## Contents

### Chapter 1 Library

1.1 Introduction ................................................................. 8  
  1.1.1 Overview ...................................................................... 8  
  1.1.2 Data types .................................................................... 8  
  1.1.3 API definition ................................................................. 8  
  1.1.4 Supported compilers .................................................... 9  
  1.1.5 Library configuration .................................................... 9  
  1.1.6 Special issues .............................................................. 9  
  1.2 Library integration into project (MCUXpresso IDE) .......... 9  
  1.3 Library integration into project (Keil µVision) ................. 13  
  1.4 Library integration into project (IAR Embedded Workbench) 20

### Chapter 2 Algorithms in detail

2.1 MLIB_Abs ........................................................................ 27  
  2.1.1 Available versions ....................................................... 27  
  2.1.2 Declaration ................................................................. 27  
  2.1.3 Function use .............................................................. 27  
  2.2 MLIB_AbsSat ................................................................. 28  
  2.2.1 Available versions ....................................................... 28  
  2.2.2 Declaration ................................................................. 28  
  2.2.3 Function use .............................................................. 28  
  2.3 MLIB_Add ................................................................. 28  
  2.3.1 Available versions ....................................................... 29  
  2.3.2 Declaration ................................................................. 29  
  2.3.3 Function use .............................................................. 29  
  2.4 MLIB_AddSat ................................................................. 30  
  2.4.1 Available versions ....................................................... 30  
  2.4.2 Declaration ................................................................. 30  
  2.4.3 Function use .............................................................. 30  
  2.5 MLIB_Add4 ................................................................. 31  
  2.5.1 Available versions ....................................................... 31  
  2.5.2 Declaration ................................................................. 31  
  2.5.3 Function use .............................................................. 31  
  2.6 MLIB_Add4Sat ............................................................... 32  
  2.6.1 Available versions ....................................................... 32  
  2.6.2 Declaration ................................................................. 32  
  2.6.3 Function use .............................................................. 33  
  2.7 MLIB_Clb ................................................................. 33  
  2.7.1 Available versions ....................................................... 33  
  2.7.2 Declaration ................................................................. 33  
  2.7.3 Function use .............................................................. 33  
  2.8 MLIB_Conv ............................................................... 34  
  2.8.1 Available versions ....................................................... 34  
  2.8.2 Declaration ................................................................. 34  
  2.8.3 Function use .............................................................. 34  
  2.9 MLIB_ConvSc ............................................................... 35  
  2.9.1 Available versions ....................................................... 35  
  2.9.2 Declaration ................................................................. 35  
  2.9.3 Function use .............................................................. 35  
  2.10 MLIB_Div ................................................................. 35
Appendix A Library types........................................................................................................... 103
A.1 bool_t.................................................................................................................................. 103
A.2 uint8_t............................................................................................................................... 103
A.3 uint16_t.............................................................................................................................. 104
A.4 uint32_t.............................................................................................................................. 105
A.5 int8_t.................................................................................................................................. 105
A.6 int16_t.................................................................................................................................. 106
A.7 int32_t.................................................................................................................................. 106
A.8 frac8_t.................................................................................................................................. 107
A.9 frac16_t.................................................................................................................................. 108
A.10 frac32_t............................................................................................................................ 108
A.11 acc16_t................................................................................................................................ 109
A.12 acc32_t................................................................................................................................ 109
A.13 FALSE............................................................................................................................... 110
A.14 TRUE................................................................................................................................. 110
A.15 FRAC8.............................................................................................................................. 111
A.16 FRAC16............................................................................................................................ 111
A.17 FRAC32............................................................................................................................ 112
A.18 ACC16.............................................................................................................................. 112
A.19 ACC32.............................................................................................................................. 112
Chapter 1
Library

