions have the following declarations:

```
frac32_t MLIB_Mac4Sat_F32ssss(frac16_t f16Add1Mult1, frac16_t f16Add1Mult2, frac16_t f16Add2Mult1,
frac16_t f16Add2Mult2)
```

2.19.3 Function use

The use of the MLIB_Mac4Sat function is shown in the following example:

```
#include "mlib.h"

static frac32_t f32Result;
static frac16_t f16Add1Mult1, f16Add1Mult2, f16Add2Mult1, f16Add2Mult2;

void main(void)
{
   f16Add1Mult1 = FRAC16(-1.0);          /* f16Add1Mult1 = -1.0 */
   f16Add1Mult2 = FRAC16(-0.9);          /* f16Add1Mult2 = -0.9 */
   f16Add2Mult1 = FRAC16(0.8);           /* f16Add2Mult1 = 0.8 */
   f16Add2Mult2 = FRAC16(0.7);           /* f16Add2Mult2 = 0.7 */
```

```
    /* f32Result = sat(f16Add1Mult1 * f16Add1Mult2 + f16Add2Mult1 * f16Add2Mult2) */
    f32Result = MLIB_Mac4Sat_F32ssss(f16Add1Mult1, f16Add1Mult2, f16Add2Mult1,
f16Add2Mult2);
}
```

2.20 MLIB_Mac4Rnd

The MLIB_Mac4Rnd functions return the rounded sum of two products of two pairs of multiplicands. The round method is the round to nearest. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_Mac4Rnd}(a, b, c, d) = \mathrm{round}(a \cdot b + c \cdot d)$$

Figure 45. Algorithm formula

2.20.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_Mac4Rnd function are shown in the following table.

Table 21. Function versions

Function name	Input type				Result type	Description
	Product 1		Product 2			
	Mult. 1	Mult. 2	Mult. 1	Mult. 2		
MLIB_Mac4Rnd_F16	frac16_t	frac16_t	frac16_t	frac16_t	frac16_t	Addition of two 16-bit fractional products (of two 16-bit fractional multiplicands), rounded to the upper 16 bits. The output is within the range <-1 ; 1).
MLIB_Mac4Rnd_F32	frac32_t	frac32_t	frac32_t	frac32_t	frac32_t	Addition of two 32-bit fractional products (of two 32-bit fractional multiplicands), rounded to the upper 32 bits. The output is within the range <-1 ; 1).

2.20.2 Declaration

The available MLIB_Mac4Rnd functions have the following declarations:

```
frac16_t MLIB_Mac4Rnd_F16(frac16_t f16Add1Mult1, frac16_t f16Add1Mult2, frac16_t f16Add2Mult1,
frac16_t f16Add2Mult2)

frac32_t MLIB_Mac4Rnd_F32(frac32_t f32Add1Mult1, frac32_t f32Add1Mult2, frac32_t f32Add2Mult1,
frac32_t f32Add2Mult2)
```

2.20.3 Function use

The use of the MLIB_Mac4Rnd function is shown in the following example:

```
#include "mlib.h"

static frac16_t f16Result, f16Add1Mult1, f16Add1Mult2, f16Add2Mult1, f16Add2Mult2;

void main(void)
{
   f16Add1Mult1 = FRAC16(0.256);        /* f16Add1Mult1 = 0.256 */
   f16Add1Mult2 = FRAC16(-0.724);       /* f16Add1Mult2 = -0.724 */
   f16Add2Mult1 = FRAC16(0.365);        /* f16Add2Mult1 = 0.365 */
   f16Add2Mult2 = FRAC16(-0.25);        /* f16Add2Mult2 = -0.25 */

   /* f16Result = round(f16Add1Mult1 * f16Add1Mult2 + f16Add2Mult1 * f16Add2Mult2) */
   f16Result = MLIB_Mac4Rnd_F16(f16Add1Mult1, f16Add1Mult2, f16Add2Mult1,
f16Add2Mult2);
}
```

2.21 MLIB_Mac4RndSat

The MLIB_Mac4RndSat functions return the rounded sum of two products of two pairs of multiplicands. The round method is the round to nearest. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_Mac4RndSat}(a, b, c, d) = \begin{cases} 1, & \mathrm{round}(a \cdot b + c \cdot d) > 1 \\ -1, & \mathrm{round}(a \cdot b + c \cdot d) < -1 \\ \mathrm{round}(a \cdot b + c \cdot d), & \text{else} \end{cases}$$

Figure 46. Algorithm formula

2.21.1 Available versions

The function is available in the following versions:

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

The available versions of the MLIB_Mac4RndSat function are shown in the following table.

Table 22. Function versions

Function name	Input type				Result type	Description
	Product 1		Product 2			
	Mult. 1	Mult. 2	Mult. 1	Mult. 2		
MLIB_Mac4RndSat_F16	frac16_t	frac16_t	frac16_t	frac16_t	frac16_t	Addition of two 16-bit fractional products (of two 16-bit fractional multiplicands), rounded to the upper 16 bits. The output is within the range <-1 ; 1).
MLIB_Mac4RndSat_F32	frac32_t	frac32_t	frac32_t	frac32_t	frac32_t	Addition of two 32-bit fractional products (of two 32-bit fractional multiplicands), rounded to the upper 32 bits. The output is within the range <-1 ; 1).

2.21.2 Declaration

The available MLIB_Mac4RndSat functions have the following declarations:

```
frac16_t MLIB_Mac4RndSat_F16(frac16_t f16Add1Mult1, frac16_t f16Add1Mult2, frac16_t f16Add2Mult1,
frac16_t f16Add2Mult2)

frac32_t MLIB_Mac4RndSat_F32(frac32_t f32Add1Mult1, frac32_t f32Add1Mult2, frac32_t f32Add2Mult1,
frac32_t f32Add2Mult2)
```

2.21.3 Function use

The use of the MLIB_Mac4RndSat function is shown in the following example:

```
#include "mlib.h"

static frac32_t f32Result, f32Add1Mult1, f32Add1Mult2, f32Add2Mult1, f32Add2Mult2;

void main(void)
{
   f32Add1Mult1 = FRAC32(-1.0);         /* f32Add1Mult1 = -1.0 */
   f32Add1Mult2 = FRAC32(-0.9);         /* f32Add1Mult2 = -0.9 */
   f32Add2Mult1 = FRAC32(0.8);          /* f32Add2Mult1 = 0.8 */
   f32Add2Mult2 = FRAC32(0.7);          /* f32Add2Mult2 = 0.7 */

   /* f32Result = sat(round(f32Add1Mult1 * f32Add1Mult2 + f32Add2Mult1 *
f32Add2Mult2))*/
   f32Result = MLIB_Mac4RndSat_F32(f32Add1Mult1, f32Add1Mult2, f32Add2Mult1,
f32Add2Mult2);
}
```

2.22 MLIB_Mnac

The MLIB_Mnac functions return the product of two multiplicands minus the input accumulator. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_Mnac}(a, b, c) = b \cdot c - a$$

Figure 47. Algorithm formula

2.22.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The result may overflow.
- Accumulator output with mixed inputs - the output is the accumulator type, where the result can be out of the range <-1 ; 1). The accumulator is the accumulator type, the multiplicands are the fractional types. The result may overflow.
- Floating-point output - the output is a floating-point number; the result is within the full range.

The available versions of the MLIB_Mnac function are shown in the following table.

Table 23. Function versions

Function name	Input type			Result type	Description
	Accum.	Mult. 1	Mult. 2		
MLIB_Mnac_F16	frac16_t	frac16_t	frac16_t	frac16_t	The 16-bit fractional accumulator is subtracted from the upper 16-bit portion [16..31] of the fractional product (of two 16-bit fractional multiplicands). The output is within the range <-1 ; 1).
MLIB_Mnac_F32lss	frac32_t	frac16_t	frac16_t	frac32_t	The 32-bit fractional accumulator is subtracted from the 32-bit fractional product (of two 16-bit fractional multiplicands). The output is within the range <-1 ; 1).
MLIB_Mnac_F32	frac32_t	frac32_t	frac32_t	frac32_t	The 32-bit fractional accumulator is subtracted from the upper 32-bit portion [32..63] of the fractional product (of two 32-bit fractional multiplicands). The output is within the range <-1 ; 1).
MLIB_Mnac_A32ass	acc32_t	frac16_t	frac16_t	acc32_t	The 32-bit accumulator is subtracted from the upper 16-bit portion [16..31] of the fractional product (of two 16-bit fractional multiplicands). The output may be out of the range <-65536 ; 65536).
MLIB_Mnac_FLT	float_t	float_t	float_t	float_t	The single-point floating-point accumulator is subtracted from the product (of two 32-bit single-point floating-point multiplicands). The output is within the full range.

2.22.2 Declaration

The available MLIB_Mnac functions have the following declarations:

```
frac16_t MLIB_Mnac_F16(frac16_t f16Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_Mnac_F32lss(frac32_t f32Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_Mnac_F32(frac32_t f32Accum, frac32_t f32Mult1, frac32_t f32Mult2)
acc32_t MLIB_Mnac_A32ass(acc32_t a32Accum, frac16_t f16Mult1, frac16_t f16Mult2)
float_t MLIB_Mnac_FLT(float_t fltAccum, float_t fltMult1, float_t fltMult2)
```

2.22.3 Function use

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

Fixed-point version:

```
#include "mlib.h"

static frac32_t f32Accum, f32Result;
static frac16_t f16Mult1, f16Mult2;

void main(void)
{
  f32Accum = FRAC32(0.3);                /* f32Accum = 0.3 */
  f16Mult1 = FRAC16(0.1);                /* f16Mult1 = 0.1 */
  f16Mult2 = FRAC16(-0.2);               /* f16Mult2 = -0.2 */

  /* f32Result = f16Mult1 * f16Mult2 - f32Accum */
```

```
  f32Result = MLIB_Mnac_F32lss(f32Accum, f16Mult1, f16Mult2);
}
```

Floating-point version:

```
#include "mlib.h"

static float_t fltAccum, fltResult;
static float_t fltMult1, fltMult2;

void main(void)
{
  fltAccum = 0.3F;              /* fltAccum = 0.3 */
  fltMult1 = 0.1F;              /* fltMult1 = 0.1 */
  fltMult2 = -0.2F;             /* fltMult2 = -0.2 */

  /* fltResult = fltMult1 * fltMult2 - fltAccum */
  fltResult = MLIB_Mnac_FLT(fltAccum, fltMult1, fltMult2);
}
```

2.23 MLIB_MnacSat

The MLIB_MnacSat functions return the product of two multiplicands minus the input accumulator. The function saturates the output. See the following equation:

$$\text{MLIB\_MnacSat}(a, b, c) = \begin{cases} 1, & b \bullet c - a > 1 \\ -1, & b \bullet c - a < -1 \\ b \bullet c - a, & \text{else} \end{cases}$$

Figure 48. Algorithm formula

2.23.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_MnacSat function are shown in the following table.

Table 24. Function versions

Function name	Input type			Result type	Description
	Accum.	Mult. 1	Mult. 2		
MLIB_MnacSat_F16	frac16_t	frac16_t	frac16_t	frac16_t	The 16-bit fractional accumulator is subtracted from the upper 16-bit portion [16..31] of the fractional product (of two 16-bit fractional multiplicands). The output is within the range <-1 ; 1).
MLIB_MnacSat_F32lss	frac32_t	frac16_t	frac16_t	frac32_t	The 32-bit fractional accumulator is subtracted from the 32-bit fractional product (of two 16-bit fractional multiplicands). The output is within the range <-1 ; 1).

Table continues on the next page...

Table 24. Function versions (continued)

Function name	Input type			Result type	Description
	Accum.	Mult. 1	Mult. 2		
MLIB_MnacSat_F32	frac32_t	frac32_t	frac32_t	frac32_t	The 32-bit fractional accumulator is subtracted from the upper 32-bit portion [32..63] of the fractional product (of two 32-bit fractional multiplicands). The output is within the range <-1 ; 1).

