MLIB User's Guide

DSP56800E

Document Number: DSP56800EMLIBUG
Rev. 2, 10/2015

Contents

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

Chapter 1
Library

Chapter 2
Algorithms in detail

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Chapter 1
Library

1.1 Introduction

1.1.1 Overview

This user's guide describes the Math Library (MLIB) for the family of DSP56800E core-based digital signal controllers. This library contains optimized functions.

1.1.2 Data types

MLIB supports several data types: (un)signed integer, fractional, and accumulator. 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 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}

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
- lss—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 1-1. Input/output types

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

1.1.4 Supported compilers

MLIB for the DSP56800E core is written in assembly language with C-callable interface. The library is built and tested using the following compilers:

- CodeWarrior™ Development Studio

For the CodeWarrior™ Development Studio, 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 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 require the core saturation mode to be turned off, otherwise the results can be incorrect. Several specific library functions are immune to the setting of the saturation mode.
3. The library functions round the result (the API contains Rnd) to the nearest (two's complement rounding) or to the nearest even number (convergent round). The mode used depends on the core option mode register (OMR) setting. See the core manual for details.
4. All non-inline functions are implemented without storing any of the volatile registers (refer to the compiler manual) used by the respective routine. Only the non-volatile registers (C10, D10, R5) are saved by pushing the registers on the stack. Therefore, if the particular registers initialized before the library function call are to be used after the function call, it is necessary to save them manually.

1.2 Library integration into project (CodeWarrior™ Development Studio)

This section provides a step-by-step guide to quickly and easily integrate the MLIB into an empty project using CodeWarrior™ Development Studio. This example uses the MC56F8257 part, and the default installation path (C:\Freescale\FSLESL \DSP56800E_FSLESL_4.2) is supposed. If you have a different installation path, you must use that path instead.

1.2.1 New project

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

1. Launch CodeWarrior™ Development Studio.
2. Choose File > New > Bareboard Project, so that the "New Bareboard Project" dialog appears.
3. Type a name of the project, for example, MyProject01.

4. If you don't use the default location, untick the “Use default location” checkbox, and type the path where you want to create the project folder; for example, C: \CWProjects\MyProject01, and click Next. See Figure 1-1.

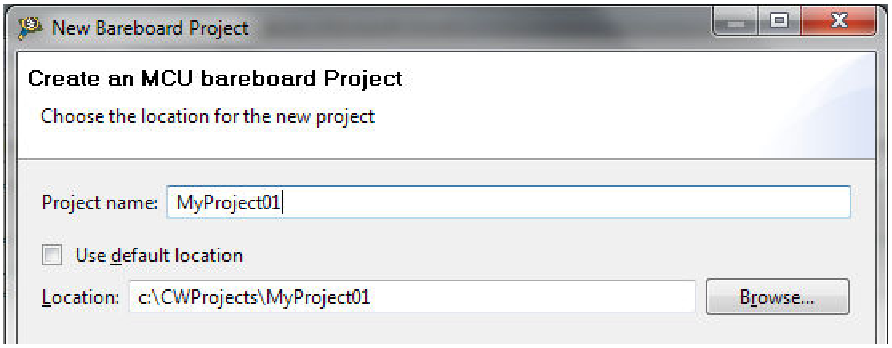

Figure 1-1. Project name and location

5. Expand the tree by clicking the 56800/E (DSC) and MC56F8257. Select the Application option and click Next. See Figure 1-2.