1.1 Introduction

1.1.1 Overview
This user’s guide describes the Math Library (MLIB) for the family of ARM Cortex M33 core-based microcontrollers. 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:

```c
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. Input/output types

<table>
<thead>
<tr>
<th>Type</th>
<th>Output</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>frac16_t</td>
<td>F16</td>
<td>s</td>
</tr>
<tr>
<td>frac32_t</td>
<td>F32</td>
<td>l</td>
</tr>
<tr>
<td>acc32_t</td>
<td>A32</td>
<td>a</td>
</tr>
</tbody>
</table>

1.1.4 Supported compilers

MLIB for the ARM Cortex M33 core is written in C language or assembly language with C-callable interface depending on the specific function. The library is built and tested using the following compilers:

- MCUXpresso IDE
- IAR Embedded Workbench
- Keil µVision

For the MCUXpresso IDE, the library is delivered in the `mlib.a` file.
For the Kinetis Design Studio, the library is delivered in the `mlib.a` file.
For the IAR Embedded Workbench, the library is delivered in the `mlib.a` file.
For the Keil µVision, the library is delivered in the `mlib.lib` file.

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

1.1.5 Library configuration

MLIB for the ARM Cortex M33 core is written in C language or assembly language with C-callable interface depending on the specific function. Some functions from this library are inline type, which are compiled together with project using this library. The optimization level for inline function is usually defined by the specific compiler setting. It can cause an issue especially when high optimization level is set. Therefore the optimization level for all inline assembly written functions is defined by compiler pragmas using macros. The configuration header file `RTCESL_cfg.h` is located in: specific library folder\MLIB\Include. The optimization level can be changed by modifying the macro value for specific compiler. In case of any change the library functionality is not guaranteed.

Similarly as optimization level the PowerQuad DSP Coprocessor and Accelerator support can be disable or enable if it has not been done by defined symbol `RTCESL_PQ_ON` or `RTCESL_PQ_OFF` in project setting described in the PowerQuad DSP Coprocessor and Accelerator support cheaper for specific compiler.

1.1.6 Special issues

1. The equations describing the algorithms are symbolic. If there is positive 1, the number is the closest number to 1 that the resolution of the used fractional type allows. If there are maximum or minimum values mentioned, check the range allowed by the type of the particular function version.

2. The library functions that round the result (the API contains Rnd) round to nearest (half up).

3. This RTCESL requires the DSP extension for some saturation functions. If the core does not support the DSP extension feature the assembler code of the RTCESL will not be buildable. For example the core1 of the LPC55s69 has no DSP extension.

1.2 Library integration into project (MCUXpresso IDE)

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

Some LPC platforms (LPC55S6x) contain a hardware accelerator dedicated to common calculations in DSP applications. This section shows how to turn the PowerQuad (PQ) support for a function on and off.

1. In the MCUXpresso SDK project name node or in the left-hand part, click Properties or select Project > Properties from the menu. A project properties dialog appears.

2. Expand the C/C++ Build node and select Settings. See Figure 1.

3. On the right-hand side, under the MCU C Compiler node, click the Preprocessor node. See Figure 1.

4. In the right-hand part of the dialog, click the Add... icon located next to the Defined symbols (-D) title.

5. In the dialog that appears (see Figure 2), type the following:
   - RTCESL_PQ_ON—to turn the PowerQuad support on
   - RTCESL_PQ_OFF—to turn the PowerQuad support off

   If neither of these two defines is defined, the hardware division and square root support is turned off by default.

6. Click OK in the dialog.

7. Click OK in the main dialog.
8. Ensure the PowerQuad module to be clocked by calling function `RTCESL_PQ_Init();` prior to the first function using PQ module calling.

See the device reference manual to verify whether the device contains the PowerQuad DSP Coprocessor and Accelerator support.

Adding RTCESL component to project

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

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

![Figure 3. MCUXpresso IDE - importing the SDK package to MCUXpresso IDE](image)

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

Find the Middleware tab in the Components part of the window and click on the checkbox to be the rtcesl component ticked. Last step is to click the Finish button and wait for project creating with all RTCESL libraries and include paths.
Type the `#include` syntax into the code where you want to call the library functions. In the left-hand dialog, open the required `.c` file. After the file opens, include the following line into the `#include` section:

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

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

### 1.3 Library integration into project (Keil µVision)

This section provides a step-by-step guide on how to quickly and easily include MLIB into an empty project or any MCUXpresso SDK example or demo application projects using Keil µVision. This example uses the default installation path (C:\NXP\RTCESL\CM33_RTCESL_4.7_KEIL). If you have a different installation path, use that path instead. If any MCUXpresso SDK project is intended to use (for example hello_world project) go to [Linking the files into the project](#) chapter otherwise read next chapter.

**NXP pack installation for new project (without MCUXpresso SDK)**

This example uses the NXP LPC55s69 part, and the default installation path (C:\NXP\RTCESL\CM33_RTCESL_4.7_KEIL) is supposed. If the compiler has never been used to create any NXP MCU-based projects before, check whether the NXP MCU pack for the particular device is installed. Follow these steps:

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

New project (without MCUXpresso SDK)

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

1. Launch Keil µVision.
2. In the main menu, select Project > New µVision Project…, and the Create New Project dialog appears.
3. Navigate to the folder where you want to create the project, for example C:\KeilProjects\MyProject01. Type the name of the project, for example MyProject01. Click Save. See Figure 8.
4. In the next dialog, select the Software Packs in the very first box.
5. Type " into the Search box, so that the device list is reduced to the devices.
6. Expand the node.
7. Click the LPC55s69 node, and then click OK. See Figure 9.
8. In the next dialog, expand the Device node, and tick the box next to the Startup node. See Figure 10.

9. Expand the CMSIS node, and tick the box next to the CORE node.

10. Click OK, and a new project is created. The new project is now visible in the left-hand part of Keil µVision. See Figure 11.
11. In the main menu, go to Project > Options for Target 'Target1'..., and a dialog appears.
12. Select the Target tab.
13. Select Not Used in the Floating Point Hardware option. See Figure 11.

**Figure 11. Project**

PowerQuad DSP Coprocessor and Accelerator support

Some LPC platforms (LPC55S6x) contain a hardware accelerator dedicated to common calculations in DSP applications. This section shows how to turn the PowerQuad (PQ) support for a function on and off.

1. In the main menu, go to Project > Options for Target 'Target1'..., and a dialog appears.
2. Select the C/C++ tab. See Figure 13.
3. In the Include Preprocessor Symbols text box, type the following:
   - RTCESL_PQ_ON—to turn the hardware division and square root support on.
   - RTCESL_PQ_OFF—to turn the hardware division and square root support off.

If neither of these two defines is defined, the hardware division and square root support is turned off by default.
4. Click OK in the main dialog.

5. Ensure the PowerQuad module to be clocked by calling function RTCESL_PQ_Init(); prior to the first function using PQ module calling.

See the device reference manual to verify whether the device contains the PowerQuad DSP Coprocessor and Accelerator support.

Linking the files into the project

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

1. Right-click the Target 1 node in the left-hand part of the Project tree, and select Add Group… from the menu. A new group with the name New Group is added.

2. Click the newly created group, and press F2 to rename it to RTCESL.

3. Right-click the RTCESL node, and select Add Existing Files to Group ‘RTCESL’… from the menu.

4. Navigate into the library installation folder C:\NXP\RTCESL\CM33_RTCESL_4.7_KEIL\MLIB\Include, and select the mlib.h file. If the file does not appear, set the Files of type filter to Text file. Click Add. See Figure 14.
5. Navigate to the parent folder C:\NXP\RTCESL\CM33_RTCESL_4.7_KEIL\MLIB, and select the `mlib.lib` file. If the file does not appear, set the Files of type filter to Library file. Click Add. See Figure 15.

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

![Figure 14. Adding .h files dialog](image)

![Figure 15. Adding .lib files dialog](image)
Library path setup

The following steps show the inclusion of all dependent modules.

1. In the main menu, go to Project > Options for Target 'Target1'…, and a dialog appears.
2. Select the C/C++ tab. See Figure 17.
3. In the Include Paths text box, type the following path (if there are more paths, they must be separated by ‘;’) or add it by clicking the … button next to the text box:
   - “C:\NXP\RTCESL\CM33_RTCESL_4.7_KEIL\MLIB\Include”
4. Click OK.
5. Click OK in the main dialog.

Figure 16. Project workspace

Figure 17. Library path addition
Type the `#include` syntax into the code. Include the library into a source file. In the new project, it is necessary to create a source file:

1. Right-click the Source Group 1 node, and Add New Item to Group ‘Source Group 1’… from the menu.
2. Select the C File (.c) option, and type a name of the file into the Name box, for example ‘main.c’. See Figure 18.
3. Click Add, and a new source file is created and opened up.
4. In the opened source file, include the following line into the `#include` section, and create a main function:

```c
#include "mlib.h"

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

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

### 1.4 Library integration into project (IAR Embedded Workbench)

This section provides a step-by-step guide on how to quickly and easily include the MLIB into an empty project or any MCUXpresso SDK example or demo application projects using IAR Embedded Workbench. This example uses the default installation path (C:\NXP\RTCESL\CM33_RTCESL_4.7_IAR). If you have a different installation path, use that path instead. If any MCUXpresso SDK project is intended to use (for example hello_world project) go to Linking the files into the project chapter otherwise read next chapter.
New project (without MCUXpresso SDK)

This example uses the NXP LPC55S69 part, and the default installation path (C:\NXP\RTCESL\CM33_RTCESL_4.7_IAR) is supposed. To start working on an application, create a new project. If the project already exists and is opened, skip to the next section. Perform these steps to create a new project:

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

![Create New Project dialog](image)

Figure 19. Create New Project dialog

3. Expand the C node in the tree, and select the "main" node. Click OK.

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

![New project](image)

Figure 20. New project

5. In the main menu, go to Project > Options…, and a dialog appears.

6. In the Target tab, select the Device option, and click the button next to the dialog to select the MCU. In this example, select NXP > LPC55S69 > NXP LPC55S69_core0. Select None in the FPU option. The DSP instructions group is required please check the DSP Extensions checkbox if not checked. Click OK. See Figure 21.

![Create New Project dialog](image)

Figure 21. Create New Project dialog
PowerQuad DSP Coprocessor and Accelerator support

Some LPC platforms (LPC55S6x) contain a hardware accelerator dedicated to common calculations in DSP applications. Only functions running faster through the PowerQuad module than the core itself are supported and targeted to be calculated by the PowerQuad module. This section shows how to turn the PowerQuad (PQ) support for a function on and off.

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

If neither of these two defines is defined, the hardware division and square root support is turned off by default.
5. Click OK in the main dialog.

6. Ensure the PowerQuad module to be clocked by calling function RTCESL_PQ_Init(); prior to the first function using PQ module calling.

See the device reference manual to verify whether the device contains the PowerQuad DSP Coprocessor and Accelerator support.

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

1. In the main menu, go to Tools > Configure Custom Argument Variables…, and a dialog appears.

2. Click the New Group button, and another dialog appears. In this dialog, type the name of the group PATH, and click OK. See Figure 23.
3. Click on the newly created group, and click the Add Variable button. A dialog appears.

4. Type this name: RTCESL_LOC

5. To set up the value, look for the library by clicking the ‘...' button, or just type the installation path into the box: C:\NXP\RTCESL\CM33_RTCESL_4.7_IAR. Click OK.

6. In the main dialog, click OK. See Figure 24.

---

Linking the files into the project

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

1. Go to the main menu Project > Add Group...

2. Type RTCESL, and click OK.

3. Click on the newly created node RTCESL, go to Project > Add Group..., and create a MLIB subgroup.

4. Click on the newly created node MLIB, and go to the main menu Project > Add Files... See Figure 26.

5. Navigate into the library installation folder C:\NXP\RTCESL\CM33_RTCESL_4.7_IAR\MLIB\Include, and select the mlib.h file. (If the file does not appear, set the file-type filter to Source Files.) Click Open. See Figure 25.
6. Navigate into the library installation folder C:\NXP\RTCESL\CM33_RTCESL_4.7_IAR\MLIB, and select the `mlib.a` file. If the file does not appear, set the file-type filter to Library / Object files. Click Open.