2.23.2 Declaration

The available MLIB_MnacSat functions have the following declarations:

```
frac16_t MLIB_MnacSat_F16(frac16_t f16Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_MnacSat_F32lss(frac32_t f32Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_MnacSat_F32(frac32_t f32Accum, frac32_t f32Mult1, frac32_t f32Mult2)
```

2.23.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Accum, f32Result;
static frac16_t f16Mult1, f16Mult2;

void main(void)
{
  f32Accum = FRAC32(0.3);              /* f32Accum = 0.3 */
  f16Mult1 = FRAC16(0.1);              /* f16Mult1 = 0.1 */
  f16Mult2 = FRAC16(-0.2);             /* f16Mult2 = -0.2 */

  /* f32Result = f16Mult1 * f16Mult2 - f32Accum */
  f32Result = MLIB_MnacSat_F32lss(f32Accum, f16Mult1, f16Mult2);
}
```

2.24 MLIB_MnacRnd

The MLIB_MnacRnd functions return the rounded product of two multiplicands minus the input accumulator. The round method is the round to nearest. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_MnacRnd}(a, b, c) = \mathrm{round}(b \cdot c) - a$$

Figure 49. Algorithm formula

2.24.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The result may overflow.
- Accumulator output with mixed inputs - the output is the accumulator type, where the result can be out of the range <-1 ; 1). The accumulator is the accumulator type, the multiplicands are the fractional types. The result may overflow.

The available versions of the MLIB_MnacRnd function are shown in the following table.

Table 25. Function versions

Function name	Input type			Result type	Description
	Accum.	Mult. 1	Mult. 2		
MLIB_MnacRnd_F16	frac16_t	frac16_t	frac16_t	frac16_t	The 16-bit fractional accumulator is subtracted from the fractional product (of two 16-bit fractional multiplicands) rounded to the upper 16 bits. The output is within the range <-1 ; 1).
MLIB_MnacRnd_F32lls	frac32_t	frac32_t	frac16_t	frac32_t	The 32-bit fractional accumulator is subtracted from the fractional product (of a 32-bit and a 16-bit fractional multiplicand) rounded to the upper 32 bits [16..48]. The output is within the range <-1 ; 1).
MLIB_MnacRnd_F32	frac32_t	frac32_t	frac32_t	frac32_t	The 32-bit fractional accumulator is subtracted from the fractional product (of two 32-bit fractional multiplicands) rounded to the upper 32 bits [32..63]. The output is within the range <-1 ; 1).
MLIB_MnacRnd_A32ass	acc32_t	frac16_t	frac16_t	acc32_t	The 32-bit accumulator is subtracted from the fractional product (of two 16-bit fractional multiplicands) rounded to the upper 16-bits [16..31]. The output may be out of the range <-65536 ; 65536).

2.24.2 Declaration

The available MLIB_MnacRnd functions have the following declarations:

```
frac16_t MLIB_MnacRnd_F16(frac16_t f16Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_MnacRnd_F32lls(frac32_t f32Accum, frac32_t f32Mult1, frac16_t f16Mult2)
frac32_t MLIB_MnacRnd_F32(frac32_t f32Accum, frac32_t f32Mult1, frac32_t f32Mult2)
acc32_t MLIB_MnacRnd_A32ass(acc32_t a32Accum, frac16_t f16Mult1, frac16_t f16Mult2)
```

2.24.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Accum, f32Result, f32Mult1;
static frac16_t f16Mult2;

void main(void)
{
  f32Accum = FRAC32(0.3);               /* f32Accum = 0.3 */
  f32Mult1 = FRAC32(0.4);               /* f32Mult1 = 0.4 */
  f16Mult2 = FRAC16(-0.2);              /* f16Mult2 = -0.2 */

  /* f32Result = round(f32Mult1 * f16Mult2 - f32Accum) */
  f32Result = MLIB_MnacRnd_F32lls(f32Accum, f32Mult1, f16Mult2);
}
```

2.25 MLIB_MnacRndSat

The MLIB_MnacRndSat functions return the rounded product of two multiplicands minus the input accumulator. The round method is the round to nearest. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_MnacRndSat}(a, b, c) = \begin{cases} 1, & \mathrm{round}(b \cdot c) - a > 1 \\ -1, & \mathrm{round}(b \cdot c) - a < -1 \\ \mathrm{round}(b \cdot c) - a, & \text{else} \end{cases}$$

Figure 50. Algorithm formula

2.25.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_MnacRndSat function are shown in the following table.

Table 26. Function versions

Function name	Input type			Result type	Description
	Accum.	Mult. 1	Mult. 2		
MLIB_MnacRndSat_F16	frac16_t	frac16_t	frac16_t	frac16_t	The 16-bit fractional accumulator is subtracted from the fractional product (of two 16-bit fractional multiplicands) rounded to the upper 16 bits. The output is within the range <-1 ; 1).
MLIB_MnacRndSat_F32lls	frac32_t	frac32_t	frac16_t	frac32_t	The 32-bit fractional accumulator is subtracted from the fractional product (of a 32-bit and a 16-bit fractional multiplicand) rounded to the upper 32 bits [16..48]. The output is within the range <-1 ; 1).
MLIB_MnacRndSat_F32	frac32_t	frac32_t	frac32_t	frac32_t	The 32-bit fractional accumulator is subtracted from the fractional product (of two 32-bit fractional multiplicands) rounded to the upper 32 bits [32..63]. The output is within the range <-1 ; 1).

2.25.2 Declaration

The available MLIB_MnacRndSat functions have the following declarations:

```
frac16_t MLIB_MnacRnd_F16(frac16_t f16Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_MnacRnd_F32lls(frac32_t f32Accum, frac32_t f32Mult1, frac16_t f16Mult2)
frac32_t MLIB_MnacRnd_F32(frac32_t f32Accum, frac32_t f32Mult1, frac32_t f32Mult2)
```

2.25.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Accum, f32Result, f32Mult1;
```

```
static frac16_t f16Mult2;

void main(void)
{
  f32Accum = FRAC32(0.3);              /* f32Accum = 0.3 */
  f32Mult1 = FRAC32(0.4);              /* f32Mult1 = 0.4 */
  f16Mult2 = FRAC16(-0.2);             /* f16Mult2 = -0.2 */

  /* f32Result = round(f32Mult1 * f16Mult2 - f32Accum) */
  f32Result = MLIB_MnacRndSat_F32lls(f32Accum, f32Mult1, f16Mult2);
}
```

2.26 MLIB_Msu

The MLIB_Msu functions return the fractional product of two multiplicands subtracted from the input accumulator. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_Msu}(a, b, c) = a - b \bullet c$$

Figure 51. Algorithm formula

2.26.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The result may overflow.
- Accumulator output with mixed inputs - the output is the accumulator type, where the result can be out of the range <-1 ; 1). The accumulator is the accumulator type, the multiplicands are the fractional types. The result may overflow.
- Floating-point output - the output is a floating-point number; the result is within the full range.

The available versions of the MLIB_Msu function are shown in the following table.

Table 27. Function versions

Function name	Input type			Result type	Description
	Accum.	Mult. 1	Mult. 2		
MLIB_Msu_F16	frac16_t	frac16_t	frac16_t	frac16_t	The upper 16-bit portion [16..31] of the fractional product (of two 16-bit fractional multiplicands) is subtracted from a 16-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_Msu_F32lss	frac32_t	frac16_t	frac16_t	frac32_t	The 32-bit fractional product (of two 16-bit fractional multiplicands) is subracted from a 32-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_Msu_F32	frac32_t	frac32_t	frac32_t	frac32_t	The upper 32-bit portion [32..63] of the fractional product (of two 32-bit fractional multiplicands) is subtracted from a 32-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_Msu_A32ass	acc32_t	frac16_t	frac16_t	acc32_t	The upper 16-bit portion [16..31] of the fractional product (of two 16-bit fractional multiplicands) is subtracted from

Table continues on the next page...

Table 27. Function versions (continued)

Function name	Input type			Result type	Description
	Accum.	Mult. 1	Mult. 2		
					a 32-bit accumulator. The output may be out of the range <-65536 ; 65536).
MLIB_Msu_FLT	float_t	float_t	float_t	float_t	The product (of two 32-bit single-point floating-point multiplicands) is subtracted from a single-point floating-point accumulator. The output is within the full range.

2.26.2 Declaration

The available MLIB_Msu functions have the following declarations:

```
frac16_t MLIB_Msu_F16(frac16_t f16Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_Msu_F32lss(frac32_t f32Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_Msu_F32(frac32_t f32Accum, frac32_t f32Mult1, frac32_t f32Mult2)
acc32_t MLIB_Msu_A32ass(acc32_t a32Accum, frac16_t f16Mult1, frac16_t f16Mult2)
float_t MLIB_Msu_FLT(float_t fltAccum, float_t fltMult1, float_t fltMult2)
```

2.26.3 Function use

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

Fixed-point version:

```
#include "mlib.h"

static acc32_t  a32Accum, a32Result;
static frac16_t f16Mult1, f16Mult2;

void main(void)
{
  a32Accum = ACC32(2.3);                /* a32Accum = 2.3 */
  f16Mult1 = FRAC16(0.1);               /* f16Mult1 = 0.1 */
  f16Mult2 = FRAC16(-0.2);              /* f16Mult2 = -0.2 */

  /* a32Result = a32Accum - f16Mult1 * f16Mult2 */
  a32Result = MLIB_Msu_A32ass(a32Accum, f16Mult1, f16Mult2);
}
```

Floating-point version:

```
#include "mlib.h"

static float_t  fltAccum, fltResult;
static float_t fltMult1, fltMult2;

void main(void)
{
  fltAccum = 2.3F;                  /* fltAccum = 2.3 */
  fltMult1 = 0.1F;                 /* fltMult1 = 0.1 */
  fltMult2 = -0.2F;                /* fltMult2 = -0.2 */
```

```
  /* fltResult = fltAccum - fltMult1 * fltMult2 */
  fltResult = MLIB_Msu_FLT(fltAccum, fltMult1, fltMult2);
}
```

2.27 MLIB_MsuSat

The MLIB_MsuSat functions return the fractional product of two multiplicands subtracted from the input accumulator. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_MsuSat}(a, b, c) = \begin{cases} 1, & a - b \cdot c > 1 \\ -1, & a - b \cdot c < -1 \\ a - b \cdot c, & \text{else} \end{cases}$$

Figure 52. Algorithm formula

2.27.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_MsuSat function are shown in the following table.

Table 28. Function versions

Function name	Input type			Result type	Description
	Accum.	Mult. 1	Mult. 2		
MLIB_MsuSat_F16	frac16_t	frac16_t	frac16_t	frac16_t	The upper 16-bit portion [16..31] of the fractional product (of two 16-bit fractional multiplicands) is subtracted from a 16-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_MsuSat_F32lss	frac32_t	frac16_t	frac16_t	frac32_t	The 32-bit fractional product (of two 16-bit fractional multiplicands) is subtracted from a 32-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_MsuSat_F32	frac32_t	frac32_t	frac32_t	frac32_t	The upper 32-bit portion [32..63] of the fractional product (of two 32-bit fractional multiplicands) is subracted from a 32-bit fractional accumulator. The output is within the range <-1 ; 1).

2.27.2 Declaration

The available MLIB_MsuSat functions have the following declarations:

```
frac16_t MLIB_MsuSat_F16(frac16_t f16Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_MsuSat_F32lss(frac32_t f32Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_MsuSat_F32(frac32_t f32Accum, frac32_t f32Mult1, frac32_t f32Mult2)
```

2.27.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Accum, f32Mult1, f32Mult2, f32Result;

void main(void)
{
  f32Accum = FRAC32(0.9);                 /* f32Accum = 0.9 */
  f32Mult1 = FRAC32(-1.0);                /* f32Mult1 = -1.0 */
  f32Mult2 = FRAC32(0.2);                 /* f32Mult2 = 0.2 */

  /* f32Result = sat(f32Accum - f32Mult1 * f32Mult2) */
  f32Result = MLIB_MsuSat_F32(f32Accum, f32Mult1, f32Mult2);
}
```

2.28 MLIB_MsuRnd

The MLIB_MsuRnd functions return the rounded fractional product of two multiplicands subtracted from the input accumulator. The round method is the round to nearest. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_MsuRnd}(a, b, c) = a - \mathrm{round}(b \cdot c)$$

Figure 53. Algorithm formula

2.28.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The result may overflow.
- Accumulator output with mixed inputs - the output is the accumulator type, where the result can be out of the range <-1 ; 1). The accumulator is the accumulator type, the multiplicands are the fractional types. The result may overflow.

The available versions of the MLIB_MsuRnd function are shown in the following table.

Table 29. Function versions

Function name	Input type			Result type	Description
	Accum.	Mult. 1	Mult. 2		
MLIB_MsuRnd_F16	frac16_t	frac16_t	frac16_t	frac16_t	The fractional product (of two 16-bit fractional multiplicands), rounded to the upper 16 bits, is subtracted from a 16-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_MsuRnd_F32lls	frac32_t	frac32_t	frac16_t	frac32_t	The fractional product (of a 32-bit and 16-bit fractional multiplicands), rounded to the upper 32 bits [16..48], is subtracted from a 32-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_MsuRnd_F32	frac32_t	frac32_t	frac32_t	frac32_t	The fractional product (of two 32-bit fractional multiplicands), rounded to the upper 32 bits [32..63],

Table continues on the next page...

Table 29. Function versions (continued)

Function name	Input type			Result type	Description
	Accum.	Mult. 1	Mult. 2		
					is subtracted from a 32-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_MsuRnd_A32ass	acc32_t	frac16_t	frac16_t	acc32_t	The fractional product (of two 16-bit fractional multiplicands), rounded to the upper 16 bits [16..31], is subtracted from a 32-bit accumulator. The output may be out of the range <-65536 ; 65536).

2.28.2 Declaration

The available MLIB_MsuRnd functions have the following declarations:

```
frac16_t MLIB_MsuRnd_F16(frac16_t f16Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_MsuRnd_F32lls(frac32_t f32Accum, frac32_t f32Mult1, frac16_t f16Mult2)
frac32_t MLIB_MsuRnd_F32(frac32_t f32Accum, frac32_t f32Mult1, frac32_t f32Mult2)
acc32_t MLIB_MsuRnd_A32ass(acc32_t a32Accum, frac16_t f16Mult1, frac16_t f16Mult2)
```

2.28.3 Function use

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

```
#include "mlib.h"

static frac16_t f16Accum, f16Mult1, f16Mult2, f16Result;

void main(void)
{
  f16Accum = FRAC16(0.3);               /* f16Accum = 0.3 */
  f16Mult1 = FRAC16(0.1);               /* f16Mult1 = 0.1 */
  f16Mult2 = FRAC16(-0.2);              /* f16Mult2 = -0.2 */

  /* f16Result = round(f16Accum - f16Mult1 * f16Mult2) */
   f16Result = MLIB_MsuRnd_F16(f16Accum, f16Mult1, f16Mult2);
}
```

2.29 MLIB_MsuRndSat

The MLIB_MsuRndSat functions return the rounded fractional product of two multiplicands subtracted from the input accumulator. The round method is the round to nearest. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_MsuRndSat}(a, b, c) = \begin{cases} 1, & a - \mathrm{round}(b \cdot c) > 1 \\ -1, & a - \mathrm{round}(b \cdot c) < -1 \\ a - \mathrm{round}(b \cdot c), & \text{else} \end{cases}$$

Figure 54. Algorithm formula

2.29.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_MsuRndSat function are shown in the following table.