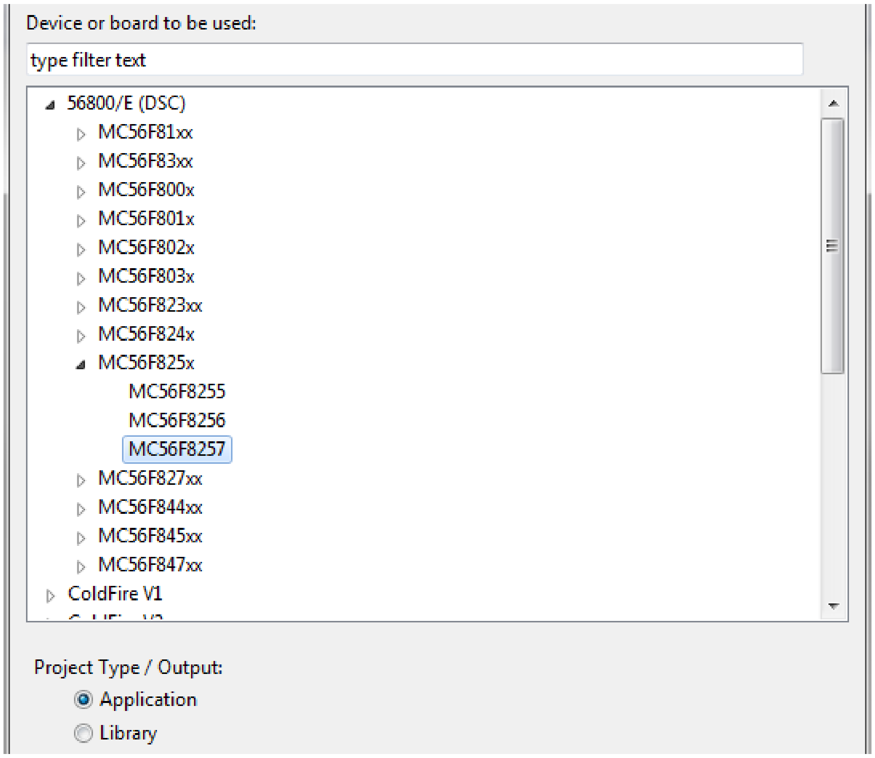

Figure 1-2. Processor selection

6. Now select the connection that will be used to download and debug the application. In this case, select the option P&E USB MultiLink Universal[FX] / USB MultiLink and Freescale USB TAP, and click Next. See Figure 1-3.

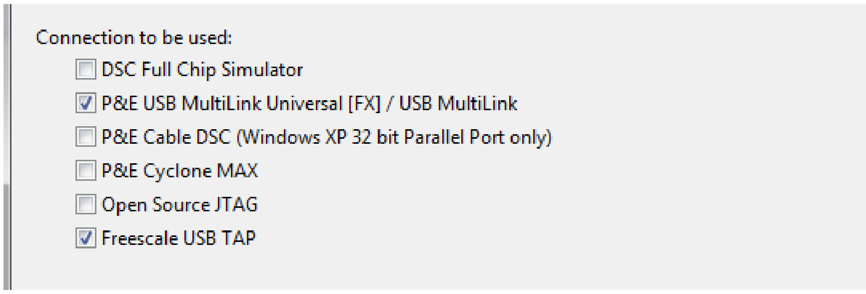

Figure 1-3. Connection selection

7. From the options given, select the Simple Mixed Assembly and C language, and click Finish. See Figure 1-4.

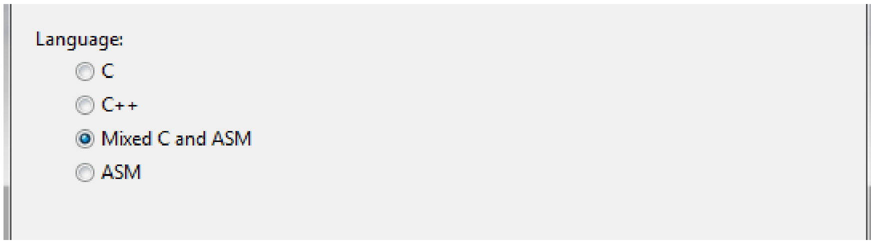

Figure 1-4. Language choice

The new project is now visible in the left-hand part of CodeWarrior™ Development Studio. See Figure 1-5.

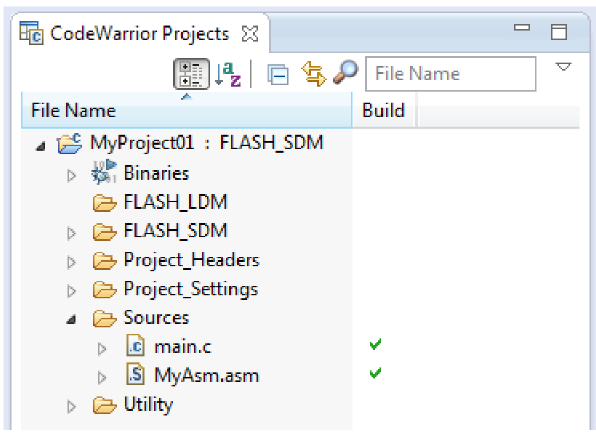

Figure 1-5. Project folder

1.2.2 Library path variable

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

1. Right-click the MyProject01 node in the left-hand part and click Properties, or select Project > Properties from the menu. The project properties dialog appears.