![Add Files dialog](image1.png)

**Figure 25. Add Files dialog**

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

![Project workspace](image2.png)

**Figure 26. Project workspace**

**Library path setup**

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

![Library path addition](image3.png)

**Figure 27. Library path addition**
Type the \#include syntax into the code. Include the library included into the *main.c* file. In the workspace tree, double-click the *main.c* file. After the *main.c* file opens up, include the following line into the \#include section:

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

When you click the Make icon, the project will be compiled without errors.
Chapter 2
Algorithms in detail

2.1 MLIB_Abs

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

\[ \text{MLIB}_\text{Abs}(x) = |x| \]

Table 2. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Abs_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Absolute value of a 16-bit fractional value. The output is within the range (-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_Abs_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>Absolute value of a 32-bit fractional value. The output is within the range (-1 ; 1).</td>
</tr>
</tbody>
</table>

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 examples:

```
#include "mlib.h"

static frac32_t f32Result;
static frac32_t f32Val;

void main(void)
{
    f32Val = FRAC32(-0.354); /* f32Val = -0.354 */
    /* f32Result = |f32Val| */
```

NXP Semiconductors
MLIB User's Guide, Rev. 5, 01 November 2021
2.2 MLIB_AbsSat

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

\[ \text{MLIB_AbsSat}(x) = |x| \]

Figure 29. Algorithm formula

2.2.1 Available versions

This function is available in the following versions:

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

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

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_AbsSat_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Absolute value of a 16-bit fractional value. The output is within the range (0 ; 1).</td>
</tr>
<tr>
<td>MLIB_AbsSat_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>Absolute value of a 32-bit fractional value. The output is within the range (0 ; 1).</td>
</tr>
</tbody>
</table>

2.2.2 Declaration

The available MLIB_AbsSat functions have the following declarations:

\[
\begin{align*}
\text{frac16_t} & \quad \text{MLIB_AbsSat}_\text{F16}(\text{frac16_t} \ f16Val) \\
\text{frac32_t} & \quad \text{MLIB_AbsSat}_\text{F32}(\text{frac32_t} \ f32Val)
\end{align*}
\]

2.2.3 Function use

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

```c
#include "mlib.h"

static frac16_t f16Val, f16Result;

void main(void)
{
    f16Val = FRAC16(-0.835);        /* f16Val = -0.835 */
    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:
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>
<thead>
<tr>
<th>Function name</th>
<th>Addend 1</th>
<th>Addend 2</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MLIB_Add_F16</code></td>
<td><code>frac16_t</code></td>
<td><code>frac16_t</code></td>
<td><code>frac16_t</code></td>
<td>Addition of two 16-bit fractional addends. The output is within the range $&lt;-1 ; 1)$.</td>
</tr>
<tr>
<td><code>MLIB_Add_F32</code></td>
<td><code>frac32_t</code></td>
<td><code>frac32_t</code></td>
<td><code>frac32_t</code></td>
<td>Addition of two 32-bit fractional addends. The output is within the range $&lt;-1 ; 1)$.</td>
</tr>
<tr>
<td><code>MLIB_Add_A32ss</code></td>
<td><code>frac16_t</code></td>
<td><code>frac16_t</code></td>
<td><code>acc32_t</code></td>
<td>Addition of two 16-bit fractional addends; the result is a 32-bit accumulator. The output may be out of the range $&lt;-2 ; 2&gt;$.</td>
</tr>
<tr>
<td><code>MLIB_Add_A32as</code></td>
<td><code>acc32_t</code></td>
<td><code>frac16_t</code></td>
<td><code>acc32_t</code></td>
<td>A 16-bit fractional addend is added to a 32-bit accumulator. The output may be out of the range $&lt;-2 ; 2&gt;$.</td>
</tr>
</tbody>
</table>

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 examples:

```
#include "mlib.h"

static acc32_t a32Result;
static frac16_t f16Add1, f16Add2;

void main(void)
{
    f16Add1 = FRAC16(-0.8);   /* f16Add1 = -0.8 */
```
2.4 MLIB_AddSat

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

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

Figure 31. Algorithm formula

2.4.1 Available versions

This function is available in the following versions:

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

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

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_AddSat_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Addition of two 16-bit fractional addends. The output is within the range &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_AddSat_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>Addition of two 32-bit fractional addends. The output is within the range &lt;-1 ; 1).</td>
</tr>
</tbody>
</table>

2.4.2 Declaration

The available MLIB_AddSat functions have the following declarations:

```c
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:

```c
#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 = MLIB_Add_A32ss(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:

\[ \text{MLIB\_Add4}(a, b, c, d) = a + b + c + d \]

Figure 32. Algorithm formula

2.5.1 Available versions

This function is available in the following versions:

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

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

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Add4_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Addition of four 16-bit fractional addends. The output is within the range (-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_Add4_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>Addition of four 32-bit fractional addends. The output is within the range (-1 ; 1).</td>
</tr>
</tbody>
</table>

2.5.2 Declaration

The available MLIB_Add4 functions have the following declarations:

\[
\begin{align*}
\text{frac16_t} & \quad \text{MLIB\_Add4\_F16}(\text{frac16_t} \; f16Add1, \; \text{frac16_t} \; f16Add2, \; \text{frac16_t} \; f16Add3, \; \text{frac16_t} \; f16Add4) \\
\text{frac32_t} & \quad \text{MLIB\_Add4\_F32}(\text{frac32_t} \; f32Add1, \; \text{frac32_t} \; f32Add2, \; \text{frac32_t} \; f32Add3, \; \text{frac32_t} \; f32Add4)
\end{align*}
\]

2.5.3 Function use

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

**Fixed-point version:**

```c
#include "mlib.h"

static frac32_t f32Result;
static frac32_t f32Add1, f32Add2, f32Add3, f32Add4;
```
```c
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:

\[
MLIB\_Add4Sat(a, b, c, d) = \begin{cases} 
1, & a + b + c + d > 1 \\
0, & -1 < a + b + c + d \leq 1 \\
(a + b + c + d), & \text{else}
\end{cases}
\]

**Figure 33. Algorithm formula**

#### 2.6.1 Available versions

This function is available in the following versions:

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

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

**Table 7. Function versions**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Add. 1</td>
<td>Add. 2</td>
<td>Add. 3</td>
</tr>
<tr>
<td>MLIB_Add4Sat_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td>MLIB_Add4Sat_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>frac32_t</td>
</tr>
</tbody>
</table>

#### 2.6.2 Declaration

The available MLIB_Add4Sat functions have the following declarations:

```c
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:

```c
#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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Clb_U16s</td>
<td>frac16_t</td>
<td>uint16_t</td>
<td>Counts the leading bits of a 16-bit fractional value. The output is within the range &lt;0; 15&gt;.</td>
</tr>
<tr>
<td>MLIB_Clb_U16l</td>
<td>frac32_t</td>
<td>uint16_t</td>
<td>Counts the leading bits of a 32-bit fractional value. The output is within the range &lt;0; 31&gt;.</td>
</tr>
</tbody>
</table>

2.7.2 Declaration

The available MLIB_Clb functions have the following declarations:

```c
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:

```c
#include "mlib.h"

static uint16_t u16Result;
```
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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Conv_F16l</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>Conversion of a 32-bit fractional value to a 16-bit fractional value. The output is within the range $(-1 ; 1)$.</td>
</tr>
<tr>
<td>MLIB_Conv_F32s</td>
<td>frac16_t</td>
<td>frac32_t</td>
<td>Conversion of a 16-bit fractional value to a 32-bit fractional value. The output is within the range $(-1 ; 1)$.</td>
</tr>
</tbody>
</table>

2.8.2 Declaration

The available MLIB_Conv functions have the following declarations:

```c
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 examples:

**Fixed-point version:**

```c
#include "mlib.h"

static frac32_t f32Result;
static frac16_t f16Val;

void main(void)
{
    f16Val = FRAC16(-0.5); /* f16Val = -0.5 */
    /* f32Result = (frac32_t)f16Val << 16 */
```
2.9 MLIB_ConvSc

The MLIB_ConvSc functions return the input value converted to the output type using a scale coefficient.