Table 30. Function versions

Function name	Input type			Result type	Description
	Accum.	Mult. 1	Mult. 2		
MLIB_MsuRndSat_F16	frac16_t	frac16_t	frac16_t	frac16_t	The fractional product (of two 16-bit fractional multiplicands), rounded to the upper 16 bits, is subtracted from a 16-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_MsuRndSat_F32lls	frac32_t	frac32_t	frac16_t	frac32_t	The fractional product (of a 32-bit and 16-bit fractional multiplicands), rounded to the upper 32 bits [16..48], is subtracted from a 32-bit fractional accumulator. The output is within the range <-1 ; 1).
MLIB_MsuRndSat_F32	frac32_t	frac32_t	frac32_t	frac32_t	The fractional product (of two 32-bit fractional multiplicands), rounded to the upper 32 bits [32..63], is subtracted from a 32-bit fractional accumulator. The output is within the range <-1 ; 1).

2.29.2 Declaration

The available MLIB_MsuRndSat functions have the following declarations:

```
frac16_t MLIB_MsuRndSat_F16(frac16_t f16Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_MsuRndSat_F32lls(frac32_t f32Accum, frac32_t f32Mult1, frac16_t f16Mult2)
frac32_t MLIB_MsuRndSat_F32(frac32_t f32Accum, frac32_t f32Mult1, frac32_t f32Mult2)
```

2.29.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Accum, f32Mult1, f32Mult2, f32Result;

void main(void)
{
  f32Accum = FRAC32(0.3);               /* f32Accum = 0.3 */
  f32Mult1 = FRAC32(0.1);               /* f32Mult1 = 0.1 */
  f32Mult2 = FRAC32(-0.2);              /* f32Mult2 = -0.2 */

  /* f32Result = sat(round(f32Accum - f32Mult1 * f32Mult2)) */
  f32Result = MLIB_MsuRndSat_F32(f32Accum, f32Mult1, f32Mult2);
}
```

2.30 MLIB_Msu4

The MLIB_Msu4 functions return the subtraction of the products of two multiplicands. The function does not saturate the output. See the following equation:

$$\text{MLIB\_Msu4}(a, b, c, d) = a \cdot b - c \cdot d$$

Figure 55. Algorithm formula

2.30.1 Available versions

The function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The result may overflow.
- Floating-point output - the output is a floating-point number; the result is within the full range.

The available versions of the MLIB_Msu4 function are shown in the following table.

Table 31. Function versions

Function name	Input type				Result type	Description
	Minuend product		Subtrahend product			
	Mult. 1	Mult. 2	Mult. 1	Mult. 2		
MLIB_Msu4_F32ssss	frac16_t	frac16_t	frac16_t	frac16_t	frac32_t	Subtraction of two 32-bit fractional products (of two 16-bit fractional multiplicands). The output is within the range <-1 ; 1).
MLIB_Msu4_FLT	float_t	float_t	float_t	float_t	float_t	Subtraction of two 32-bit single-point floating-point products (of two 32-bit single-point floating-point multiplicands). The output is within the full range.

2.30.2 Declaration

The available MLIB_Msu4 functions have the following declarations:

```
frac32_t MLIB_Msu4_F32ssss(frac16_t f16MinMult1, frac16_t f16MinMult2, frac16_t f16SubMult1,
frac16_t f16SubMult2)

float_t MLIB_Msu4_FLT(float_t fltMinMult1, float_t fltMinMult2, float_t fltSubMult1,
float_t fltSubMult2)
```

2.30.3 Function use

The use of the MLIB_Msu4 function is shown in the following examples:

Fixed-point version:

```
#include "mlib.h"

static frac32_t f32Result;
static frac16_t f16MinMult1, f16MinMult2, f16SubMult1, f16SubMult2;

void main(void)
```

```
{
    f16MinMult1 = FRAC16(0.2);          /* f16MinMult1 = 0.2 */
    f16MinMult2 = FRAC16(-0.7);         /* f16MinMult2 = -0.7 */
    f16SubMult1 = FRAC16(0.3);          /* f16SubMult1 = 0.3 */
    f16SubMult2 = FRAC16(-0.25);        /* f16SubMult2 = -0.25 */

    /* f32Result = f16MinMult1 * f16MinMult2 - f16SubMult1 * f16SubMult2 */
    f32Result = MLIB_Msu4_F32ssss(f16MinMult1, f16MinMult2, f16SubMult1, f16SubMult2);
}
```

Floating-point version:

```
#include "mlib.h"

static float_t fltResult;
static float_t fltMinMult1, fltMinMult2, fltSubMult1, fltSubMult2;

void main(void)
{
    fltMinMult1 = 0.2F;          /* fltMinMult1 = 0.2 */
    fltMinMult2 = -0.7F;         /* fltMinMult2 = -0.7 */
    fltSubMult1 = 0.3F;          /* fltSubMult1 = 0.3 */
    fltSubMult2 = -0.25F;        /* fltSubMult2 = -0.25 */

    /* fltResult = fltMinMult1 * fltMinMult2 - fltSubMult1 * fltSubMult2 */
    fltResult = MLIB_Msu4_FLT(fltMinMult1, fltMinMult2, fltSubMult1, fltSubMult2);
}
```

2.31 MLIB_Msu4Sat

The MLIB_Msu4Sat functions return the subtraction of the products of two multiplicands. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_Msu4Sat}(a, b, c, d) = \begin{cases} 1, & a \cdot b - c \cdot d > 1 \\ -1, & a \cdot b - c \cdot d < -1 \\ a \cdot b - c \cdot d, & \text{else} \end{cases}$$

Figure 56. Algorithm formula

2.31.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_Msu4Sat function are shown in the following table.

Table 32. Function versions

Function name	Input type				Result type	Description
	Minuend product		Subtrahend product			
	Mult. 1	Mult. 2	Mult. 1	Mult. 2		
MLIB_Msu4Sat_F32ssss	frac16_t	frac16_t	frac16_t	frac16_t	frac32_t	Subtraction of two 32-bit fractional products (of two 16-bit fractional multiplicands). The output is within the range <-1 ; 1).

2.31.2 Declaration

The available MLIB_Msu4Sat functions have the following declarations:

```
frac32_t MLIB_Msu4Sat_F32ssss(frac16_t f16MinMult1, frac16_t f16MinMult2, frac16_t f16SubMult1,
frac16_t f16SubMult2)
```

2.31.3 Function use

The use of the MLIB_Msu4Sat function is shown in the following example:

```
#include "mlib.h"

static frac32_t f32Result;
static frac16_t f16MinMult1, f16MinMult2, f16SubMult1, f16SubMult2;

void main(void)
{
   f16MinMult1 = FRAC16(0.8);           /* f16MinMult1 = 0.8 */
   f16MinMult2 = FRAC16(-0.9);          /* f16MinMult2 = -0.9 */
   f16SubMult1 = FRAC16(0.7);           /* f16SubMult1 = 0.7 */
   f16SubMult2 = FRAC16(0.9);           /* f16SubMult2 = 0.9 */

   /* f32Result = sat(f16MinMult1 * f16MinMult2 - f16SubMult1 * f16SubMult2) */
   f32Result = MLIB_Msu4Sat_F32ssss(f16MinMult1, f16MinMult2, f16SubMult1,
f16SubMult2);
}
```

2.32 MLIB_Msu4Rnd

The MLIB_Msu4Rnd functions return the rounded subtraction of two products of two pairs of multiplicands. The round method is the round to nearest. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_Msu4Rnd}(a, b, c, d) = \mathrm{round}(a \cdot b - c \cdot d)$$

Figure 57. Algorithm formula

2.32.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_Msu4Rnd function are shown in the following table.

Table 33. Function versions

Function name	Input type				Result type	Description
	Minuend product		Subtrahend product			
	Mult. 1	Mult. 2	Mult. 1	Mult. 2		
MLIB_Msu4Rnd_F16	frac16_t	frac16_t	frac16_t	frac16_t	frac16_t	Subtraction of two 16-bit fractional products (of two 16-bit fractional multiplicands), rounded to the upper 16 bits. The output is within the range <-1 ; 1).
MLIB_Msu4Rnd_F32	frac32_t	frac32_t	frac32_t	frac32_t	frac32_t	Subtraction of two 32-bit fractional products (of two 32-bit fractional multiplicands), rounded to the upper 32 bits. The output is within the range <-1 ; 1).

2.32.2 Declaration

The available MLIB_Msu4Rnd functions have the following declarations:

```
frac16_t MLIB_Msu4Rnd_F16(frac16_t f16MinMult1, frac16_t f16MinMult2, frac16_t f16SubMult1,
frac16_t f16SubMult2)

frac32_t MLIB_Msu4Rnd_F32(frac32_t f32MinMult1, frac32_t f32MinMult2, frac32_t f32SubMult1,
frac32_t f32SubMult2)
```

2.32.3 Function use

The use of the MLIB_Msu4Rnd function is shown in the following example:

```
#include "mlib.h"

static frac16_t f16Result, f16MinMult1, f16MinMult2, f16SubMult1, f16SubMult2;

void main(void)
{
   f16MinMult1 = FRAC16(0.256);         /* f16MinMult1 = 0.256 */
   f16MinMult2 = FRAC16(-0.724);        /* f16MinMult2 = -0.724*/
   f16SubMult1 = FRAC16(0.365);         /* f16SubMult1 = 0.365 */
   f16SubMult2 = FRAC16(-0.25);         /* f16SubMult2 = -0.25 */

   /* f32Result = round(f16MinMult1 * f16MinMult2 - f16SubMult1 * f16SubMult2) */
```

```
    f16Result = MLIB_Msu4Rnd_F16(f16MinMult1, f16MinMult2, f16SubMult1, f16SubMult2);
}
```

2.33 MLIB_Msu4RndSat

The MLIB_Msu4RndSat functions return the rounded subtraction of two products of two pairs of multiplicands. The round method is the round to nearest. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_Msu4RndSat}(a, b, c, d) = \begin{cases} 1, & \mathrm{round}(a \cdot b - c \cdot d) > 1 \\ -1, & \mathrm{round}(a \cdot b - c \cdot d) < -1 \\ \mathrm{round}(a \cdot b - c \cdot d), & \text{else} \end{cases}$$

Figure 58. Algorithm formula

2.33.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_Msu4RndSat function are shown in the following table.

Table 34. Function versions

Function name	Input type				Result type	Description
	Minuend product		Subtrahend product			
	Mult. 1	Mult. 2	Mult. 1	Mult. 2		
MLIB_Msu4RndSat_F16	frac16_t	frac16_t	frac16_t	frac16_t	frac16_t	Subtraction of two 16-bit fractional products (of two 16-bit fractional multiplicands), rounded to the upper 16 bits. The output is within the range <-1 ; 1).
MLIB_Msu4RndSat_F32	frac32_t	frac32_t	frac32_t	frac32_t	frac32_t	Subtraction of two 32-bit fractional products (of two 32-bit fractional multiplicands), rounded to the upper 32 bits. The output is within the range <-1 ; 1).

2.33.2 Declaration

The available MLIB_Msu4RndSat functions have the following declarations:

```
frac16_t MLIB_Msu4RndSat_F16(frac16_t f16MinMult1, frac16_t f16MinMult2, frac16_t f16SubMult1,
frac16_t f16SubMult2)

frac32_t MLIB_Msu4RndSat_F32(frac32_t f32MinMult1, frac32_t f32MinMult2, frac32_t f32SubMult1,
frac32_t f32SubMult2)
```

2.33.3 Function use

The use of the MLIB_Msu4RndSat function is shown in the following example:

```
#include "mlib.h"

static frac16_t f16Result, f16MinMult1, f16MinMult2, f16SubMult1, f16SubMult2;

void main(void)
{
    f16MinMult1 = FRAC16(0.8);          /* f16MinMult1 = 0.8 */
    f16MinMult2 = FRAC16(-0.9);         /* f16MinMult2 = -0.9 */
    f16SubMult1 = FRAC16(0.7);          /* f16SubMult1 = 0.7 */
    f16SubMult2 = FRAC16(0.9);          /* f16SubMult2 = 0.9 */

    /* f16Result = sat(round(f16MinMult1 * f16MinMult2 - f16SubMult1 * f16SubMult2)) */
    f16Result = MLIB_Msu4RndSat_F16(f16MinMult1, f16MinMult2, f16SubMult1,
f16SubMult2);
}
```

2.34 MLIB_Mul

The MLIB_Mul functions return the product of two multiplicands. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_Mul}(a, b) = a \bullet b$$

Figure 59. Algorithm formula

2.34.1 Available versions

This function is available in the following versions:

- Fractional output with fractional inputs - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The inputs are the fractional values only. The result may overflow.
- Fractional output with mixed inputs - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The inputs are the accumulator and fractional values. The result may overflow.
- Accumulator output - the output is the accumulator type where the result can be out of the range <-1 ; 1). The result may overflow.
- Floating-point output - the output is a floating-point number; the result is within the full range.

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

Table 35. Function versions

Function name	Input type		Result type	Description
	Mult. 1	Mult. 2		
MLIB_Mul_F16	frac16_t	frac16_t	frac16_t	Product of two 16-bit fractional multiplicands; the output are the upper 16 bits of the results [16..31]. The output is within the range <-1 ; 1).
MLIB_Mul_F16as	acc32_t	frac16_t	frac16_t	Product of a 32-bit accumulator and a 16-bit fractional multiplicand; the output is a 16-bit fractional portion, which has the upper 16 bits of the

Table continues on the next page...

Table 35. Function versions (continued)

Function name	Input type		Result type	Description
	Mult. 1	Mult. 2		
				fractional value of the result [16..31]. The output is within the range <-1 ; 1).
MLIB_Mul_F32ss	frac16_t	frac16_t	frac32_t	Product of two 16-bit fractional multiplicands; the result is a 32-bit fractional value. The output is within the range <-1 ; 1).
MLIB_Mul_F32	frac32_t	frac32_t	frac32_t	Product of two 32-bit fractional multiplicands; the output are the upper 32 bits of the results [16..31]. The output is within the range <-1 ; 1).
MLIB_Mul_A32	acc32_t	acc32_t	acc32_t	Product of two 32-bit accumulator multiplicands; the output is a 32-bit accumulator, which has the upper mid bits of the result [16..47]. The output is within the range <-65536.0 ; 65536.0).
MLIB_Mul_FLT	float_t	float_t	float_t	Product of two 32-bit single precision floating-point multiplicands. The output is within the full range.

2.34.2 Declaration

The available MLIB_Mul functions have the following declarations:

```
frac16_t MLIB_Mul_F16(frac16_t f16Mult1, frac16_t f16Mult2)
frac16_t MLIB_Mul_F16as(acc32_t a32Accum, frac16_t f16Mult)
frac32_t MLIB_Mul_F32ss(frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_Mul_F32(frac32_t f32Mult1, frac32_t f32Mult2)
acc32_t MLIB_Mul_A32(acc32_t a32Mult1, acc32_t a32Mult1)
float_t MLIB_Mul_FLT(float_t fltMult1, float_t fltMult2)
```

2.34.3 Function use

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

Fixed-point version:

```
#include "mlib.h"

static frac32_t f32Result;
static frac16_t f16Mult1, f16Mult2;

void main(void)
{
  f16Mult1 = FRAC16(0.4);        /* f16Mult1 = 0.4 */
  f16Mult2 = FRAC16(-0.2);       /* f16Mult2 = -0.2 */

  /* f32Result = f16Mult1 * f16Mult2 */
  f32Result = MLIB_Mul_F32ss(f16Mult1, f16Mult2);
}
```

Floating-point version:

```
#include "mlib.h"
```

```
static float_t fltResult;
static float_t fltMult1, fltMult2;

void main(void)
{
  fltMult1 = 0.4F;        /* fltMult1 = 0.4 */
  fltMult2 = -0.2F;       /* fltMult2 = -0.2 */

  /* fltResult = fltMult1 * fltMult2 */
  fltResult = MLIB_Mul_FLT(fltMult1, fltMult2);
}
```

2.35 MLIB_MulSat

The MLIB_MulSat functions return the product of two multiplicands. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_MulSat}(a, b) = \begin{cases} max, & a \bullet b > max \\ min, & a \bullet b < min \\ a \bullet b, & \text{else} \end{cases}$$

Figure 60. Algorithm formula

2.35.1 Available versions

This function is available in the following versions:

- Fractional output with fractional inputs - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The inputs are the fractional values only. The result may saturate.
- Fractional output with mixed inputs - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The inputs are the accumulator and fractional values. The result may saturate.
- Accumulator output - the output is the accumulator type where the result can be out of the range <-1;1). The result may overflow.

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

Table 36. Function versions

Function name	Input type		Result type	Description
	Mult. 1	Mult. 2		
MLIB_MulSat_F16	frac16_t	frac16_t	frac16_t	Product of two 16-bit fractional multiplicands; the output is the upper 16 bits of the results [16..31]. The output is within the range <-1 ; 1).
MLIB_MulSat_F16as	acc32_t	frac16_t	frac16_t	Product of a 32-bit accumulator and a 16-bit fractional multiplicand; the output is a 16-bit fractional value, which has the upper 16 bits of the fractional portion of the result [16..31]. The output is within the range <-1 ; 1).
MLIB_MulSat_F32ss	frac16_t	frac16_t	frac32_t	Product of two 16-bit fractional multiplicands; the result is a 32-bit fractional value. The output is within the range <-1 ; 1).
MLIB_MulSat_F32	frac32_t	frac32_t	frac32_t	Product of two 32-bit fractional multiplicands; the output are the upper 32 bits of the results [16..31]. The output is within the range <-1 ; 1).

Table continues on the next page...

Table 36. Function versions (continued)

Function name	Input type		Result type	Description
	Mult. 1	Mult. 2		
MLIB_MulSat_A32	acc32_t	acc32_t	acc32_t	Product of two 32-bit accumulator multiplicands; the output is a 32-bit accumulator, which has the mid bits of the result [16..47]. The output is within the range <-65536.0 ; 65536.0).

2.35.2 Declaration

The available MLIB_MulSat functions have the following declarations:

```
frac16_t MLIB_MulSat_F16(frac16_t f16Mult1, frac16_t f16Mult2)
frac16_t MLIB_MulSat_F16as(acc32_t a32Accum, frac16_t f16Mult)
frac32_t MLIB_MulSat_F32ss(frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_MulSat_F32(frac32_t f32Mult1, frac32_t f32Mult2)
acc32_t MLIB_MulSat_A32(acc32_t a32Mult1, acc32_t a32Mult1)
```

2.35.3 Function use

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

```
#include "mlib.h"

static acc32_t a32Accum;
static frac16_t f16Mult, f16Result;

void main(void)
{
   a32Accum = ACC32(-5.5);        /* a32Accum = -5.5 */
   f16Mult = FRAC16(0.3);         /* f16Mult = 0.3 */

   /* f16Result = sat(a32Accum * f16Mult) */
   f16Result = MLIB_MulSat_F16as(a32Accum, f16Mult);
}
```

2.36 MLIB_MulNeg

The MLIB_MulNeg functions return the negative product of two multiplicands. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_MulNeg}(a, b) = -a \cdot b$$

Figure 61. Algorithm formula

2.36.1 Available versions

This function is available in the following versions:

- Fractional output with fractional inputs - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The inputs are the fractional values only.
- Fractional output with mixed inputs - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The inputs are the accumulator and fractional values. The result may overflow.

- Accumulator output - the output is the accumulator type where the result can be out of the range <-1;1). The result may overflow.
- Floating-point output - the output is a floating-point number; the result is within the full range.

The available versions of the MLIB_MulNeg function are shown in the following table.

Table 37. Function versions

Function name	Input type		Result type	Description
	Mult. 1	Mult. 2		
MLIB_MuNegl_F16	frac16_t	frac16_t	frac16_t	Negative product of two 16-bit fractional multiplicands; the output are the upper 16 bits of the results [16..31]. The output is within the range <-1 ; 1).
MLIB_MulNeg_F16as	acc32_t	frac16_t	frac16_t	Negative product of a 32-bit accumulator and a 16-bit fractional multiplicand; the output is a 16-bit fractional value, which has the upper 16 bits of the fractional portion of the result [16..31]. The output is within the range <-1 ; 1).
MLIB_MulNeg_F32ss	frac16_t	frac16_t	frac32_t	Negative product of two 16-bit fractional multiplicands; the result is a 32-bit fractional value. The output is within the range <-1 ; 1).
MLIB_MulNeg_F32	frac32_t	frac32_t	frac32_t	Negative product of two 32-bit fractional multiplicands; the output are the upper 32 bits of the results [16..31]. The output is within the range <-1 ; 1).
MLIB_MulNeg_A32	acc32_t	acc32_t	acc32_t	Product of two 32-bit accumulator multiplicands; the output is a 32-bit accumulator, which has the mid bits of the result [16..47]. The output is within the range <-65536.0 ; 65536.0).
MLIB_MulNeg_FLT	float_t	float_t	float_t	Negative product of two 32-bit single precision floating-point multiplicands. The output is within the full range.

2.36.2 Declaration

The available MLIB_MulNeg functions have the following declarations:

```
frac16_t MLIB_MulNeg_F16(frac16_t f16Mult1, frac16_t f16Mult2)
frac16_t MLIB_MulNeg_F16as(acc32_t a32Accum, frac16_t f16Mult)
frac32_t MLIB_MulNeg_F32ss(frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_MulNeg_F32(frac32_t f32Mult1, frac32_t f32Mult2)
acc32_t MLIB_MulNeg_A32(acc32_t a32Mult1, acc32_t a32Mult1)
float_t MLIB_MulNeg_FLT(float_t fltMult1, float_t fltMult2)
```

2.36.3 Function use

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

Fixed-point version:

```
#include "mlib.h"

static frac32_t f32Result;
static frac16_t f16Mult1, f16Mult2;

void main(void)
{
```

```
  f16Mult1 = FRAC16(0.5);        /* f16Mult1 = 0.5 */
  f16Mult2 = FRAC16(-0.3);       /* f16Mult2 = -0.3 */

  /* f32Result = f16Mult1 * (-f16Mult2) */
  f32Result = MLIB_MulNeg_F32ss(f16Mult1, f16Mult2);
}
```

Floating-point version:

```
#include "mlib.h"

static float_t fltResult;
static float_t fltMult1, fltMult2;

void main(void)
{
  fltMult1 = 0.5F;        /* fltMult1 = 0.5 */
  fltMult2 = -0.3F;       /* fltMult2 = -0.3 */

  /* fltResult = fltMult1 * (-fltMult2) */
  fltResult = MLIB_MulNeg_FLT(fltMult1, fltMult2);
}
```

2.37 MLIB_MulNegSat

The MLIB_MulNegSat functions return the negative product of two multiplicands. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_MulNegSat}(a, b) = \begin{cases} max, & -a \bullet b > max \\ min, & -a \bullet b < min \\ -a \bullet b, & \text{else} \end{cases}$$

Figure 62. Algorithm formula

2.37.1 Available versions

This function is available in the following versions:

- Fractional output with mixed inputs - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The inputs are the accumulator and fractional values. The result may saturate.
- Accumulator output - the output is the accumulator type where the result can be out of the range <-1 ; 1). The result may overflow.

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

Table 38. Function versions

Function name	Input type		Result type	Description
	Mult. 1	Mult. 2		
MLIB_MulNegSat_F16as	acc32_t	frac16_t	frac16_t	Negative product of a 32-bit accumulator and a 16-bit fractional multiplicand; the output is a 16-bit fractional value, which has the

Table continues on the next page...

Table 38. Function versions (continued)

Function name	Input type		Result type	Description
	Mult. 1	Mult. 2		
				upper 16 bits of the fractional portion of the result [16..31]. The output is within the range <-1 ; 1).
MLIB_MulNegSat_A32	acc32_t	acc32_t	acc32_t	Negative product of two 32-bit accumulator multiplicands; the output is a 32-bit accumulator, which has the middle bits of the result [16..47]. The output is within the range <-65536.0 ; 65536.0).

2.37.2 Declaration

The available MLIB_MulNegSat functions have the following declarations:

```
frac16_t MLIB_MulNegSat_F16as(acc32_t a32Accum, frac16_t f16Mult)
acc32_t MLIB_MulNegSat_A32(acc32_t a32Mult1, acc32_t a32Mult2)
```

2.37.3 Function use

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

```
#include "mlib.h"

static acc32_t a32M1, a32M2, a32Result;

void main(void)
{
   a32M1 = ACC32(1.5);       /* a32M1 = 1.5 */
   a32M2 = ACC32(4.1);        /* a32M2 = 4.1 */

   /* f16Result = sat(-a32M1 * f32M2) */
   a32Result = MLIB_MulNegSat_A32(a32M1, a32M2);
}
```

2.38 MLIB_MulRnd

The MLIB_MulRnd functions return the rounded product of two multiplicands. The round method is the round to nearest. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_MulRnd}(a, b) = \mathrm{round}(a \cdot b)$$

Figure 63. Algorithm formula

2.38.1 Available versions

This function is available in the following versions:

- Fractional output with fractional inputs - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The inputs are the fractional values only. The result may overflow.
- Fractional output with mixed inputs - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The inputs are the accumulator and fractional values. The result may overflow.

- Accumulator output - the output is the accumulator type where the result can be out of the range <-1 ; 1). The result may overflow.

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

Table 39. Function versions

Function name	Input type		Result type	Description
	Mult. 1	Mult. 2		
MLIB_MulRnd_F16	frac16_t	frac16_t	frac16_t	Product of two 16-bit fractional multiplicands; the output is rounded to the upper 16 bits of the results [16..31]. The output is within the range <-1 ; 1).
MLIB_MulRnd_F16as	acc32_t	frac16_t	frac16_t	Product of a 32-bit accumulator and a 16-bit fractional multiplicand; the output is a 16-bit fractional value, which is rounded to the upper 16 bits of the fractional portion of the result [16..31]. The output is within the range <-1 ; 1).
MLIB_MulRnd_F32ls	frac32_t	frac16_t	frac32_t	Product of a 32-bit and a 16-bit fractional multiplicand; the output is rounded to the upper 32 bits of the fractional portion of the result [16..47]. The output is within the range <-1 ; 1).
MLIB_MulRnd_F32	frac32_t	frac32_t	frac32_t	Product of two 32-bit fractional multiplicands; the output is rounded to the upper 32 bits of the results [16..31]. The output is within the range <-1 ; 1).
MLIB_MulRnd_A32	acc32_t	acc32_t	acc32_t	Product of two 32-bit accumulator multiplicands; the output is rounded to the middle bits of the result [16..47]. The output is within the range <-65536.0 ; 65536.0).

2.38.2 Declaration

The available MLIB_MulRnd functions have the following declarations:

```
frac16_t MLIB_MulRnd_F16(frac16_t f16Mult1, frac16_t f16Mult2)
frac16_t MLIB_MulRnd_F16as(acc32_t a32Accum, frac16_t f16Mult)
frac32_t MLIB_MulRnd_F32ls(frac32_t f32Mult1, frac16_t f16Mult2)
frac32_t MLIB_MulRnd_F32(frac32_t f32Mult1, frac32_t f32Mult2)
acc32_t MLIB_MulRnd_A32(acc32_t a32Mult1, acc32_t a32Mult1)
```

2.38.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Mult1, f32Mult2, f32Result;

void main(void)
{
  f32Mult1 = FRAC32(0.5);        /* f32Mult1 = 0.5 */
  f32Mult2 = FRAC32(-0.24564);   /* f32Mult2 = -0.24564 */

  /* f32Result = round(f32Mult1 * f32Mult2) */
  f32Result = MLIB_MulRnd_F32(f32Mult1, f32Mult2);
}
```

2.39 MLIB_MulRndSat

The MLIB_MulRndSat functions return the rounded product of two multiplicands. The round method is the round to nearest. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_MulRndSat}(a, b) = \begin{cases} max, & \mathrm{round}(a \bullet b) > max \\ min, & \mathrm{round}(a \bullet b) < min \\ \mathrm{round}(a \bullet b), & \mathrm{else} \end{cases}$$

Figure 64. Algorithm formula

2.39.1 Available versions

This function is available in the following versions:

- Fractional output with fractional inputs - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The inputs are the fractional values only. The result may saturate.
- Fractional output with mixed inputs - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The inputs are the accumulator and fractional values. The result may saturate.
- Accumulator output - the output is the accumulator type where the result can be out of the range <-1 ; 1). The result may overflow.