2. Expand the Resource node and click Linked Resources. See Figure 1-6.

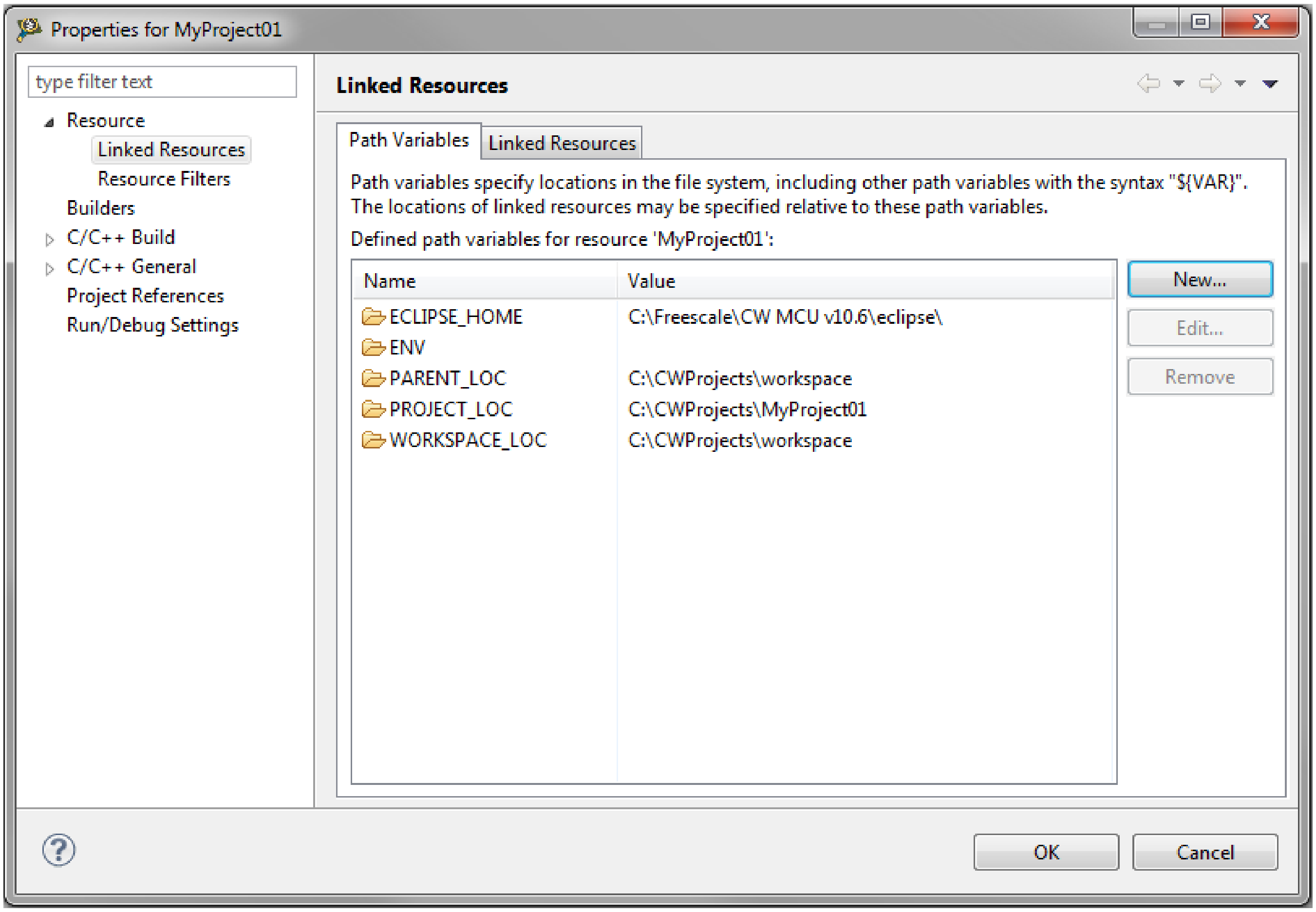

Figure 1-6. Project properties

3. Click the 'New…' button on the right-hand side.
4. In the dialog that appears (see Figure 1-7), type this variable name into the Name box: FSLESL_LOC
5. Select the library parent folder by clicking 'Folder…' or just typing the following path into the Location box: C:\Freescale\FSLESL\DSP56800E_FSLESL_4.2_CW and click OK.
6. Click OK in the previous dialog.

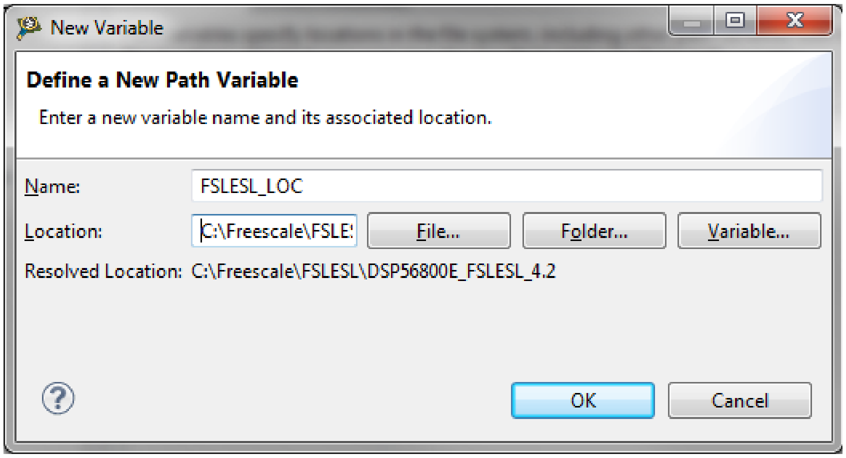

Figure 1-7. New variable

1.2.3 Library folder addition

To use the library, add it into the CodeWarrior Project tree dialog.