2.9.1 Available versions

The function is available in the following versions:

- Fractional output - the output is the fractional portion of the result. The input value is divided by the scale to be within the range \(-1 ; 1\). The result may be saturated.
- Accumulator output - the output is the accumulator type, where the result may be out of the range \(-1 ; 1\).
- Floating-point output - the output is a floating-point number. The input value is multiplied by the scale to transform the fractional value into the real floating-point number. The result is within the full range.

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

2.9.2 Declaration

The available MLIB_ConvSc functions have the following declarations:

2.9.3 Function use

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

Fixed-point version:

```c
#include "mlib.h"

static frac32_t f32Result;
static float_t fltVal, fltScale;

void main(void)
{
    fltVal = -0.25F;        /* fltVal = -0.25 */
    fltScale = 10.0F;        /* Scale is 10 */
    /* f32Result = (frac32_t)((fltVal / fltScale) * 2 ^ 31) */
    f32Result = MLIB_ConvSc_F32ff(fltVal, fltScale);
}
```

2.10 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:

Figure 34. 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 function is only defined for $|\text{nominator}| < |\text{denominator}|$. 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>
<thead>
<tr>
<th>Function name</th>
<th>Num. type</th>
<th>Denom. type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Div_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Division of a 16-bit fractional numerator and denominator. The output is within the range $&lt;-1 ; 1)$.</td>
</tr>
<tr>
<td>MLIB_Div_F16ls</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>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 $&lt;-1 ; 1)$.</td>
</tr>
<tr>
<td>MLIB_Div_F16ll</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>Division of a 32-bit fractional numerator and denominator; the output is a 16-bit fractional result. The output is within the range $&lt;-1 ; 1)$.</td>
</tr>
<tr>
<td>MLIB_Div_F32ls</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>frac32_t</td>
<td>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 $&lt;-1 ; 1)$.</td>
</tr>
<tr>
<td>MLIB_Div_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>Division of a 32-bit fractional numerator and denominator. The output is within the range $&lt;-1 ; 1)$.</td>
</tr>
<tr>
<td>MLIB_Div_A32ss</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>acc32_t</td>
<td>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 $&lt;-65536 ; 65536)$.</td>
</tr>
<tr>
<td>MLIB_Div_A32ls</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>acc32_t</td>
<td>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 $&lt;-65536 ; 65536)$.</td>
</tr>
<tr>
<td>MLIB_Div_A32ll</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>acc32_t</td>
<td>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 $&lt;-65536 ; 65536)$.</td>
</tr>
<tr>
<td>MLIB_Div_A32as</td>
<td>acc32_t</td>
<td>frac16_t</td>
<td>acc32_t</td>
<td>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 $&lt;-65536 ; 65536)$.</td>
</tr>
</tbody>
</table>

2.10.2 Declaration

The available MLIB_Div functions have the following declarations:

```c
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)
```
2.10.3 Function use

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

### Fixed-point version:

```c
#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);
}
```

2.11 MLIB_DivSat

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

\[
MLIB\_DivSat(a, b) = \begin{cases} 
    \max, & \frac{a}{b} > \max \land a \geq 0 \land b = 0 \\
    \min, & \frac{a}{b} < \min \land a < 0 \land b = 0 \\
    \frac{a}{b}, & \text{else}
\end{cases}
\]

Figure 35. 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 \(-1 ; 1\). The result may saturate.
- Accumulator output - the output is the accumulator type, where the result may be out of the range \(-65536 ; 65536\).

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

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_DivSat_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Division of a 16-bit fractional numerator and denominator. The output is within the range (-1 ; 1).</td>
</tr>
</tbody>
</table>
Table 12. Function versions (continued)

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_DivSat_F16ls</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>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 &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_DivSat_F16ll</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>Division of a 32-bit fractional numerator and denominator; the output is a 16-bit fractional result. The output is within the range &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_DivSat_F32ls</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>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 &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_DivSat_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>Division of a 32-bit fractional numerator and denominator. The output is within the range &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_DivSat_A32as</td>
<td>acc32_t</td>
<td>frac16_t</td>
<td>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 &lt;-65536 ; 65536).</td>
</tr>
</tbody>
</table>

2.11.2 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)
```

2.11.3 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);
}
2.12 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:

\[
\text{MLIB}_\text{Div1Q}(a, b) = \begin{cases} 
\text{max}, & a \geq 0 \land b = 0 \\
\frac{a}{b}, & a \geq 0 \land b > 0
\end{cases}
\]

Figure 36. 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 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Div1Q_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Division of a non-negative 16-bit fractional numerator and denominator. The output is within the range (&lt;0 ; 1)).</td>
</tr>
<tr>
<td>MLIB_Div1Q_F16ls</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>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 (&lt;0 ; 1)).</td>
</tr>
<tr>
<td>MLIB_Div1Q_F16ll</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>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 (&lt;0 ; 1)).</td>
</tr>
<tr>
<td>MLIB_Div1Q_F32ls</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>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 (&lt;0 ; 1)).</td>
</tr>
<tr>
<td>MLIB_Div1Q_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>Division of a non-negative 32-bit fractional numerator and denominator. The output is within the range (&lt;0 ; 1)).</td>
</tr>
<tr>
<td>MLIB_Div1Q_A32ss</td>
<td>frac16_t</td>
<td>acc32_t</td>
<td>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.</td>
</tr>
<tr>
<td>MLIB_Div1Q_A32ls</td>
<td>frac32_t</td>
<td>acc32_t</td>
<td>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.</td>
</tr>
<tr>
<td>MLIB_Div1Q_A32ll</td>
<td>frac32_t</td>
<td>acc32_t</td>
<td>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.</td>
</tr>
<tr>
<td>MLIB_Div1Q_A32as</td>
<td>acc32_t</td>
<td>frac16_t</td>
<td>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.</td>
</tr>
</tbody>
</table>
2.12.2 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)

2.12.3 Function use

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

```c
#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 = MLIB_Div1Q_F32(f32Num, f32Denom);
}
```

2.13 MLIB_Div1QSat

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

\[
\text{MLIB Div1QSat}(a, b) = \begin{cases} 
\max, & \frac{a}{b} > \max \land a \geq 0 \land b \geq 0 \\
\frac{a}{b}, & a \geq 0 \land b > 0 
\end{cases}
\]

Figure 37. Algorithm formula

2.13.1 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 14. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Div1QSat_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Division of a non-negative 16-bit fractional numerator and denominator. The output is within the range $&lt;0 ; 1)$.</td>
</tr>
<tr>
<td>MLIB_Div1QSat_F16ls</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>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 $&lt;0 ; 1)$.</td>
</tr>
<tr>
<td>MLIB_Div1QSat_F16ll</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>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 $&lt;0 ; 1)$.</td>
</tr>
<tr>
<td>MLIB_Div1QSat_F32ls</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>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 $&lt;0 ; 1)$.</td>
</tr>
<tr>
<td>MLIB_Div1QSat_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>Division of a non-negative 32-bit fractional numerator and denominator. The output is within the range $&lt;0 ; 1)$.</td>
</tr>
<tr>
<td>MLIB_Div1QSat_A32as</td>
<td>acc32_t</td>
<td>frac16_t</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 2.13.2 Declaration

The available MLIB_Div1QSat functions have the following declarations:

```c
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)
```

### 2.13.3 Function use

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

```c
#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 */
```
2.14 MLIB_Log2

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

\[ \text{MLIB\_Log2}(x) = \begin{cases} 
0, & x \leq 1 \\
\log_2(x), & \text{else} 
\end{cases} \]

Figure 38. Algorithm formula

2.14.1 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Log2_U16</td>
<td>uint16_t</td>
<td>uint16_t</td>
<td>Binary logarithm of a 16-bit unsigned integer value. The output is greater than or equal to 0.</td>
</tr>
</tbody>
</table>

2.14.2 Declaration

The available MLIB_Log2 functions have the following declarations:

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

2.14.3 Function use

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