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

Table 40. Function versions

Function name	Input type		Result type	Description
	Mult. 1	Mult. 2		
MLIB_MulRndSat_F16	frac16_t	frac16_t	frac16_t	Product of two 16-bit fractional multiplicands; the output is rounded to the upper 16 bits of the results [16..31]. The output is within the range <-1 ; 1).
MLIB_MulRndSat_F16as	acc32_t	frac16_t	frac16_t	Product of a 32-bit accumulator and a 16-bit fractional multiplicand; the output is a 16-bit fractional value, which is rounded to the upper 16 bits of the fractional portion of the result [16..31]. The output is within the range <-1 ; 1).
MLIB_MulRndSat_F32ls	frac32_t	frac16_t	frac32_t	Product of a 32-bit multiplicand and a 16-bit fractional multiplicand; the output is rounded to the upper 32 bits of the fractional portion of the result [16..47]. The output is within the range <-1 ; 1).
MLIB_MulRndSat_F32	frac32_t	frac32_t	frac32_t	Product of two 32-bit fractional multiplicands; the output is rounded to the upper 32 bits of the results [16..31]. The output is within the range <-1 ; 1).
MLIB_MulRndSat_A32	acc32_t	acc32_t	acc32_t	Product of two 32-bit accumulator multiplicands; the output is rounded to the the mid bits of the result [16..47]. The output is within the range <-65536.0 ; 65536.0).

2.39.2 Declaration

The available MLIB_MulRndSat functions have the following declarations:

```
frac16_t MLIB_MulRndSat_F16(frac16_t f16Mult1, frac16_t f16Mult2)
frac16_t MLIB_MulRndSat_F16as(acc32_t a32Accum, frac16_t f16Mult)
frac32_t MLIB_MulRndSat_F32ls(frac32_t f32Mult1, frac16_t f16Mult2)
```

```
frac32_t MLIB_MulRndSat_F32(frac32_t f32Mult1, frac32_t f32Mult2)
acc32_t MLIB_MulRndSat_A32(acc32_t a32Mult1, acc32_t a32Mult1)
```

2.39.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Mult1, f32Mult2, f32Result;

void main(void)
{
  f32Mult1 = FRAC32(-1.0);               /* f32Mult1 = -1.0 */
  f32Mult2 = FRAC32(-1.0);               /* f32Mult2 = -1.0 */

  /* f32Result = sat(round(f32Mult1 * f32Mult2))*/
  f32Result = MLIB_MulRndSat_F32(f32Mult1, f32Mult2);
}
```

2.40 MLIB_MulNegRnd

The MLIB_MulNegRnd functions return the rounded negative product of two multiplicands. The round method is the round to nearest. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_MulNegRnd}(a, b) = \mathrm{round}(-a \cdot b)$$

Figure 65. Algorithm formula

2.40.1 Available versions

This function is available in the following versions:

- Fractional output with fractional inputs - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The inputs are the fractional values only.
- Fractional output with mixed inputs - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The inputs are the accumulator and fractional values. The result may overflow.
- Accumulator output - the output is the accumulator type where the result can be out of the range <-1 ; 1). The result may overflow.

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

Table 41. Function versions

Function name	Input type		Result type	Description
	Mult. 1	Mult. 2		
MLIB_MulNegRnd_F16	frac16_t	frac16_t	frac16_t	Negative product of two 16-bit fractional multiplicands; the output is rounded to the upper 16 bits of the results [16..31]. The output is within the range <-1 ; 1).
MLIB_MulNegRnd_F16as	acc32_t	frac16_t	frac16_t	Negative product of a 32-bit accumulator and a 16-bit fractional multiplicand; the output is a 16-bit fractional value, which is

Table continues on the next page...

Table 41. Function versions (continued)

Function name	Input type		Result type	Description
	Mult. 1	Mult. 2		
				rounded to the upper 16 bits of the fractional portion of the result [16..31]. The output is within the range <-1 ; 1).
MLIB_MulNegRnd_F32ls	frac32_t	frac16_t	frac32_t	Negative product of a 32-bit fractional multiplicand and a 16-bit fractional multiplicand; the output is rounded to the upper 32 bits of the fractional portion of the result [16..47]. The output is within the range <-1 ; 1).
MLIB_MulNegRnd_F32	frac32_t	frac32_t	frac32_t	Negative product of two 32-bit fractional multiplicands; the output is rounded to the upper 32 bits of the results [16..31]. The output is within the range <-1 ; 1).
MLIB_MulNegRnd_A32	acc32_t	acc32_t	acc32_t	Product of two 32-bit accumulator multiplicands; the output is rounded to the the middle bits of the result [16..47]. The output is within the range <-65536.0 ; 65536.0).

2.40.2 Declaration

The available MLIB_MulNegRnd functions have the following declarations:

```
frac16_t MLIB_MulNegRnd_F16(frac16_t f16Mult1, frac16_t f16Mult2)
frac16_t MLIB_MulNegRnd_F16as(acc32_t a32Accum, frac16_t f16Mult)
frac32_t MLIB_MulNegRnd_F32ls(frac32_t f32Mult1, frac16_t f16Mult2)
frac32_t MLIB_MulNegRnd_F32(frac32_t f32Mult1, frac32_t f32Mult2)
acc32_t MLIB_MulNegRnd_A32(acc32_t a32Mult1, acc32_t a32Mult1)
```

2.40.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Mult1, f32Mult2, f32Result;

void main(void)
{
  f32Mult1 = FRAC32(0.3);       /* f32Mult1 = 0.3 */
  f32Mult2 = FRAC32(-0.5);      /* f32Mult2 = -0.5 */

  /* f32Result = round(f32Mult1 * (-f32Mult2)) */
  f32Result = MLIB_MulNegRnd_F32(f32Mult1, f32Mult2);
}
```

2.41 MLIB_MulNegRndSat

The MLIB_MulNegRndSat functions return the rounded negative product of two multiplicands. The round method is the round to nearest. The function saturates the output. See the following equation:

$$\text{MLIB\_MulNegRndSat}(a, b) = \begin{cases} max, & \text{round}(-a \cdot b > max) \\ min, & \text{round}(-a \cdot b < min) \\ \text{round}(-a \cdot b), & \text{else} \end{cases}$$

Figure 66. Algorithm formula

2.41.1 Available versions

This function is available in the following versions:

- Fractional output with mixed inputs - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The inputs are the accumulator and fractional values. The result may saturate.
- Accumulator output - the output is the accumulator type where the result can be out of the range <-1 ; 1). The result may overflow.

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

Table 42. Function versions

Function name	Input type		Result type	Description
	Mult. 1	Mult. 2		
MLIB_MulNegRndSat_F16as	acc32_t	frac16_t	frac16_t	Negative product of a 32-bit accumulator and a 16-bit fractional multiplicand; the output is rounded to the upper 16 bits of the fractional portion of the result [16..31]. The output is within the range <-1 ; 1).
MLIB_MulNegRndSat_A32	acc32_t	acc32_t	acc32_t	Negative product of two 32-bit accumulator multiplicands; the output is rounded to the middle 32 bits of the result [16..47]. The output is within the range <-65536.0 ; 65536.0).

2.41.2 Declaration

The available MLIB_MulNegRndSat functions have the following declarations:

```
frac16_t MLIB_MulNegRndSat_F16as(acc32_t a32Accum, frac16_t f16Mult)
acc32_t MLIB_MulNegRndSat_A32(acc32_t a32Mult1, acc32_t a32Mult2)
```

2.41.3 Function use

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

```
#include "mlib.h"

static acc32_t a32M1, a32M2, a32Result;

void main(void)
{
   a32M1 = ACC32(-5.5);        /* a32M1 = -5.5 */
   a32M2 = ACC32(3.1);         /* a32M2 = 3.1 */

   /* f16Result = sat(round(-a32M1 * f32M2)) */
   a32Result = MLIB_MulNegRndSat_A32(a32M1, a32M2);
}
```

2.42 MLIB_Neg

The MLIB_Neg functions return the negative value of the input. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_Neg}(x) = -x$$

Figure 67. Algorithm formula

2.42.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The result may overflow.
- Floating-point output - the output is a floating-point number; the result is within the full range.

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

Table 43. Function versions

Function name	Input type	Result type	Description
MLIB_Neg_F16	frac16_t	frac16_t	Negative value of a 16-bit fractional value. The output is within the range <-1 ; 1).
MLIB_Neg_F32	frac32_t	frac32_t	Negative value of a 32-bit fractional value. The output is within the range <-1 ; 1).
MLIB_Neg_FLT	float_t	float_t	Negative value of a 32-bit single precision floating-point value. The output is within the full range.

2.42.2 Declaration

The available MLIB_Neg functions have the following declarations:

```
frac16_t MLIB_Neg_F16(frac16_t f16Val)
frac32_t MLIB_Neg_F32(frac32_t f32Val)
float_t MLIB_Neg_FLT(float_t fltVal)
```

2.42.3 Function use

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

Fixed-point version:

```
#include "mlib.h"

static frac32_t f32Val, f32Result;

void main(void)
{
  f32Val = FRAC32(0.85);      /* f32Val = 0.85 */

  /* f32Result = -f32Val */
  f32Result = MLIB_Neg_F32(f32Val);
}
```

Floating-point version:

```
#include "mlib.h"

static float_t fltVal, fltResult;

void main(void)
{
  fltVal = 0.85F;      /* fltVal = 0.85 */

  /* fltResult = -fltVal */
  fltResult = MLIB_Neg_FLT(fltVal);
}
```

2.43 MLIB_NegSat

The MLIB_NegSat functions return the negative value of the input. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_NegSat}(x) = -x$$

Figure 68. Algorithm formula

2.43.1 Available versions

The function is available in the following versions:

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

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

Table 44. Function versions

Function name	Input type	Result type	Description
MLIB_NegSat_F16	frac16_t	frac16_t	Negative value of a 16-bit value. The output is within the range <-1 ; 1).
MLIB_NegSat_F32	frac32_t	frac32_t	Negative value of a 32-bit value. The output is within the range <-1 ; 1).

2.43.2 Declaration

The available MLIB_NegSat functions have the following declarations:

```
frac16_t MLIB_NegSat_F16(frac16_t f16Val)
frac32_t MLIB_NegSat_F32(frac32_t f32Val)
```

2.43.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Val, f32Result;

void main(void)
{
```

```
    f32Val = FRAC32(-1.0);           /* f32Val = -1.0*/

    /* f32Result = sat(-f32Val)  */
    f32Result = MLIB_NegSat_F32(f32Val);
}
```

2.44 MLIB_Rcp

The MLIB_Rcp functions return the reciprocal value for the input value. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_Rcp}(x) = \begin{cases} \max, & x = 0 \\ \min, & x = -0 \\ \frac{1}{x}, & \text{else} \end{cases}$$

Figure 69. Algorithm formula

2.44.1 Available versions

This function is available in the following versions:

- Accumulator output with fractional input - the output is the accumulator type, where the absolute value of the result is greater than or equal to 1. The input is the fractional type.

The available versions of the MLIB_Rcp function are shown in the following table.

Table 45. Function versions

Function name	Input type	Result type	Description
MLIB_Rcp_A32s	frac16_t	acc32_t	Reciprocal for a 16-bit fractional value; the output is a 32-bit accumulator value. The absolute value of the output is greater than or equal to 1. The division is performed with 32-bit accuracy.
MLIB_Rcp1_A32s	frac16_t	acc32_t	Reciprocal for a 16-bit fractional value; the output is a 32-bit accumulator value. The absolute value of the output is greater than or equal to 1. Faster version, where the division is performed with 16-bit accuracy.

2.44.2 Declaration

The available MLIB_Rcp functions have the following declarations:

```
acc32_t MLIB_Rcp_A32s(frac16_t f16Denom)
acc32_t MLIB_Rcp1_A32s(frac16_t f16Denom)
```

2.44.3 Function use

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

```
#include "mlib.h"

static acc32_t a32Result;
static frac16_t f16Denom;

void main(void)
{
```

```
    f16Denom = FRAC16(0.354);           /* f16Denom = 0.354 */

    /* a32Result = 1/f16Denom */
    a32Result = MLIB_Rcp1_A32s(f16Denom);
}
```

2.45 MLIB_Rcp1Q

The MLIB_Rcp1Q functions return the single quadrant reciprocal value for the input value. The input value must be a nonnegative number, otherwise the function returns undefined results. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_Rcp1Q}(x) = \begin{cases} \max, & x = 0 \\ \frac{1}{x}, & x > 0 \end{cases}$$

Figure 70. Algorithm formula

2.45.1 Available versions

This function is available in the following versions:

- Accumulator output with fractional input - the output is the accumulator type, where the result is greater than or equal to 1. The function is not defined for negative inputs. The input is the fractional type.

The available versions of the MLIB_Rcp1Q function are shown in the following table.

Table 46. Function versions

Function name	Input type	Result type	Description
MLIB_Rcp1Q_A32s	frac16_t	acc32_t	Reciprocal for a nonnegative 16-bit fractional value; the output is a positive 32-bit accumulator value. The output is greater than or equal to 1. The division is performed with 32-bit accuracy.
MLIB_Rcp1Q1_A32s	frac16_t	acc32_t	Reciprocal for a nonnegative 16-bit fractional value; the output is a positive 32-bit accumulator value. The output is greater than or equal to 1. Faster version, where the division is performed with 16-bit accuracy.

2.45.2 Declaration

The available MLIB_Rcp1Q functions have the following declarations:

```
acc32_t MLIB_Rcp1Q_A32s(frac16_t f16Denom)
acc32_t MLIB_Rcp1Q1_A32s(frac16_t f16Denom)
```

2.45.3 Function use

The use of the MLIB_Rcp1Q function is shown in the following example:

```
#include "mlib.h"

static acc32_t a32Result;
static frac16_t f16Denom;

void main(void)
{
    f16Denom = FRAC16(0.354);           /* f16Denom = 0.354 */
```

```
    /* a32Result = 1/f16Denom */
    a32Result = MLIB_Rcp1Q1_A32s(f16Denom);
}
```

2.46 MLIB_Rnd

The MLIB_Rnd functions round the input to the nearest value to meet the return type's size. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_Rnd}(x) = \mathrm{round}(x)$$

Figure 71. Algorithm formula

2.46.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_Rnd function are shown in the following table.

Table 47. Function versions

Function name	Input type	Result type	Description
MLIB_Rnd_F16l	frac32_t	frac16_t	Rounding of a 32-bit fractional value to a 16-bit fractional value. The output is within the range <-1 ; 1).

2.46.2 Declaration

The available MLIB_Rnd functions have the following declarations:

```
frac16_t MLIB_Rnd_F16l(frac32_t f32Val)
```

2.46.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Val;
static frac16_t f16Result;

void main(void)
{
  f32Val = FRAC32(0.85);         /* f32Val = 0.85 */

  /* f16Result = round(f32Val)*/
  f16Result = MLIB_Rnd_F16l(f32Val);
}
```

2.47 MLIB_RndSat

The MLIB_RndSat functions round the input to the nearest value to meet the return type's size. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_RndSat}(x) = \mathrm{round}(x)$$

Figure 72. Algorithm formula

2.47.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_RndSat function are shown in the following table.