1. Right-click the MyProject01 node in the left-hand part and click New > Folder, or select File > New > Folder from the menu. A dialog appears.
2. Click Advanced to show the advanced options.
3. To link the library source, select the third option—Link to alternate location (Linked Folder).
4. Click Variables…, and select the FSLESL_LOC variable in the dialog that appears, click OK, and/or type the variable name into the box. See Figure 1-8.
5. Click Finish, and you will see the library folder linked in the project. See Figure 1-9

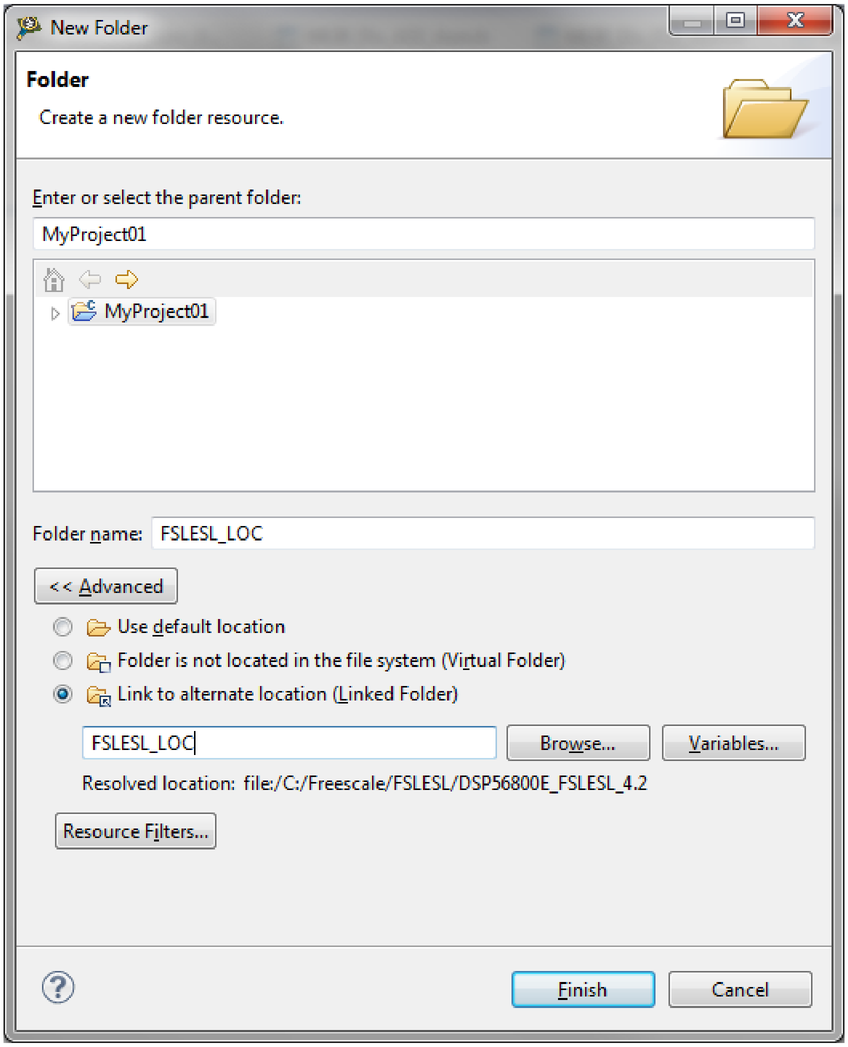

Figure 1-8. Folder link

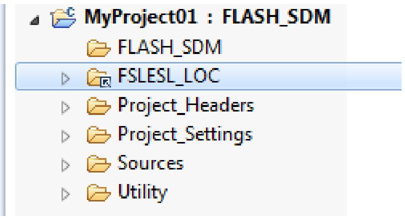

Figure 1-9. Projects libraries paths

1.2.4 Library path setup

1. Right-click the MyProject01 node in the left-hand part and click Properties, or select Project > Properties from the menu. A dialog with the project properties appears.
2. Expand the C/C++ Build node, and click Settings.
3. In the right-hand tree, expand the DSC Linker node, and click Input. See Figure 1-11.
4. In the third dialog Additional Libraries, click the 'Add…' icon, and a dialog appears.

5. Look for the FSLESL_LOC variable by clicking Variables…, and then finish the path in the box by adding one of the following:
    - ${FSLESL_LOC}\MLIB\mlib_SDM.lib—for small data model projects
    - ${FSLESL_LOC}\MLIB\mlib_LDM.lib—for large data model projects
6. Tick the box Relative To, and select FSLESL_LOC next to the box. See Figure 1-9. Click OK.
7. Now, you will see the library added in the box. See Figure 1-11.