```c
#include "mlib.h"

static uint16_t u16Result, u16Val;

void main(void)
{
    u16Val = 5;            /* u16Val = 5 */
    /* u16Result = log2(u16Val) */
    u16Result = MLIB_Log2_U16(u16Val);
}
```

2.15 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:
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 \(-1 \leq \text{result} \leq 1\). The result may overflow.

- Accumulator output with mixed inputs - the output is the accumulator type, where the result can be out of the range \(-1 \leq \text{result} \leq 1\). 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 16. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accum.</td>
<td>Mult. 1</td>
<td>Mult. 2</td>
</tr>
<tr>
<td>MLIB_Mac_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td>MLIB_Mac_F32lss</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td>MLIB_Mac_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>frac32_t</td>
</tr>
<tr>
<td>MLIB_Mac_A32ass</td>
<td>acc32_t</td>
<td>frac16_t</td>
<td>frac16_t</td>
</tr>
</tbody>
</table>

2.15.2 Declaration

The available MLIB_Mac functions have the following declarations:

```c
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)
```

2.15.3 Function use

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

**Fixed-point version:**

```c
#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);
}

2.16 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:

\[
MLIB\_MacSat(a, b, c) = \begin{cases} 
    1, & a + b \cdot c > 1 \\
    -1, & a + b \cdot c < -1 \\
    a + b \cdot c, & \text{else}
\end{cases}
\]

Figure 40. 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 saturate.

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

Table 17. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_MacSat_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td></td>
<td>Accum.</td>
<td>Mult. 1</td>
<td>Mult. 2</td>
</tr>
<tr>
<td>MLIB_MacSat_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>frac32_t</td>
</tr>
<tr>
<td></td>
<td>Accum.</td>
<td>Mult. 1</td>
<td>Mult. 2</td>
</tr>
<tr>
<td>MLIB_MacSat_F32lss</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>frac32_t</td>
</tr>
<tr>
<td></td>
<td>Accum.</td>
<td>Mult. 1</td>
<td>Mult. 2</td>
</tr>
</tbody>
</table>
2.16.2 Declaration

The available MLIB_MacSat functions have the following declarations:

\[
\begin{align*}
\text{frac16}_t & \quad \text{MLIB}_\text{MacSat}\_F16(\text{frac16}_t \ f16Accum, \ \text{frac16}_t \ f16Mult1, \ \text{frac16}_t \ f16Mult2) \\
\text{frac32}_t & \quad \text{MLIB}_\text{MacSat}\_F32\_\text{ss}(\text{frac32}_t \ f32Accum, \ \text{frac16}_t \ f16Mult1, \ \text{frac16}_t \ f16Mult2) \\
\text{frac32}_t & \quad \text{MLIB}_\text{MacSat}\_F32(\text{frac32}_t \ f32Accum, \ \text{frac32}_t \ f32Mult1, \ \text{frac32}_t \ f32Mult2)
\end{align*}
\]

2.16.3 Function use

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

```c
#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);
}
```

2.17 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:

\[
\text{MLIB}_\text{MacRnd}(a, b, c) = a + \text{round}(b \cdot c)
\]

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 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_MacRnd_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>The fractional product (of two 16-bit fractional multiplicands), rounded to the upper 16 bits, is added</td>
</tr>
</tbody>
</table>

Table continues on the next page...
Table 18. Function versions (continued)

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accum.</td>
<td>Mult. 1</td>
<td>Mult. 2</td>
</tr>
<tr>
<td>MLIB_MacRnd_F32lls</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td>MLIB_MacRnd_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>frac32_t</td>
</tr>
<tr>
<td>MLIB_MacRnd_A32ass</td>
<td>acc32_t</td>
<td>frac16_t</td>
<td>frac16_t</td>
</tr>
</tbody>
</table>

2.17.2 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)
```

2.17.3 Function use

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

```c
#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);
}
```

2.18 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:
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 saturate.

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

Table 19. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MLIB_MacRndSat_F16</code></td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td><code>MLIB_MacRndSat_F32lls</code></td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>frac32_t</td>
</tr>
<tr>
<td><code>MLIB_MacRndSat_F32</code></td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>frac32_t</td>
</tr>
</tbody>
</table>

2.18.2 Declaration
The available `MLIB_MacRndSat` functions have the following declarations:

```c
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)
```

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

```c
#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 */
    ...  
```
2.19 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:

\[ \text{MLIB\_Mac4}\left(a, b, c, d\right) = a \cdot b + c \cdot d \]

Figure 43. 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 overflow.

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

Table 20. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Product 1</td>
<td>Product 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mult. 1</td>
<td>Mult. 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mult. 1</td>
<td>Mult. 2</td>
<td></td>
</tr>
<tr>
<td>MLIB_Mac4_F32ssss</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Addition of two 32-bit fractional products (of two 16-bit fractional multiplicands). The output is within the range (-1 ; 1).</td>
</tr>
</tbody>
</table>

2.19.2 Declaration

The available MLIB_Mac4 functions have the following declarations:

\[
\text{frac32\_t \ MLIB\_Mac4\_F32ssss(frac16\_t f16Add1Mult1, frac16\_t f16Add1Mult2, frac16\_t f16Add2Mult1, frac16\_t f16Add2Mult2)}
\]

2.19.3 Function use

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

Fixed-point version:

```c
#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);
}

2.20 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:

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

Figure 44. Algorithm formula

2.20.1 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 saturate.

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

Table 21. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Mac4Sat_F32ssss</td>
<td>frac16_t</td>
<td>frac32_t</td>
<td>Addition of two 32-bit fractional products (of two 16-bit fractional multiplicands). The output is within the range &lt;-1 ; 1).</td>
</tr>
</tbody>
</table>

2.20.2 Declaration

The available MLIB_Mac4Sat functions have the following declarations:

\[
\text{frac32_t MLIB\_Mac4Sat\_F32ssss}(\text{frac16_t f16Add1Mult1, frac16_t f16Add1Mult2, frac16_t f16Add2Mult1, frac16_t f16Add2Mult2})
\]
2.20.3 Function use

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

```c
#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);
}
```

2.21 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:

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

Figure 45. Algorithm formula

2.21.1 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Product 1</td>
<td>Product 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mult. 1</td>
<td>Mult. 2</td>
<td></td>
</tr>
<tr>
<td>MLIB_Mac4Rnd_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>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 $&lt;-1; 1)$.</td>
</tr>
<tr>
<td>MLIB_Mac4Rnd_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>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 $&lt;-1; 1)$.</td>
</tr>
</tbody>
</table>
### 2.21.2 Declaration

The available **MLIB_Mac4Rnd** functions have the following declarations:

```c
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)
```

### 2.21.3 Function use

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

```c
#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);
}
```

### 2.22 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:

$$MLIB_{\text{Mac4RndSat}}(a, b, c, d) = \begin{cases} 
1, & \text{round}(a \cdot b + c \cdot d) > 1 \\
-1, & \text{round}(a \cdot b + c \cdot d) < -1 \\
\text{round}(a \cdot b + c \cdot d), & \text{else}
\end{cases}$$

Figure 46. Algorithm formula

#### 2.22.1 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 23. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Product 1</td>
<td>Product 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mult. 1</td>
<td>Mult. 2</td>
<td>Mult. 1</td>
</tr>
<tr>
<td>MLIB_Mac4RndSat_F16</td>
<td>frac16_t</td>
<td></td>
<td>frac16_t</td>
</tr>
<tr>
<td>MLIB_Mac4RndSat_F32</td>
<td>frac32_t</td>
<td></td>
<td>frac32_t</td>
</tr>
</tbody>
</table>

2.22.2 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)
```

2.22.3 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);
}
```
2.23 MLIB_Mnac

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

\[
\text{MLIB}_\text{Mnac}(a, b, c) = b \cdot c - a
\]