Table 48. Function versions

Function name	Input type	Result type	Description
MLIB_RndSat_F16l	frac32_t	frac16_t	Rounding of a 32-bit fractional value to a 16-bit fractional value. The output is within the range <-1 ; 1).

2.47.2 Declaration

The available MLIB_RndSat functions have the following declarations:

```
frac16_t MLIB_RndSat_F16l(frac32_t f32Val)
```

2.47.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Val;
static frac16_t f16Result;

void main(void)
{
  f32Val = FRAC32(0.9997996);   /* f32Val = 0.9997996 */

  /* f16Result = sat(round(f32Val)) */
  f16Result = MLIB_RndSat_F16l(f32Val);
}
```

2.48 MLIB_Sat

The MLIB_Sat functions return the fractional portion of the accumulator input. The output is saturated if necessary. See the following equation:

$$\mathrm{MLIB\_Sat}(x) = \begin{cases} 1, & x > 1 \\ -1, & x < -1 \\ x, & \text{else} \end{cases}$$

Figure 73. Algorithm formula

2.48.1 Available versions

This function is available in the following versions:

- Fractional output with accumulator input - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The result is saturated.

The available versions of the MLIB_Sat function are shown in the following table.

Table 49. Function versions

Function name	Input type	Result type	Description
MLIB_Sat_F16a	acc32_t	frac16_t	Saturation of a 32-bit accumulator value to a 16-bit fractional value. The output is within the range <-1 ; 1).

2.48.2 Declaration

The available MLIB_Sat functions have the following declarations:

```
frac16_t MLIB_Sat_F16a(acc32_t a32Accum)
```

2.48.3 Function use

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

```
#include "mlib.h"

static acc32_t a32Accum;
static frac16_t f16Result;

void main(void)
{
   a32Accum = ACC32(5.6);            /* a32Accum = 5.6 */

   /* f16Result = sat(a32Accum) */
   f16Result = MLIB_Sat_F16a(a32Accum);
}
```

2.49 MLIB_Sh1L

The MLIB_Sh1L functions return the arithmetically one-time-shifted value to the left. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_Sh1L}(x) = x \ll 1$$

Figure 74. Algorithm formula

2.49.1 Available versions

The function is available in the following versions:

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

The available versions of the MLIB_Sh1L function are shown in the following table.

Table 50. Function versions

Function name	Input type	Result type	Description
MLIB_Sh1L_F16	frac16_t	frac16_t	Shift of a 16-bit fractional value by one time to the left. The output is within the range <-1 ; 1).
MLIB_Sh1L_F32	frac32_t	frac32_t	Shift of a 32-bit fractional value by one time to the left. The output is within the range <-1 ; 1).

2.49.2 Declaration

The available MLIB_Sh1L functions have the following declarations:

```
frac16_t MLIB_Sh1L_F16(frac16_t f16Val)
frac32_t MLIB_Sh1L_F32(frac32_t f32Val)
```

2.49.3 Function use

The use of the MLIB_Sh1L function is shown in the following example:

```
#include "mlib.h"

static frac32_t f32Result, f32Val;

void main(void)
{
   f32Val = FRAC32(-0.354);         /* f32Val = -0.354 */

   /* f32Result = f32Val << 1 */
   f32Result = MLIB_Sh1L_F32(f32Val);
}
```

2.50 MLIB_Sh1LSat

The MLIB_Sh1LSat functions return the arithmetically one-time-shifted value to the left. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_Sh1LSat}(x) = \begin{cases} 1, & x > 0.5 \\ -1, & x < -0.5 \\ x \ll 1, & \text{else} \end{cases}$$

Figure 75. Algorithm formula

2.50.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_Sh1LSat function are shown in the following table.

Table 51. Function versions

Function name	Input type	Result type	Description
MLIB_Sh1LSat_F16	frac16_t	frac16_t	Shift of a 16-bit fractional value by one time to the left. The output is within the range <-1 ; 1).
MLIB_Sh1LSat_F32	frac32_t	frac32_t	Shift of a 32-bit fractional value by one time to the left. The output is within the range <-1 ; 1).

2.50.2 Declaration

The available MLIB_Sh1LSat functions have the following declarations:

```
frac16_t MLIB_Sh1LSat_F16(frac16_t f16Val)
frac32_t MLIB_Sh1LSat_F32(frac32_t f32Val)
```

2.50.3 Function use

The use of the MLIB_Sh1LSat function is shown in the following example:

```
#include "mlib.h"

static frac16_t f16Result, f16Val;

void main(void)
{
   f16Val = FRAC16(0.354);          /* f16Val = 0.354 */

   /* f16Result = sat(f16Val << 1) */
   f16Result = MLIB_Sh1LSat_F16(f16Val);
}
```

2.51 MLIB_Sh1R

The MLIB_Sh1R functions return the arithmetically one-time-shifted value to the right. See the following equation:

$$\mathrm{MLIB\_Sh1R}(x) = x \gg 1$$

Figure 76. Algorithm formula

2.51.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-0.5 ; 0.5).

The available versions of the MLIB_Sh1R function are shown in the following table.

Table 52. Function versions

Function name	Input type	Result type	Description
MLIB_Sh1R_F16	frac16_t	frac16_t	Shift of a 16-bit fractional value by one time to the right. The output is within the range <-0.5 ; 0.5).
MLIB_Sh1R_F32	frac32_t	frac32_t	Shift of a 32-bit fractional value by one time to the right. The output is within the range <-0.5 ; 0.5).

2.51.2 Declaration

The available MLIB_Sh1R functions have the following declarations:

```
frac16_t MLIB_Sh1R_F16(frac16_t f16Val)
frac32_t MLIB_Sh1R_F32(frac32_t f32Val)
```

2.51.3 Function use

The use of the MLIB_Sh1R function is shown in the following example:

```
#include "mlib.h"

static frac32_t f32Result, f32Val;

void main(void)
{
   f32Val = FRAC32(-0.354);          /* f32Val = -0.354 */

   /* f32Result = f32Val >> 1 */
   f32Result = MLIB_Sh1R_F32(f32Val);
}
```

2.52 MLIB_ShL

The MLIB_ShL functions return the arithmetically shifted value to the left a specified number of times. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_ShL}(x, n) = x \ll \mathrm{n}$$

Figure 77. Algorithm formula

2.52.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_ShL function are shown in the following table.

Table 53. Function versions

Function name	Input type		Result type	Description
	Value	Shift		
MLIB_ShL_F16	frac16_t	uint16_t	frac16_t	Shift of a 16-bit fractional value to the left by a number of times given by the second argument; the shift is allowed within the range <0 ; 15>. The output is within the range <-1 ; 1).
MLIB_ShL_F32	frac32_t	uint16_t	frac32_t	Shift of a 32-bit fractional value to the left by a number of times given by the second argument; the shift is allowed within the range <0 ; 31>. The output is within the range <-1 ; 1).

2.52.2 Declaration

The available MLIB_ShL functions have the following declarations:

```
frac16_t MLIB_ShL_F16(frac16_t f16Val, uint16_t u16Sh)
frac32_t MLIB_ShL_F32(frac32_t f32Val, uint16_t u16Sh)
```

2.52.3 Function use

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

```
#include "mlib.h"

static frac16_t f16Result, f16Val;
static uint16_t u16Sh;

void main(void)
{
   f16Val = FRAC16(-0.354);      /* f16Val = -0.354 */
   u16Sh = 6;                    /* u16Sh = 6 */

   /* f16Result =  f16Val << u16Sh */
   f16Result = MLIB_ShL_F16(f16Val, u16Sh);
}
```

2.53 MLIB_ShLSat

The MLIB_ShLSat functions return the arithmetically shifted value to the left a specified number of times. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_ShLSat}(x, n) = \begin{cases} 1, & x > \frac{1}{2^n} \\ -1, & x < \frac{-1}{2^n} \\ x \ll n, & \text{else} \end{cases}$$

Figure 78. Algorithm formula

2.53.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_ShLSat function are shown in the following table.

Table 54. Function versions

Function name	Input type		Result type	Description
	Value	Shift		
MLIB_ShLSat_F16	frac16_t	uint16_t	frac16_t	Shift of a 16-bit fractional value to the left by a number of times given by the second argument; the shift is allowed within the range <0 ; 15>. The output is within the range <-1 ; 1).
MLIB_ShLSat_F32	frac32_t	uint16_t	frac32_t	Shift of a 32-bit fractional value to the left by a number of times given by the second argument; the shift is allowed within the range <0 ; 31>. The output is within the range <-1 ; 1).

2.53.2 Declaration

The available MLIB_ShLSat functions have the following declarations:

```
frac16_t MLIB_ShLSat_F16(frac16_t f16Val, uint16_t u16Sh)
frac32_t MLIB_ShLSat_F32(frac32_t f32Val, uint16_t u16Sh)
```

2.53.3 Function use

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

```
#include "mlib.h"

static frac16_t f16Result, f16Val;
static uint16_t u16Sh;

void main(void)
{
   f16Val = FRAC16(-0.003);      /* f16Val = -0.003 */
   u16Sh = 6;                   /* u16Sh = 6 */

   /* f16Result =  sat(f16Val << u16Sh) */
   f16Result = MLIB_ShLSat_F16(f16Val, u16Sh);
}
```

2.54 MLIB_ShR

The MLIB_ShR functions return the arithmetically shifted value to the right a specified number of times. See the following equation:

$$\mathrm{MLIB\_ShR}(x, n) = x \gg \mathrm{n}$$

Figure 79. Algorithm formula

2.54.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_ShR function are shown in the following table.

Table 55. Function versions

Function name	Input type		Result type	Description
	Value	Shift		
MLIB_ShR_F16	frac16_t	uint16_t	frac16_t	Shift of a 16-bit fractional value to the right by a number of times given by the second argument; the shift is allowed within the range <0 ; 15>. The output is within the range <-1 ; 1).
MLIB_ShR_F32	frac32_t	uint16_t	frac32_t	Shift of a 32-bit fractional value to the right by a number of times given by the second argument; the shift is allowed within the range <0 ; 31>. The output is within the range <-1 ; 1).

2.54.2 Declaration

The available MLIB_ShR functions have the following declarations:

```
frac16_t MLIB_ShR_F16(frac16_t f16Val, uint16_t u16Sh)
frac32_t MLIB_ShR_F32(frac32_t f32Val, uint16_t u16Sh)
```

2.54.3 Function use

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

```
#include "mlib.h"

static frac16_t f16Result, f16Val;
static uint16_t u16Sh;

void main(void)
{
   f16Val = FRAC32(-0.354);      /* f16Val = -0.354 */
   u16Sh = 8;                    /* u16Sh = 8 */

   /* f16Result = f16Val >> u16Sh */
   f16Result = MLIB_ShR_F16(f16Val, u16Sh);
}
```

2.55 MLIB_ShLBi

The MLIB_ShLBi functions return the arithmetically shifted value to the left a specified number of times. If the number of shifts is positive, the shift is performed to the left; if negative, to the right. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_ShLBi}(x, n) = x \ll \mathrm{n}$$

Figure 80. Algorithm formula

2.55.1 Available versions

The function is available in the following versions:

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

The available versions of the MLIB_ShLBi function are shown in the following table.

Table 56. Function versions

Function name	Input type		Result type	Description
	Value	Shift		
MLIB_ShLBi_F16	frac16_t	int16_t	frac16_t	Bidirectional shift of a 16-bit fractional value to the left by a number of times given by the second argument; if the second argument is negative, the shift is performed to the right. The shift is allowed within the range <-15 ; 15>. The output is within the range <-1 ; 1).
MLIB_ShLBi_F32	frac32_t	int16_t	frac32_t	Bidirectional shift of a 32-bit fractional value to the left by a number of times given by the second argument; if the second argument is negative, the shift is performed to the right. The shift is allowed within the range <-31 ; 31>. The output is within the range <-1 ; 1).

2.55.2 Declaration

The available MLIB_ShLBi functions have the following declarations:

```
frac16_t MLIB_ShLBi_F16(frac16_t f16Val, int16_t i16Sh)
frac32_t MLIB_ShLBi_F32(frac32_t f32Val, int16_t i16Sh)
```

2.55.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Result, f32Val;
static int16_t i16Sh;

void main(void)
{
   f32Val = FRAC32(-0.354);      /* f32Val = -0.354 */
   i16Sh = -3;                   /* i16Sh = -3 */

   /* f32Result = f32Val << i16Sh */
   f32Result = MLIB_ShLBi_F32(f32Val, i16Sh);
}
```

2.56 MLIB_ShLBiSat

The MLIB_ShLBiSat functions return the arithmetically shifted value to the left a specified number of times. If the number of shifts is positive, the shift is performed to the left; if negative, to the right. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_ShLBiSat}(x, n) = \begin{cases} 1, & x > \frac{1}{2^n} \wedge n > 0 \\ -1, & x < \frac{-1}{2^n} \wedge n > 0 \\ x \ll \mathrm{n}, & \mathrm{else} \end{cases}$$

Figure 81. Algorithm formula

2.56.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_ShLBiSat function are shown in the following table.