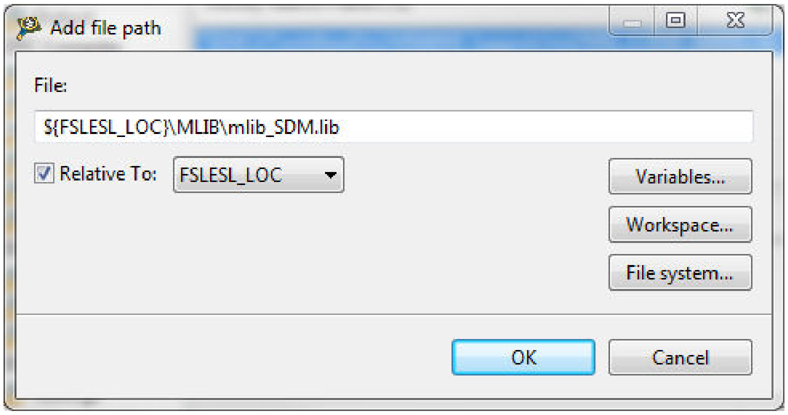

Figure 1-10. Library file inclusion

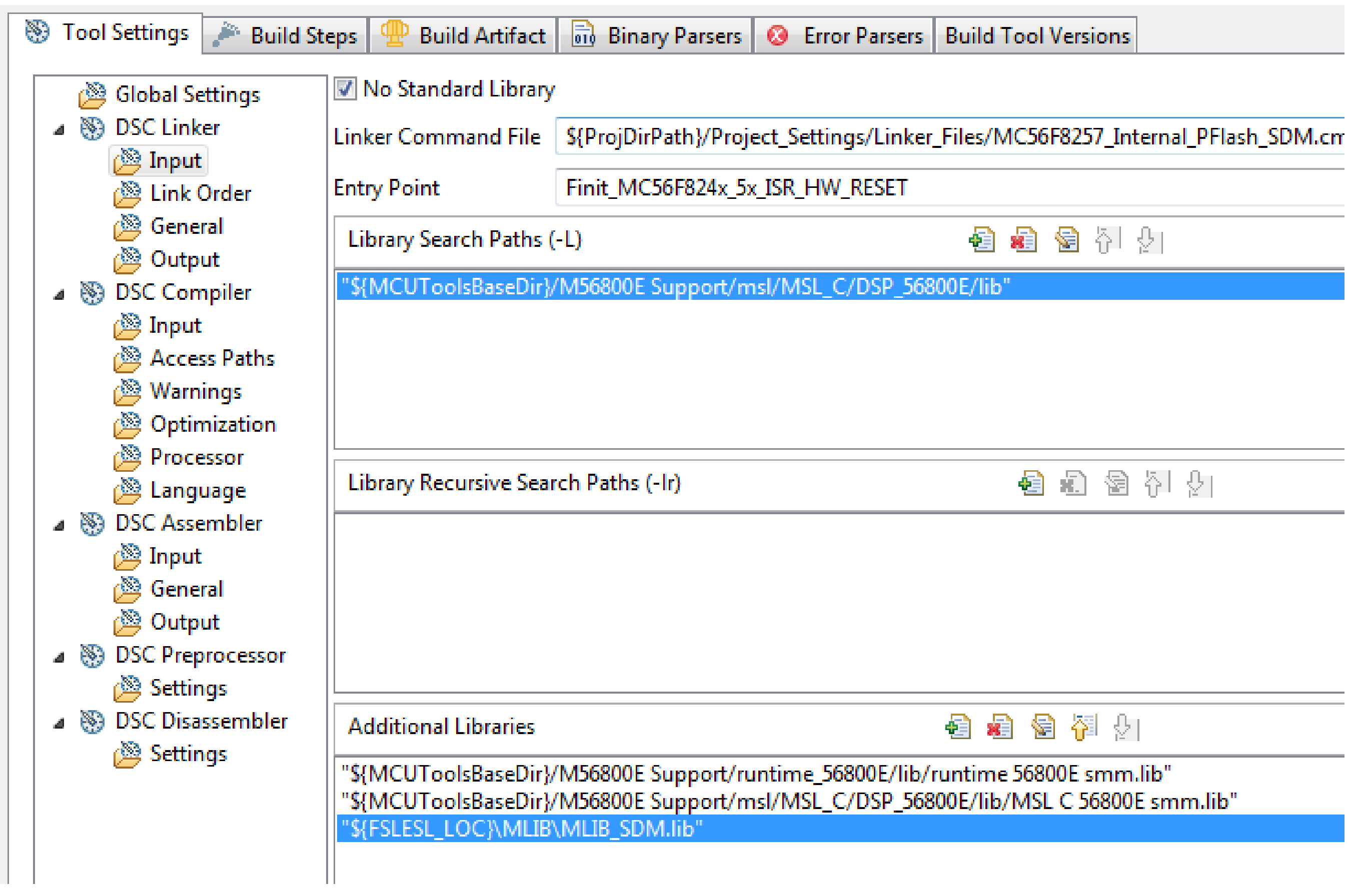

Figure 1-11. Linker setting

8. In the tree under the DSC Compiler node, click Access Paths.
9. In the Search User Paths dialog (#include "…"), click the 'Add…' icon, and a dialog will appear.
10. Look for the FSLESL_LOC variable by clicking Variables…, and then finish the path in the box to be: ${FSLESL_LOC}\MLIB\include.
11. Tick the box Relative To, and select FSLESL_LOC next to the box. See Figure 1-12. Click OK.
12. Now you will see the path added in the box. See Figure 1-13. Click OK.