Figure 47. 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 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_Mnac function are shown in the following table.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Mnac_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>The 16-bit fractional accumulator is subtracted from the upper 16-bit portion [16..31] of the fractional product (of two 16-bit fractional multiplicands). The output is within the range (-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_Mnac_F32lss</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>The 32-bit fractional accumulator is subtracted from the 32-bit fractional product (of two 16-bit fractional multiplicands). The output is within the range (-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_Mnac_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>The 32-bit fractional accumulator is subtracted from the upper 32-bit portion [32..63] of the fractional product (of two 32-bit fractional multiplicands). The output is within the range (-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_Mnac_A32ass</td>
<td>acc32_t</td>
<td>frac16_t</td>
<td>The 32-bit accumulator is subtracted from the upper 16-bit portion [16..31] of the fractional product (of two 16-bit fractional multiplicands). The output may be out of the range (-65536 ; 65536).</td>
</tr>
</tbody>
</table>

2.23.2 Declaration

The available MLIB_Mnac functions have the following declarations:

```c
frac16_t MLIB_Mnac_F16(frac16_t f16Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_Mnac_F32lss(frac32_t f32Accum, frac16_t f16Mult1, frac16_t f16Mult2)
frac32_t MLIB_Mnac_F32(frac32_t f32Accum, frac32_t f32Mult1, frac32_t f32Mult2)
acc32_t MLIB_Mnac_A32ass(acc32_t a32Accum, frac16_t f16Mult1, frac16_t f16Mult2)
```

2.23.3 Function use

The use of the MLIB_Mnac function is shown in the following examples:
Fixed-point version:

```c
#include "mlib.h"

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

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

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

2.24 MLIB_MnacSat

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

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

Figure 48. 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 saturate.

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

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_MnacSat_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td>MLIB_MnacSat_F32lss</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>frac32_t</td>
</tr>
<tr>
<td>MLIB_MnacSat_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>frac32_t</td>
</tr>
</tbody>
</table>
2.24.2 Declaration

The available MLIB_MnacSat functions have the following declarations:

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

2.24.3 Function use

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

```c
#include "mlib.h"

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

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

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

2.25 MLIB_MnacRnd

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

\[
\text{MLIB_MnacRnd}(a, b, c) = \text{round}(b \cdot c) - a
\]

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

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

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_MnacRnd_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>The 16-bit fractional accumulator is subtracted from the fractional product (of two 16-bit fractional...</td>
</tr>
</tbody>
</table>
Table 26. Function versions (continued)

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accum.</td>
<td>Mult. 1</td>
<td>Mult. 2</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLIB_MnacRnd_F32lls</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td>MLIB_MnacRnd_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>frac32_t</td>
</tr>
<tr>
<td>MLIB_MnacRnd_A32ass</td>
<td>acc32_t</td>
<td>frac16_t</td>
<td>frac16_t</td>
</tr>
</tbody>
</table>

2.25.2 Declaration

The available MLIB_MnacRnd functions have the following declarations:

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

2.25.3 Function use

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

```c
#include "mlib.h"

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

void main(void)
{
    f32Accum = FRAC32(0.3);     /* f32Accum = 0.3 */
    f32Mult1 = FRAC32(0.4);     /* f32Mult1 = 0.4 */
    f16Mult2 = FRAC16(-0.2);    /* f16Mult2 = -0.2 */
    /* f32Result = round(f32Mult1 * f16Mult2 - f32Accum) */
    f32Result = MLIB_MnacRnd_F32lls(f32Accum, f32Mult1, f16Mult2);
}
```
### 2.26 MLIB_MnacRndSat

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

\[
\text{MLIB_MnacRndSat}(a, b, c) = \begin{cases} 
1, & \text{round}(b \cdot c) - a > 1 \\
-1, & \text{round}(b \cdot c) - a < -1 \\
\text{round}(b \cdot c) - a, & \text{else}
\end{cases}
\]

Figure 50. Algorithm formula

#### 2.26.1 Available versions

This function is available in the following versions:

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

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

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_MnacRndSat_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>The 16-bit fractional accumulator is subtracted from the fractional product (of two 16-bit fractional multiplicands) rounded to the upper 16 bits. The output is within the range &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_MnacRndSat_F32lls</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>The 32-bit fractional accumulator is subtracted from the fractional product (of a 32-bit and a 16-bit fractional multiplicand) rounded to the upper 32 bits [16..48]. The output is within the range &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_MnacRndSat_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>The 32-bit fractional accumulator is subtracted from the fractional product (of two 32-bit fractional multiplicands) rounded to the upper 32 bits [32..63]. The output is within the range &lt;-1 ; 1).</td>
</tr>
</tbody>
</table>

#### 2.26.2 Declaration

The available MLIB_MnacRndSat functions have the following declarations:

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

#### 2.26.3 Function use

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

```
#include "mlib.h"

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

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

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

2.27 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:

\[
MLIB\_Msu(a, b, c) = a - b \cdot c
\]

Figure 51. 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 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 28. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accum.</td>
<td>Mult. 1</td>
<td>Mult. 2</td>
</tr>
<tr>
<td>MLIB_Msu_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>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 &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_Msu_F32lss</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>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 &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_Msu_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>frac32_t</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>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 &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_Msu_A32ass</td>
<td>acc32_t</td>
<td>frac16_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>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 &lt;-65536 ; 65536).</td>
</tr>
</tbody>
</table>
2.27.2 Declaration
The available MLIB_Msu functions have the following declarations:

\[
\begin{align*}
\text{frac16_t } & \text{ MLIB_Msu_F16(frac16_t f16Accum, frac16_t f16Mult1, frac16_t f16Mult2)} \\
\text{frac32_t } & \text{ MLIB_Msu_F32lss(frac32_t f32Accum, frac16_t f16Mult1, frac16_t f16Mult2)} \\
\text{frac32_t } & \text{ MLIB_Msu_F32(frac32_t f32Accum, frac16_t f16Mult1, frac16_t f32Mult2)} \\
\text{acc32_t } & \text{ MLIB_Msu_A32ass(acc32_t a32Accum, frac16_t f16Mult1, frac16_t f16Mult2)}
\end{align*}
\]

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

Fixed-point version:

```c
#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 = MLIB_Msu_A32ass(a32Accum, f16Mult1, f16Mult2);
}
```

2.28 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:

\[
\text{MLIB_MsuSat}(a, b, c) = \begin{cases} 
1, & a - b \cdot c > 1 \\
-1, & a - b \cdot c < -1 \\
\text{round}(a - b \cdot c), & \text{else}
\end{cases}
\]

Figure 52. 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 saturate.

The available versions of the MLIB_MsuSat function are shown in the following table.
Table 29. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_MsuSat_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>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 &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_MsuSat_F32lss</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>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 &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_MsuSat_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>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 &lt;-1 ; 1).</td>
</tr>
</tbody>
</table>

2.28.2 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)
```

2.28.3 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);
}
```

2.29 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:

```
LIB_MsuRnd(a, b, c) = a - round(b * c)
```

Figure 53. 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 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_MsuRnd_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>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 $&lt;-1 ; 1)$.</td>
</tr>
<tr>
<td>MLIB_MsuRnd_F32lls</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>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 $&lt;-1 ; 1)$.</td>
</tr>
<tr>
<td>MLIB_MsuRnd_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>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 $&lt;-1 ; 1)$.</td>
</tr>
<tr>
<td>MLIB_MsuRnd_A32ass</td>
<td>acc32_t</td>
<td>frac16_t</td>
<td>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 $&lt;-65536 ; 65536)$.</td>
</tr>
</tbody>
</table>

2.29.2 Declaration

The available MLIB_MsuRnd functions have the following declarations:

```c
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)
```

2.29.3 Function use

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

```c
#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 */
```
2.30 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:

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

Figure 54. Algorithm formula

2.30.1 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 31. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_MsuRndSat_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td>MLIB_MsuRndSat_F32lls</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td>MLIB_MsuRndSat_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>frac32_t</td>
</tr>
</tbody>
</table>

2.30.2 Declaration

The available MLIB_MsuRndSat functions have the following declarations:

```c
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)
```
2.30.3 Function use

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

```c
#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);
}
```

2.31 `MLIB_Msu4`

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

\[ \text{MLIB}_\text{Mu}_4(a,b,c,d) = a \cdot b - c \cdot d \]

Figure 55. Algorithm formula

2.31.1 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MLIB_Msu4_F32ssss</code></td>
<td>frac16_t</td>
<td>frac32_t</td>
<td>Subtraction of two 32-bit fractional products (of two 16-bit fractional multiplicands). The output is within the range (-1 ; 1).</td>
</tr>
</tbody>
</table>

2.31.2 Declaration

The available `MLIB_Msu4` functions have the following declarations:

```c
frac32_t MLIB_Msu4_F32ssss(frac16_t f16MinMult1, frac16_t f16MinMult2, frac16_t f16SubMult1, frac16_t f16SubMult2)
```
2.31.3 Function use

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

Fixed-point version:

```c
#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 = MLIB_Msu4_F32ssss(f16MinMult1, f16MinMult2, f16SubMult1, f16SubMult2);
}
```

2.32 MLIB_Msu4Sat

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

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

Figure 56. Algorithm formula

2.32.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range \([-1, 1)\). The result may saturate.

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

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Minuend product</td>
<td>Subtrahend product</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mult. 1</td>
<td>Mult. 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mult. 1</td>
<td>Mult. 2</td>
<td></td>
</tr>
<tr>
<td>MLIB_Msu4Sat_F32ssss</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Subtraction of two 32-bit fractional products (of two 16-bit fractional)</td>
</tr>
<tr>
<td></td>
<td>frac16_t</td>
<td>frac16_t</td>
<td></td>
</tr>
<tr>
<td></td>
<td>frac16_t</td>
<td>frac32_t</td>
<td></td>
</tr>
</tbody>
</table>

Table continues on the next page...
### Table 33. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Minuend product</td>
<td>Subtrahend product</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mult. 1</td>
<td>Mult. 2</td>
<td>Mult. 1</td>
</tr>
</tbody>
</table>

#### 2.32.2 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)
```

#### 2.32.3 Function use

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

```c
#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);
}
```

#### 2.33 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:

```
MLIB_Msu4Rnd(a, b, c, d) = round(a * b - c * d)
```

#### 2.33.1 Available versions

This function is available in the following versions:
• Fractional output - the output is the fractional portion of the result; the result is within the range $<-1; 1)$. The result may overflow.

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

Table 34. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Minuend product</td>
<td>Subtrahend product</td>
<td>Mult. 1</td>
</tr>
<tr>
<td>MLIB_Msu4Rnd_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td>MLIB_Msu4Rnd_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>frac32_t</td>
</tr>
</tbody>
</table>

2.33.2 Declaration

The available MLIB_Msu4Rnd functions have the following declarations:

```c
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)
```

2.33.3 Function use

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

```c
#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);
}
2.34 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:

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

Figure 58. Algorithm formula

2.34.1 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Minuend product</td>
<td>Subtrahend product</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mult. 1</td>
<td>Mult. 2</td>
<td>Mult. 1</td>
</tr>
<tr>
<td>MLIB_Msu4RndSat_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td>MLIB_Msu4RndSat_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>frac32_t</td>
</tr>
</tbody>
</table>

2.34.2 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)`
2.34.3 Function use

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

```c
#include "mlib.h"

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

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

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

2.35 MLIB_Mul

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

\[
\text{MLIB}_\text{Mul}(a, b) = a \cdot b
\]

Figure 59. Algorithm formula

2.35.1 Available versions

This function is available in the following versions:

- Fractional output with fractional inputs - the output is the fractional portion of the result; the result is within the range \([-1; 1)\). The inputs are the fractional values only. The result may 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MLIB_Mul_F16</code></td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>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)).</td>
</tr>
<tr>
<td><code>MLIB_Mul_F16as</code></td>
<td>acc32_t</td>
<td>frac16_t</td>
<td>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)).</td>
</tr>
</tbody>
</table>
Table 36. Function versions (continued)

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Mul_F32ss</td>
<td>frac16_t</td>
<td>frac32_t</td>
<td>Product of two 16-bit fractional multiplicands; the result is a 32-bit fractional value. The output is within the range &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_Mul_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>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 &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_Mul_A32</td>
<td>acc32_t</td>
<td>acc32_t</td>
<td>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 &lt;-65536.0 ; 65536.0).</td>
</tr>
</tbody>
</table>

2.35.2 Declaration

The available MLIB_Mul functions have the following declarations:

```c
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)
```

2.35.3 Function use

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

```c
#include "mlib.h"

static frac32_t f32Result;
static frac16_t f16Mult1, f16Mult2;

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

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

2.36 MLIB_MulSat

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

\[
MLIB_{\text{MulSat}}(a, b) = \begin{cases} 
    \max, & a \cdot b > \max \\
    \min, & a \cdot b < \min \\
    a \cdot b, & \text{else}
\end{cases}
\]

Figure 60. Algorithm formula
2.36.1 Available versions

This function is available in the following versions:

- **Fractional output with fractional inputs** - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The inputs are the fractional values only. 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_MulSat_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td>MLIB_MulSat_F16as</td>
<td>acc32_t</td>
<td>frac16_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td>MLIB_MulSat_F32ss</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>frac32_t</td>
</tr>
<tr>
<td>MLIB_MulSat_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>frac32_t</td>
</tr>
<tr>
<td>MLIB_MulSat_A32</td>
<td>acc32_t</td>
<td>acc32_t</td>
<td>acc32_t</td>
</tr>
</tbody>
</table>

2.36.2 Declaration

The available MLIB_MulSat functions have the following declarations:

```c
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)
```

2.36.3 Function use

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

```c
#include "mlib.h"

static acc32_t a32Accum;
static frac16_t f16Mult, f16Result;

void main(void)
{

```
2.37 MLIB_MulNeg

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

\[ \text{MLIB\_MulNeg}(a, b) = -a \times b \]

2.37.1 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_MulNeg_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>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)).</td>
</tr>
<tr>
<td>MLIB_MulNeg_F16as</td>
<td>acc32_t</td>
<td>frac16_t</td>
<td>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)).</td>
</tr>
<tr>
<td>MLIB_MulNeg_F32ss</td>
<td>frac16_t</td>
<td>frac32_t</td>
<td>Negative product of two 16-bit fractional multiplicands; the result is a 32-bit fractional value. The output is within the range ((-1 ; 1)).</td>
</tr>
<tr>
<td>MLIB_MulNeg_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>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)).</td>
</tr>
<tr>
<td>MLIB_MulNeg_A32</td>
<td>acc32_t</td>
<td>acc32_t</td>
<td>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)).</td>
</tr>
</tbody>
</table>
2.37.2 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 a32Mult2)`

2.37.3 Function use

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

```
Fixed-point version:

#include "mlib.h"

static frac32_t f32Result;
static frac16_t f16Mult1, f16Mult2;

void main(void)
{
    f16Mult1 = FRAC16(0.5);     /* f16Mult1 = 0.5 */
    f16Mult2 = FRAC16(-0.3);    /* f16Mult2 = -0.3 */

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

2.38 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:

MLIB_MulNegRnd(a, b) = round(-a * b)

Figure 62. Algorithm formula

2.38.1 Available versions

The 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.