Table 57. Function versions

Function name	Input type		Result type	Description
	Value	Shift		
MLIB_ShLBiSat_F16	frac16_t	int16_t	frac16_t	Bidirectional shift of a 16-bit fractional value to the left by a number of times given by the second argument; if the second argument is negative, the shift is performed to the right. The shift is allowed within the range <-15 ; 15>. The output is within the range <-1 ; 1).
MLIB_ShLBiSat_F32	frac32_t	int16_t	frac32_t	Bidirectional shift of a 32-bit fractional value to the left by a number of times given by the second argument; if the second argument is negative, the shift is performed to the right. The shift is allowed within the range <-31 ; 31>. The output is within the range <-1 ; 1).

2.56.2 Declaration

The available MLIB_ShLBiSat functions have the following declarations:

```
frac16_t MLIB_ShLBiSat_F16(frac16_t f16Val, int16_t i16Sh)
frac32_t MLIB_ShLBiSat_F32(frac32_t f32Val, int16_t i16Sh)
```

2.56.3 Function use

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

```
#include "mlib.h"

static frac16_t f16Result, f16Val;
static int16_t i16Sh;

void main(void)
{
   f16Val = FRAC16(-0.354);      /* f16Val = -0.354 */
   i16Sh = 14;                   /* i16Sh = 14 */

   /* f16Result = sat(f16Val << i16Sh) */
   f16Result = MLIB_ShLBiSat_F16(f16Val, i16Sh);
}
```

2.57 MLIB_ShRBi

The MLIB_ShRBi functions return the arithmetically shifted value to the right a specified number of times. If the number of shifts is positive, the shift is performed to the right; if negative, to the left. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_ShRBi}(x, n) = x \gg \mathrm{n}$$

Figure 82. Algorithm formula

2.57.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_ShRBi function are shown in the following table.

Table 58. Function versions

Function name	Input type		Result type	Description
	Value	Shift		
MLIB_ShRBi_F16	frac16_t	int16_t	frac16_t	Bidirectional shift of a 16-bit fractional value to the right by a number of times given by the second argument; if the second argument is negative, the shift is performed to the left. The shift is allowed within the range <-15 ; 15>. The output is within the range <-1 ; 1).
MLIB_ShRBi_F32	frac32_t	int16_t	frac32_t	Bidirectional shift of a 32-bit fractional value to the right by a number of times given by the second argument; if the second argument is negative, the shift is performed to the left. The shift is allowed within the range <-31 ; 31>. The output is within the range <-1 ; 1).

2.57.2 Declaration

The available MLIB_ShRBi functions have the following declarations:

```
frac16_t MLIB_ShRBi_F16(frac16_t f16Val, int16_t i16Sh)
frac32_t MLIB_ShRBi_F32(frac32_t f32Val, int16_t i16Sh)
```

2.57.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Result, f32Val;
static int16_t i16Sh;

void main(void)
{
    f32Val = FRAC32(0.354);      /* f32In = 0.354 */
    i16Sh = 8;                   /* i16Sh = 8 */

    /* f32Result = f32Val >> i16Sh */
```

```
    f32Result = MLIB_ShRBi_F32(f32Val, i16Sh);
}
```

2.58 MLIB_ShRBiSat

The MLIB_ShRBiSat functions return the arithmetically shifted value to the right a specified number of times. If the number of shifts is positive, the shift is performed to the right; if negative, to the left. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_ShRBiSat}(x, n) = \begin{cases} 1, & x > \frac{1}{2^n} \wedge n < 0 \\ -1, & x < \frac{-1}{2^n} \wedge n < 0 \\ x \gg \mathrm{n}, & \text{else} \end{cases}$$

Figure 83. Algorithm formula

2.58.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_ShRBiSat function are shown in the following table.

Table 59. Function versions

Function name	Input type		Result type	Description
	Value	Shift		
MLIB_ShRBiSat_F16	frac16_t	int16_t	frac16_t	Bidirectional shift of a 16-bit fractional value to the right by a number of times given by the second argument; if the second argument is negative, the shift is performed to the left. The shift is allowed within the range <-15 ; 15>. The output is within the range <-1 ; 1).
MLIB_ShRBiSat_F32	frac32_t	int16_t	frac32_t	Bidirectional shift of a 32-bit fractional value to the right by a number of times given by the second argument; if the second argument is negative, the shift is performed to the left. The shift is allowed within the range <-31 ; 31>. The output is within the range <-1 ; 1).

2.58.2 Declaration

The available MLIB_ShRBiSat functions have the following declarations:

```
frac16_t MLIB_ShRBiSat_F16(frac16_t f16Val, int16_t i16Sh)
frac32_t MLIB_ShRBiSat_F32(frac32_t f32Val, int16_t i16Sh)
```

2.58.3 Function use

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

```
include "mlib.h"

static frac32_t f32Result, f32Val;
static int16_t i16Sh;

void main(void)
```

```
{
   f32Val = FRAC32(-0.354);      /* f32Val = -0.354 */
   i16Sh = 13;                   /* i16Sh = 13 */

   /* f32Result = sat(f32Val >> i16Sh) */
   f32Result = MLIB_ShRBiSat_F32(f32Val, i16Sh);
}
```

2.59 MLIB_Sign

The MLIB_Sign functions return the sign of the input. See the following equation:

$$\mathrm{MLIB\_Sign}(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases}$$

Figure 84. Algorithm formula

2.59.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1).
- Floating-point output - the output is a floating-point number; the result is within the full range.

The available versions of the MLIB_Sign function are shown in the following table.

Table 60. Function versions

Function name	Input type	Result type	Description
MLIB_Sign_F16	frac16_t	frac16_t	Sign of a 16-bit fractional value. The output is within the range <-1 ; 1).
MLIB_Sign_F32	frac32_t	frac32_t	Sign of a 32-bit fractional value. The output is within the range <-1 ; 1).
MLIB_Sign_FLT	float_t	float_t	Sign of a 32-bit single precision floating-point value. The output is within the full range.

2.59.2 Declaration

The available MLIB_Sign functions have the following declarations:

```
frac16_t MLIB_Sign_F16(frac16_t f16Val)
frac32_t MLIB_Sign_F32(frac32_t f32Val)
float_t MLIB_Sign_FLT(float_t fltVal)
```

2.59.3 Function use

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

Fixed-point version:

```
#include "mlib.h"

static frac32_t f32In, f32Result;

void main(void)
```

```
{
  f32In = FRAC32(-0.95);         /* f32In = -0.95 */

  /* f32Result = sign(f32In)*/
  f32Result = MLIB_Sign_F32(f32In);
}
```

Floating-point version:

```
#include "mlib.h"

static float_t fltIn, fltResult;

void main(void)
{
  fltIn = -0.95F;         /* fltIn = -0.95 */

  /* fltResult = sign(fltIn)*/
  fltResult = MLIB_Sign_FLT(fltIn);
}
```

2.60 MLIB_Sub

The MLIB_Sub functions subtract the subtrahend from the minuend. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_Sub}(a, b) = a - b$$

Figure 85. Algorithm formula

2.60.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The result may overflow.
- Accumulator output with fractional inputs - the output is the accumulator type, where the result can be out of the range <-1 ; 1). The inputs are the fractional values only.
- Accumulator output with mixed inputs - the output is the accumulator type, where the result can be out of the range <-1 ; 1). The inputs are the accumulator and fractional values. The result may overflow.
- Floating-point output - the output is a floating-point number; the result is within the full range.

The available versions of the MLIB_Sub function are shown in the following table.

Table 61. Function versions

Function name	Input type		Result type	Description
	Minuend	Subtrahend		
MLIB_Sub_F16	frac16_t	frac16_t	frac16_t	Subtraction of a 16-bit fractional subtrahend from a 16-bit fractional minuend. The output is within the range <-1 ; 1).

Table continues on the next page...

Table 61. Function versions (continued)

Function name	Input type		Result type	Description
	Minuend	Subtrahend		
MLIB_Sub_F32	frac32_t	frac32_t	frac32_t	Subtraction of a 32-bit fractional subtrahend from a 32-bit fractional minuend. The output is within the range <-1 ; 1).
MLIB_Sub_A32ss	frac16_t	frac16_t	acc32_t	Subtraction of a 16-bit fractional subtrahend from a 16-bit fractional minuend; the result is a 32-bit accumulator. The output may be out of the range <-65536 ; 65536).
MLIB_Sub_A32as	acc32_t	frac16_t	acc32_t	Subtraction of a 16-bit fractional subtrahend from a 32-bit accumulator. The output may be out of the range <-65536 ; 65536).
MLIB_Sub_FLT	float_t	float_t	float_t	Subtraction of a 32-bit single precision floating-point subtrahend from a 32-bit single precision floating-point minuend. The output is within the full range.

2.60.2 Declaration

The available MLIB_Sub functions have the following declarations:

```
frac16_t MLIB_Sub_F16(frac16_t f16Min, frac16_t f16Sub)
frac32_t MLIB_Sub_F32(frac32_t f32Min, frac32_t f32Sub)
acc32_t MLIB_Sub_A32ss(frac16_t f16Min, frac16_t f16Sub)
acc32_t MLIB_Sub_A32as(acc32_t a32Accum, frac16_t f16Sub)
float_t MLIB_Sub_FLT(float_t fltMin, float_t fltSub)
```

2.60.3 Function use

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

Fixed-point version:

```
#include "mlib.h"

static acc32_t a32Accum, a32Result;
static frac16_t f16Sub;

void main(void)
{
  a32Accum = ACC32(4.5);         /* a32Accum = 4.5 */
  f16Sub = FRAC16(0.4);          /* f16Sub = 0.4 */

  /* a32Result = a32Accum - f16Sub */
  a32Result = MLIB_Sub_A32as(a32Accum,f16Sub);
}
```

Floating-point version:

```
#include "mlib.h"

static float_t fltMin, fltResult, fltSub;
```

```
void main(void)
{
  fltMin = 4.5F;         /* fltMin = 4.5 */
  fltSub = 0.4F;          /* fltSub = 0.4 */

  /* fltResult = fltMin - fltSub */
  fltResult = MLIB_Sub_FLT(fltMin,fltSub);
}
```

2.61 MLIB_SubSat

The MLIB_SubSat functions subtract the subtrahend from the minuend. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_SubSat}(a, b) = \begin{cases} 1, & a - b > 1 \\ -1, & a - b < -1 \\ a - b, & \text{else} \end{cases}$$

Figure 86. Algorithm formula

2.61.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_SubSat function are shown in the following table.

Table 62. Function versions

Function name	Input type		Result type	Description
	Minuend	Subtrahend		
MLIB_SubSat_F16	frac16_t	frac16_t	frac16_t	Subtraction of a 16-bit fractional subtrahend from a 16-bit fractional minuend. The output is within the range <-1 ; 1).
MLIB_SubSat_F32	frac32_t	frac32_t	frac32_t	Subtraction of a 32-bit fractional subtrahend from a 32-bit fractional minuend. The output is within the range <-1 ; 1).

2.61.2 Declaration

The available MLIB_SubSat functions have the following declarations:

```
frac16_t MLIB_SubSat_F16(frac16_t f16Min, frac16_t f16Sub)
frac32_t MLIB_SubSat_F32(frac32_t f32Min, frac32_t f32Sub)
```

2.61.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Min, f32Sub, f32Result;

void main(void)
```

```
{
  f32Min = FRAC32(-0.5);          /* f32Min = -0.5 */
  f32Sub = FRAC32(0.8);           /* f32Sub = 0.8 */

  /* f32Result = sat(f32Min - f32Sub) */
  f32Result = MLIB_SubSat_F32(f32Min,f32Sub);
}
```

2.62 MLIB_Sub4

The MLIB_Sub4 functions return the subtraction of three subtrahends from the minuend. The function does not saturate the output. See the following equation:

$$\mathrm{MLIB\_Sub4}(a, b, c, d) = a - b - c - d$$

Figure 87. Algorithm formula

2.62.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The result may overflow.
- Floating-point output - the output is a floating-point number; the result is within the full range.

The available versions of the MLIB_Sub4 function are shown in the following table.

Table 63. Function versions

Function name	Input type				Result type	Description
	Minuend	Sub. 1	Sub. 2	Sub. 3		
MLIB_Sub4_F16	frac16_t	frac16_t	frac16_t	frac16_t	frac16_t	Subtraction of three 16-bit fractional subtrahends from 16-bit fractional minuend. The output is within the range <-1 ; 1).
MLIB_Sub4_F32	frac32_t	frac32_t	frac32_t	frac32_t	frac32_t	Subtraction of three 32-bit fractional subtrahends from 32-bit fractional minuend. The output is within the range <-1 ; 1).
MLIB_Sub4_FLT	float_t	float_t	float_t	float_t	float_t	Subtraction of three 32-bit single precision floating-point subtrahends from 32-bit single precision floating-point. The output is within the full range.