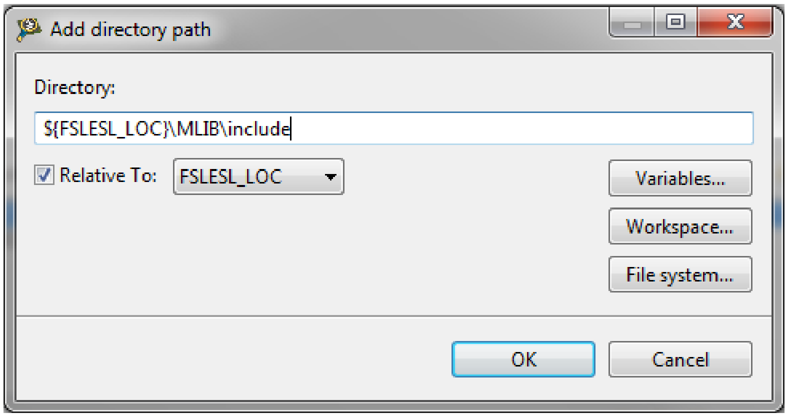

Figure 1-12. Library include path addition

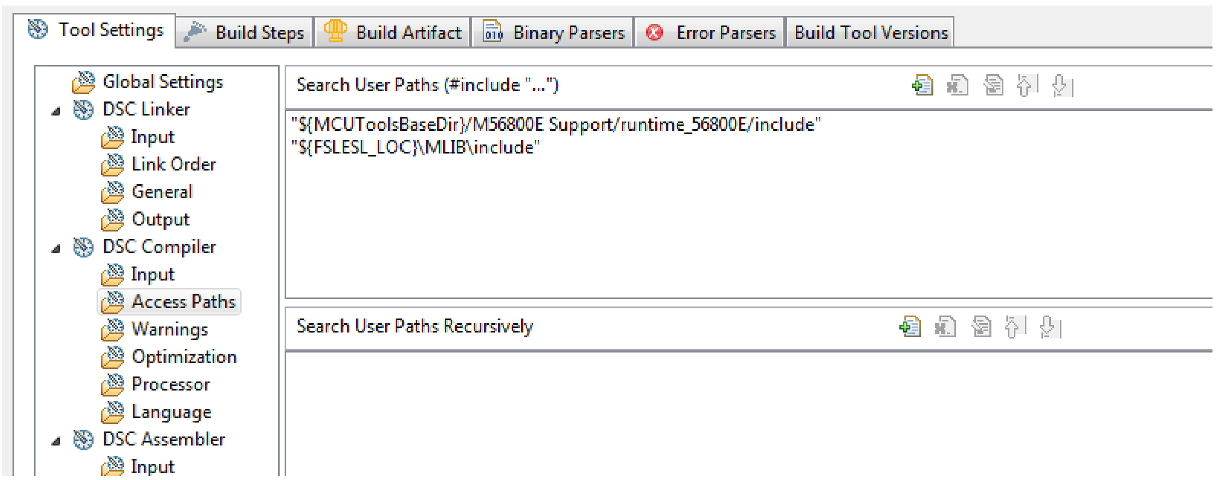

Figure 1-13. Compiler setting

The final step is typing the #include syntax into the code. Include the library into the *main.c* file. In the left-hand dialog, open the Sources folder of the project, and double-click the *main.c* file. After the *main.c* file opens up, include the following line into the #include section:

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

When you click the Build icon (hammer), 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:

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

Equation 1. 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.

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

Table 2-1. 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).

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)
```

2.1.3 Function use

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

```
#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);
}
```

2.2 MLIB_AbsSat

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

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

Equation 2. 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 2-2. 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:

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

Equation 3. 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.

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

Table 2-3. 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 <-1 ; 1).
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 <-1 ; 1).

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)
```

2.3.3 Function use

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

```
#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);
}
```

2.4 MLIB_AddSat

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

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

Equation 4. 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 2-4. 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:

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

Equation 5. 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.

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

Table 2-5. 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).

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)
```

2.5.3 Function use

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

```
#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);
}
```

2.6 MLIB_Add4Sat

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

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

Equation 6. 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 2-6. 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).
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 2-7. 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).

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

Table 2-8. 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).
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).

2.8.2 Declaration

The available MLIB_Conv functions have the following declarations:

```
frac16_t MLIB_Conv_F16l(frac32_t f32Val)
frac32_t MLIB_Conv_F32s(frac16_t f16Val)
```

2.8.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Result;
static frac16_t f16Val;

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

   /* f32Result = (frac32_t)f16Val << 16 */
   f32Result = MLIB_Conv_F32s(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:

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

Equation 7. 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).

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

Table 2-9. 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).
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 <-1 ; 1).

Table continues on the next page...

Table 2-9. Function versions (continued)

Function name	Input type		Result type	Description
	Num.	Denom.		
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 <-1 ; 1).
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 <-1 ; 1).
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 <-1 ; 1).

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)
```

2.9.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Num, f32Result;
static frac16_t f16Denom;

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

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

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:

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

Equation 8. 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 <-1 ; 1).

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

Table 2-10. 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 <-1 ; 1).

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:

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

Equation 9. 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 2-11. 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:

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

Equation 10. 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 2-12. 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).

Table continues on the next page...

Table 2-12. Function versions (continued)

Function name	Input type		Result type	Description
	Num.	Denom.		
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:

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

Equation 11. 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 2-13. 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:

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

Equation 12. 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.

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

Table 2-14. 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 <-1 ; 1).

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)
```

2.14.3 Function use

The use of the MLIB_Mac 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 = f32Accum + f16Mult1 * f16Mult2 */
  f32Result = MLIB_Mac_F32lss(f32Accum, f16Mult1, f16Mult2);
}
```

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:

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

Equation 13. 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 2-15. 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:

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

Equation 14. 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 2-16. 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 <-1 ; 1).

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:

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

Equation 15. 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 2-17. 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:

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

Equation 16. 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.

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

Table 2-18. 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).

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)
```

2.18.3 Function use

The use of the MLIB_Mac4 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(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);
}
```

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:

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

Equation 17. 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 2-19. 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:

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

Equation 18. 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 2-20. 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:

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

Equation 19. 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 2-21. 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_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:

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

Equation 20. 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.

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

Table 2-22. 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 a 32-bit accumulator. The output may be out of the range <-1 ; 1).

2.22.2 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)
```

2.22.3 Function use

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

```
#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);
}
```

2.23 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:

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

Equation 21. 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_MsuSat function are shown in the following table.

Table 2-23. 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).

2.23.2 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)
```

2.23.3 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);
}
```

2.24 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:

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

Equation 22. 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_MsuRnd function are shown in the following table.

Table 2-24. 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], 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 <-1 ; 1).

2.24.2 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)
```

2.24.3 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);
}
```

2.25 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:

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

Equation 23. 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_MsuRndSat function are shown in the following table.

Table 2-25. 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).

2.25.2 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)
```

2.25.3 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);
}
```

2.26 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:

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

Equation 24. Algorithm formula

2.26.1 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_Msu4 function are shown in the following table.

Table 2-26. 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).

2.26.2 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)
```

2.26.3 Function use

The use of the MLIB_Msu4 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(0.2);          /* 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);
}
```

2.27 MLIB_Msu4Sat

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

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

Equation 25. 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_Msu4Sat function are shown in the following table.

Table 2-27. 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).

2.27.2 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)
```

2.27.3 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);
}
```

2.28 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:

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

Equation 26. 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.

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

Table 2-28. 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).

2.28.2 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)
```

2.28.3 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);
}
```

2.29 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:

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

Equation 27. 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_Msu4RndSat function are shown in the following table.

Table 2-29. 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).

2.29.2 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)
```

2.29.3 Function use

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

```
#include "mlib.h"

static frac32_t ;
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);
}
```

2.30 MLIB_Mul

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

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

Equation 28. Algorithm formula

2.30.1 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_Mul function are shown in the following table:

Table 2-30. 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 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).

2.30.2 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)
```

2.30.3 Function use

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

```
#include "mlib.h"

static frac32_t f32Result;
static frac16_t f32Mult1, 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);
}
```

2.31 MLIB_MulSat

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

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

Equation 29. Algorithm formula

2.31.1 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 2-31. 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).
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).

2.31.2 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)
```

2.31.3 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);
}
```

2.32 MLIB_MulNeg

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

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

Equation 30. Algorithm formula

2.32.1 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_MulNeg function are shown in the following table.

Table 2-32. 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).

Table continues on the next page...

Table 2-32. Function versions (continued)

Function name	Input type		Result type	Description
	Mult. 1	Mult. 2		
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).

2.32.2 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)
```

2.32.3 Function use

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

```
#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);
}
```

2.33 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:

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

Equation 31. Algorithm formula

2.33.1 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 2-33. 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).

2.33.2 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)
```

2.33.3 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);
}
```

2.34 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:

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

Equation 32. 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 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 2-34. 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).

2.34.2 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)
```

2.34.3 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);
}
```

2.35 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:

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

Equation 33. 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.
- 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 2-35. 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).

Table continues on the next page...

Table 2-35. Function versions (continued)

Function name	Input type		Result type	Description
	Mult. 1	Mult. 2		
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 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).

2.35.2 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)
```

2.35.3 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);
}
```

2.36 MLIB_Neg

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

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

Equation 34. Algorithm formula

2.36.1 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_Neg function are shown in the following table:

Table 2-36. 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).