2.62.2 Declaration

The available MLIB_Sub4 functions have the following declarations:

```
frac16_t MLIB_Sub4_F16(frac16_t f16Min, frac16_t f16Sub1, frac16_t f16Sub2, frac16_t f16Sub3)
frac32_t MLIB_Sub4_F32(frac32_t f32Min, frac32_t f32Sub1, frac32_t f32Sub2, frac32_t f32Sub3)
float_t MLIB_Sub4_FLT(float_t fltMin, float_t fltSub1, float_t fltSub2, float_t fltSub3)
```

2.62.3 Function use

The use of the MLIB_Sub4 function is shown in the following examples:

Fixed-point version:

```
#include "mlib.h"

static frac16_t f16Result, f16Min, f16Sub1, f16Sub2, f16Sub3;

void main(void)
{
   f16Min = FRAC16(0.2);         /* f16Min = 0.2 */
   f16Sub1 = FRAC16(0.3);        /* f16Sub1 = 0.3 */
   f16Sub2 = FRAC16(-0.5);       /* f16Sub2 = -0.5 */
   f16Sub3 = FRAC16(0.2);        /* f16Sub3 = 0.2 */

   /* f16Result = sat(f16Min - f16Sub1 - f16Sub2 - f16Sub3) */
   f16Result = MLIB_Sub4_F16(f16Min, f16Sub1, f16Sub2, f16Sub3);
}
```

Floating-point version:

```
#include "mlib.h"

static float_t fltResult, fltMin, fltSub1, fltSub2, fltSub3;

void main(void)
{
   fltMin = 0.2F;          /* fltMin = 0.2 */
   fltSub1 = 0.3F;         /* fltSub1 = 0.3 */
   fltSub2 = -0.5F;        /* fltSub2 = -0.5 */
   fltSub3 = 0.2F;         /* fltSub3 = 0.2 */

   /* fltResult = sat(fltMin - fltSub1 - fltSub2 - fltSub3) */
   fltResult = MLIB_Sub4_FLT(fltMin, fltSub1, fltSub2, fltSub3);
}
```

2.63 MLIB_Sub4Sat

The MLIB_Sub4Sat functions return the subtraction of three subtrahends from the minuend. The function saturates the output. See the following equation:

$$\mathrm{MLIB\_Sub4Sat}(a, b, c, d) = \begin{cases} 1, & a-b-c-d > 1 \\ -1, & a-b-c-d < -1 \\ a-b-c-d, & \text{else} \end{cases}$$

Figure 88. Algorithm formula

2.63.1 Available versions

This function is available in the following versions:

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

The available versions of the MLIB_Sub4Sat function are shown in the following table.

Table 64. Function versions

Function name	Input type				Result type	Description
	Minuend	Sub. 1	Sub. 2	Sub. 3		
MLIB_Sub4Sat_F16	frac16_t	frac16_t	frac16_t	frac16_t	frac16_t	Subtraction of three 16-bit fractional subtrahends from 16-bit fractional minuend. The output is within the range <-1 ; 1).
MLIB_Sub4Sat_F32	frac32_t	frac32_t	frac32_t	frac32_t	frac32_t	Subtraction of three 32-bit fractional subtrahends from 32-bit fractional minuend. The output is within the range <-1 ; 1).

2.63.2 Declaration

The available MLIB_Sub4Sat functions have the following declarations:

```
frac16_t MLIB_Sub4Sat_F16(frac16_t f16Min, frac16_t f16Sub1, frac16_t f16Sub2, frac16_t f16Sub3)

frac32_t MLIB_Sub4Sat_F32(frac32_t f32Min, frac32_t f32Sub1, frac32_t f32Sub2, frac32_t f32Sub3)
```

2.63.3 Function use

The use of the MLIB_Sub4Sat function is shown in the following example:

```
#include "mlib.h"

static frac32_t f32Result, f32Min, f32Sub1, f32Sub2, f32Sub3;

void main(void)
{
   f32Min = FRAC32(0.2);         /* f32Min = 0.2 */
   f32Sub1 = FRAC32(0.8);        /* f32Sub1 = 0.8 */
   f32Sub2 = FRAC32(-0.1);       /* f32Sub2 = -0.1 */
   f32Sub3 = FRAC32(0.7);        /* f32Sub3 = 0.7 */

   /* f32Result = sat(f32Min - f32Sub1 - f32Sub2 - f32Sub3) */
   f32Result = MLIB_Sub4Sat_F32(f32Min, f32Sub1, f32Sub2, f32Sub3);
}
```

Appendix A Library types

A.1 bool_t

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

```
typedef unsigned short bool_t;
```

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

Table 65. Data storage

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

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

A.2 uint8_t

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

```
typedef unsigned char uint8_t;
```

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

Table 66. Data storage

	7	6	5	4	3	2	1	0
Value	Integer							
255	1	1	1	1	1	1	1	1
	F				F			

Table continues on the next page...

Table 66. Data storage (continued)

11	0	0	0	0	1	0	1	1
	0				B			
124	0	1	1	1	1	1	0	0
	7				C			
159	1	0	0	1	1	1	1	1
	9				F			

A.3 uint16_t

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

```
typedef unsigned short uint16_t;
```

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

Table 67. Data storage

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

A.4 uint32_t

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

```
typedef unsigned long uint32_t;
```

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

Table 68. Data storage

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

A.5 int8_t

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

```
typedef char int8_t;
```

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

Table 69. Data storage

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

Table continues on the next page...

Table 69. Data storage (continued)

-97	1	0	0	1	1	1	1	1
	9				F			

A.6 int16_t

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

```
typedef short int16_t;
```

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

Table 70. Data storage

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

A.7 int32_t

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

```
typedef long int32_t;
```

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

Table 71. Data storage

Table continues on the next page...

Table 71. Data storage (continued)

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

A.8 frac8_t

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

```
typedef char frac8_t;
```

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

Table 72. Data storage

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

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

A.9 frac16_t

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

```
typedef short frac16_t;
```

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

Table 73. Data storage

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	Sign	Fractional														
0.99997	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	7				F				F				F			
-1.0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	8				0				0				0			
0.47357	0	0	1	1	1	1	0	0	1	0	0	1	1	1	1	0
	3				C				9				E			
-0.75586	1	0	0	1	1	1	1	1	0	1	0	0	0	0	0	0
	9				F				4				0			

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

A.10 frac32_t

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

```
typedef long frac32_t;
```

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

Table 74. Data storage

	31	24	23	16	15	8	7	0
Value	S	Fractional						
0.9999999995	7	F	F	F	F	F	F	F

Table continues on the next page...

Table 74. Data storage (continued)

-1.0	8	0	0	0	0	0	0	0
0.02606645970	0	3	5	6	2	5	5	0
-0.3929787632	C	D	B	2	D	F	3	4

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

A.11 acc16_t

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

```
typedef short acc16_t;
```

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

Table 75. Data storage

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

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

A.12 acc32_t

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

```
typedef long acc32_t;
```

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

Table 76. Data storage

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

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

A.13 float_t

The float_t type is a signed 32-bit single precision floating-point type, defined by IEEE 754. It is able to store the full precision (normalized) finite variables within the range <$-3.40282 \cdot 10^{38}$; $3.40282 \cdot 10^{38}$) with the minimum resolution of 2^{-23}. The smallest normalized number is $\pm 1.17549 \cdot 10^{-38}$. Nevertheless, the denormalized numbers (with reduced precision) reach yet lower values, from $\pm 1.40130 \cdot 10^{-45}$ to $\pm 1.17549 \cdot 10^{-38}$. The standard also defines the additional values:

- Negative zero
- Infinity
- Negative infinity
- Not a number

The 32-bit type is composed of:

- Sign (bit 31)
- Exponent (bits 23 to 30)
- Mantissa (bits 0 to 22)

The conversion of the number is straighforward. The sign of the number is stored in bit 31. The binary exponent is decoded as an integer from bits 23 to 30 by subtracting 127. The mantissa (fraction) is stored in bits 0 to 22. An invisible leading bit (it is not actually stored) with value 1.0 is placed in front; therefore, bit 23 has a value of 0.5, bit 22 has a value 0.25, and so on. As a result, the mantissa has a value between 1.0 and 2. If the exponent reaches -127 (binary 00000000), the leading 1.0 is no longer used to enable the gradual underflow.

The float_t type definition is as follows:

```
typedef float float_t;
```

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

Table 77. Data storage - normalized values

Value	31 … 24	23 … 16	15 … 8	7 … 0
	S, Exponent	Exponent, Mantissa	Mantissa	Mantissa
$(2.0 - 2^{-23}) \cdot 2^{127}$	0 1 1 1 1 1 1 1	0 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1
$\approx 3.40282 \cdot 10^{38}$	7 F	7 F	F F	F F
$-(2.0 - 2^{-23}) \cdot 2^{127}$	1 1 1 1 1 1 1 1	0 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1
$\approx -3.40282 \cdot 10^{38}$	F F	7 F	F F	F F
2^{-126}	0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
$\approx 1.17549 \cdot 10^{-38}$	0 0	8 0	0 0	0 0
-2^{-126}	1 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
$\approx -1.17549 \cdot 10^{-38}$	8 0	8 0	0 0	0 0
1.0	0 0 1 1 1 1 1 1	1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
	3 F	8 0	0 0	0 0
-1.0	1 0 1 1 1 1 1 1	1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
	B F	8 0	0 0	0 0
π	0 1 0 0 0 0 0 0	0 1 0 0 1 0 0 1	0 0 0 0 1 1 1 1	1 1 0 1 1 0 1 1
≈ 3.1415927	4 0	4 9	0 F	D B
-20810.086	1 1 0 0 0 1 1 0	1 0 1 0 0 0 1 0	1 0 0 1 0 1 0 0	0 0 1 0 1 1 0 0
	C 6	A 2	9 4	2 C

Table continues on the next page...

Table 77. Data storage - normalized values (continued)

Table 78. Data storage - denormalized values

<table>
<tr><td></td><td colspan="3">31   24 23   16 15   8 7   0</td></tr>
<tr><td>Value</td><td>S</td><td>Exponent</td><td>Mantissa</td></tr>
<tr><td>0.0</td><td>0</td><td>0 0 0 0 0 0 0 0</td><td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td></tr>
<tr><td></td><td colspan="3">0 0 | 0 0 | 0 0 | 0 0</td></tr>
<tr><td>-0.0</td><td>1</td><td>0 0 0 0 0 0 0 0</td><td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td></tr>
<tr><td></td><td colspan="3">8 0 | 0 0 | 0 0 | 0 0</td></tr>
<tr><td>$(1.0 - 2^{-23}) \cdot 2^{-126}$</td><td>0</td><td>0 0 0 0 0 0 0 0</td><td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td></tr>
<tr><td>$\approx 1.17549 \cdot 10^{-38}$</td><td colspan="3">0 0 | 7 F | F F | F F</td></tr>
<tr><td>$-(1.0 - 2^{-23}) \cdot 2^{-126}$</td><td>1</td><td>0 0 0 0 0 0 0 0</td><td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td></tr>
<tr><td>$\approx -1.17549 \cdot 10^{-38}$</td><td colspan="3">8 0 | 7 F | F F | F F</td></tr>
<tr><td>$2^{-1} \cdot 2^{-126}$</td><td>0</td><td>0 0 0 0 0 0 0 0</td><td>1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td></tr>
<tr><td>$\approx 5.87747 \cdot 10^{-39}$</td><td colspan="3">0 0 | 4 0 | 0 0 | 0 0</td></tr>
<tr><td>$-2^{-1} \cdot 2^{-126}$</td><td>1</td><td>0 0 0 0 0 0 0 0</td><td>1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td></tr>
<tr><td>$\approx -5.87747 \cdot 10^{-39}$</td><td colspan="3">8 0 | 4 0 | 0 0 | 0 0</td></tr>
<tr><td>$2^{-23} \cdot 2^{-126}$</td><td>0</td><td>0 0 0 0 0 0 0 0</td><td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td></tr>
<tr><td>$\approx 1.40130 \cdot 10^{-45}$</td><td colspan="3">0 0 | 0 0 | 0 0 | 0 1</td></tr>
<tr><td>$-2^{-23} \cdot 2^{-126}$</td><td>1</td><td>0 0 0 0 0 0 0 0</td><td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td></tr>
<tr><td>$\approx -1.40130 \cdot 10^{-45}$</td><td colspan="3">8 0 | 0 0 | 0 0 | 0 1</td></tr>
</table>

Table 79. Data storage - special values

Value	S	Exponent	Mantissa
	31	24 23	16 15 8 7 0
∞	0	1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
	7 F		8 0 0 0 0 0
-∞	1	1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
	F F		8 0 0 0 0 0
Not a number	*	1 1 1 1 1 1 1 1	non zero
	7/F F		800001 to FFFFFF

A.14 FALSE

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

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

```
#include "mlib.h"

static bool_t bVal;

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

A.15 TRUE

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

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

```
#include "mlib.h"

static bool_t bVal;

void main(void)
{
```

```
  bVal = TRUE;                /* bVal = TRUE */
}
```

A.16 FRAC8

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

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

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

```
#include "mlib.h"

static frac8_t f8Val;

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

A.17 FRAC16

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

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

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

```
#include "mlib.h"

static frac16_t f16Val;

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

A.18 FRAC32

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

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

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

```
#include "mlib.h"
```

```
static frac32_t f32Val;

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

A.19 ACC16

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

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

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

```
#include "mlib.h"

static acc16_t a16Val;

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

A.20 ACC32

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

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

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

```
#include "mlib.h"

static acc32_t a32Val;

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

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 01 November 2021

Document identifier: CM7FMLIBUG