2.36.2 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)
```

2.36.3 Function use

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

```
#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);
}
```

2.37 MLIB_NegSat

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

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

Equation 35. Algorithm formula

2.37.1 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 2-37. 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).

2.37.2 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)
```

2.37.3 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);
}
```

2.38 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:

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

Equation 36. Algorithm formula

2.38.1 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 2-38. 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.

2.38.2 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)
```

2.38.3 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);
}
```

2.39 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:

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

Equation 37. Algorithm formula

2.39.1 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 2-39. 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.

2.39.2 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)
```

2.39.3 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);
}
```

2.40 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:

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

Equation 38. Algorithm formula

2.40.1 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 2-40. 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).

2.40.2 Declaration

The available MLIB_Rnd functions have the following declarations:

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

2.40.3 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);
}
```

2.41 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:

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

Equation 39. Algorithm formula

2.41.1 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 2-41. 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).

2.41.2 Declaration

The available MLIB_RndSat functions have the following declarations:

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

2.41.3 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);
}
```

2.42 MLIB_Sat

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

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

Equation 40. Algorithm formula

2.42.1 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 2-42. 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).

2.42.2 Declaration

The available MLIB_Sat functions have the following declarations:

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

2.42.3 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);
}
```

2.43 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:

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

Equation 41. 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 overflow.

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

Table 2-43. 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).

2.43.2 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)
```

2.43.3 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);
}
```

2.44 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:

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

Equation 42. Algorithm formula

2.44.1 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 2-44. 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).

2.44.2 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)
```

2.44.3 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);
}
```

2.45 MLIB_Sh1R

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

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

Equation 43. Algorithm formula

2.45.1 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 2-45. 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).

2.45.2 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)
```

2.45.3 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);
}
```

2.46 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:

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

Equation 44. 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_ShL function are shown in the following table.

Table 2-46. 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).

2.46.2 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)
```

2.46.3 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);
}
```

2.47 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:

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

Equation 45. 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_ShLSat function are shown in the following table.

Table 2-47. 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).

2.47.2 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)
```

2.47.3 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);
}
```

2.48 MLIB_ShR

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

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

Equation 46. Algorithm formula

2.48.1 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 2-48. 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).

2.48.2 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)
```

2.48.3 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);
}
```

2.49 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:

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

Equation 47. 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_ShLBi function are shown in the following table.

Table 2-49. 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).

Table continues on the next page...

Table 2-49. Function versions (continued)

Function name	Input type		Result type	Description
	Value	Shift		
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).

2.49.2 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)
```

2.49.3 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);
}
```

2.50 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:

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

Equation 48. 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_ShLBiSat function are shown in the following table.

Table 2-50. 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).

2.50.2 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)
```

2.50.3 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);
}
```

2.51 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:

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

Equation 49. 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 <-1 ; 1). The result may overflow.

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

Table 2-51. 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).

2.51.2 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)
```

2.51.3 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);
}
```

2.52 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:

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

Equation 50. 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 saturate.

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

Table 2-52. 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).

2.52.2 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)
```

2.52.3 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);
}
```

2.53 MLIB_Sign

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

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

Equation 51. 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 available versions of the MLIB_Sign function are shown in the following table.

Table 2-53. 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).

2.53.2 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)
```

2.53.3 Function use

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

```
#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);
}
```

2.54 MLIB_Sub

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

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

Equation 52. 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 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.

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

Table 2-54. 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).
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 <-1 ; 1).
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 <-1 ; 1).

2.54.2 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)
```

2.54.3 Function use

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

```
#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);
}
```

2.55 MLIB_SubSat

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

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

Equation 53. Algorithm formula

2.55.1 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 2-55. 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).

2.55.2 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)
```

2.55.3 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);
}
```

2.56 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:

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

Equation 54. 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 overflow.

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

Table 2-56. 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).

2.56.2 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)
```

2.56.3 Function use

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

```
#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);
}
```

2.57 MLIB_Sub4Sat

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

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

Equation 55. 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 saturate.

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

Table 2-57. 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).

2.57.2 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)
```

2.57.3 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 A-1. 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 int8_t;
```

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

Table A-2. Data storage

Value	7	6	5	4	3	2	1	0
	Integer							
255	1	1	1	1	1	1	1	1
	F				F			
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 A-3. Data storage

Value	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	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 A-4. 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 A-5. 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			
-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 A-6. 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 A-7. Data storage

	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 A-8. 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 A-9. 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

Table continues on the next page...

Table A-9. Data storage (continued)

	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 A-10. Data storage

	31	24	23	16	15	8	7	0
Value	S	Fractional						
0.9999999995	7	F	F	F	F	F	F	F
-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 A-11. Data storage

Value	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	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 A-12. 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 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.14 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.15 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.16 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.17 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.18 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.19 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 */
}
```

Document Number DSP56800EMLIBUG
Revision 2, 10/2015