The available versions of the `MLIB_MulNegRnd` function are shown in the following table.
Table 39. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_MulNegRnd_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Negative product of two 16-bit fractional multiplicands; the output is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>frac16_t</td>
<td>rounded to the upper 16 bits of the results [16..31]. The output is within</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the range &lt;-1;1).</td>
</tr>
<tr>
<td>MLIB_MulNegRnd_F16as</td>
<td>acc32_t</td>
<td>frac16_t</td>
<td>Negative product of a 32-bit accumulator and a 16-bit fractional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>multiplicand; the output is a 16-bit fractional value which is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>rounded to the upper 16 bits of the fractional portion of the result</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[16..31]. The output is within the range &lt;-1;1).</td>
</tr>
<tr>
<td>MLIB_MulNegRnd_F32ls</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>Negative product of a 32-bit and a 16-bit fractional multiplicands; the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>frac32_t</td>
<td>output is rounded to the upper 32 bits of the fractional portion of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>result [16..47]. The output is within the range &lt;-1;1).</td>
</tr>
<tr>
<td>MLIB_MulNegRnd_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>Negative product of two 32-bit fractional multiplicands; the output is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>frac32_t</td>
<td>rounded to the upper 32 bits of the results [16..31]. The output is within</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the range &lt;-1;1).</td>
</tr>
</tbody>
</table>

2.38.2 Declaration

The available MLIB_MulNegRnd functions have the following declarations:

```c
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)
```

2.38.3 Function use

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

```c
#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);
}
```

2.39 MLIB_MulNegSat

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

This function is available in the following versions:

- Fractional output with mixed inputs - the output is the fractional portion of the result; the result is within the range \([-1 ; 1)\). The inputs are the accumulator and fractional values. The result may saturate.

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

The available versions of the \texttt{MLIB\_MulNegSat} function are shown in the following table:

Table 40. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{MLIB_MulNegSat_F16_as}</td>
<td>acc32_t, frac16_t</td>
<td>frac16_t</td>
<td>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)).</td>
</tr>
<tr>
<td>\texttt{MLIB_MulNegSat_A32}</td>
<td>acc32_t, acc32_t</td>
<td>acc32_t</td>
<td>Negative product of two 32-bit accumulator multiplicands; the output is a 32-bit accumulator, which has the middle bits of the result [16..47]. The output is within the range ([-65536.0 ; 65536.0)).</td>
</tr>
</tbody>
</table>

2.39.2 Declaration

The available \texttt{MLIB\_MulNegSat} functions have the following declarations:

\begin{verbatim}
frac16\_t \texttt{MLIB\_MulNegSat\_F16\_as}(acc32\_t a32Accum, frac16\_t f16Mult)
acc32\_t \texttt{MLIB\_MulNegSat\_A32}(acc32\_t a32Mult1, acc32\_t a32Mult2)
\end{verbatim}

2.39.3 Function use

The use of the \texttt{MLIB\_MulNegSat} function is shown in the following example:

\begin{verbatim}
#include "mlib.h"

static acc32\_t a32M1, a32M2, a32Result;

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

  /* f16Result = sat(-a32M1 * f32M2) */
  a32Result = MLIB\_MulNegSat\_A32(a32M1, a32M2);
}
\end{verbatim}
2.40 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:

\[
\text{MLIB\_MulRnd}(a, b) = \text{round}(a \times b)
\]

Figure 64. Algorithm formula

2.40.1 Available versions

This function is available in the following versions:

- Fractional output with fractional inputs - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The inputs are the fractional values only. 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_MulRnd_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>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 &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_MulRnd_F16as</td>
<td>acc32_t</td>
<td>frac16_t</td>
<td>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 &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_MulRnd_F32ls</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>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 &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_MulRnd_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>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 &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_MulRnd_A32</td>
<td>acc32_t</td>
<td>acc32_t</td>
<td>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 &lt;-65536.0 ; 65536.0).</td>
</tr>
</tbody>
</table>

2.40.2 Declaration

The available MLIB_MulRnd functions have the following declarations:

\[\text{frac16}_t \text{ MLIB\_MulRnd\_F16}(\text{frac16}_t \text{ f16Mult1}, \text{frac16}_t \text{ f16Mult2})\]
\[\text{frac16}_t \text{ MLIB\_MulRnd\_F16as}(\text{acc32}_t \text{ a32Accum}, \text{frac16}_t \text{ f16Mult})\]
\[\text{frac32}_t \text{ MLIB\_MulRnd\_F32ls}(\text{frac32}_t \text{ f32Mult1}, \text{frac16}_t \text{ f16Mult2})\]
\[\text{frac32}_t \text{ MLIB\_MulRnd\_F32}(\text{frac32}_t \text{ f32Mult1}, \text{frac32}_t \text{ f32Mult2})\]
\[\text{acc32}_t \text{ MLIB\_MulRnd\_A32}(\text{acc32}_t \text{ a32Mult1}, \text{acc32}_t \text{ a32Mult1})\]
2.40.3 Function use

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

```c
#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);
}
```

2.41 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:

```
MLIB_MulRndSat(a, b) = \begin{cases} 
  \text{max}, & \text{round}(a \cdot b) > \text{max} \\
  \text{min}, & \text{round}(a \cdot b) < \text{min} \\
  \text{round}(a \cdot b), & \text{else}
\end{cases}
```

Figure 65. Algorithm formula

2.41.1 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_MulRndSat_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>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 &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_MulRndSat_F16as</td>
<td>acc32_t</td>
<td>frac16_t</td>
<td>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 &lt;-1 ; 1).</td>
</tr>
</tbody>
</table>

Table continues on the next page...
Table 42. Function versions (continued)

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_MulRndSat_F32ls</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>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 &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_MulRndSat_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>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 &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_MulRndSat_A32</td>
<td>acc32_t</td>
<td>acc32_t</td>
<td>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 &lt;-65536.0 ; 65536.0).</td>
</tr>
</tbody>
</table>

2.41.2 Declaration

The available MLIB_MulRndSat functions have the following declarations:

```c
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)
```

2.41.3 Function use

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

```c
#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 = MLIB_MulRndSat_F32(f32Mult1, f32Mult2);
}
```

2.42 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:

```
MLIB_MulNegRnd(a, b) = round(-α·b)
```

Figure 66. Algorithm formula
2.42.1 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_MulNegRnd_F16</td>
<td>frac16_t, frac16_t</td>
<td>frac16_t</td>
<td>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 &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_MulNegRnd_F16as</td>
<td>acc32_t, frac16_t</td>
<td>frac16_t</td>
<td>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 &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_MulNegRnd_F32ls</td>
<td>frac32_t, frac16_t</td>
<td>frac32_t</td>
<td>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 &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_MulNegRnd_F32</td>
<td>frac32_t, frac32_t</td>
<td>frac32_t</td>
<td>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 &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_MulNegRnd_A32</td>
<td>acc32_t, acc32_t</td>
<td>acc32_t</td>
<td>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 &lt;-65536.0 ; 65536.0).</td>
</tr>
</tbody>
</table>

2.42.2 Declaration

The available MLIB_MulNegRnd functions have the following declarations:

```c
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 a32Mult2)
```

2.42.3 Function use

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

```c
#include "mlib.h"

static frac32_t f32Mult1, f32Mult2, f32Result;
```
```c
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);
}
```

### 2.43 MLIB_MulNegRndSat

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

\[
\text{MLIB_MulNegRndSat}(a, b) = \begin{cases} 
    \text{max}, & \text{round}(-a \cdot b > \text{max}) \\
    \text{min}, & \text{round}(-a \cdot b < \text{min}) \\
    \text{round}(-a \cdot b), & \text{else}
\end{cases}
\]

Figure 67. Algorithm formula

#### 2.43.1 Available versions

This function is available in the following versions:

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

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

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_MulNegRndSat_F16as</td>
<td>acc32_t</td>
<td>frac16_t</td>
<td>Negative product of a 32-bit accumulator and a 16-bit fractional multiplicand; the output is rounded to the upper 16 bits of the fractional portion of the result ([16..31]). The output is within the range (&lt;-1 ; 1)).</td>
</tr>
<tr>
<td>MLIB_MulNegRndSat_A32</td>
<td>acc32_t</td>
<td>acc32_t</td>
<td>Negative product of two 32-bit accumulator multiplicands; the output is rounded to the middle 32 bits of the result ([16..47]). The output is within the range (&lt;-65536.0 ; 65536.0)).</td>
</tr>
</tbody>
</table>

#### 2.43.2 Declaration

The available MLIB_MulNegRndSat functions have the following declarations:

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

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

```c
#include "mlib.h"

static acc32_t a32M1, a32M2, a32Result;
void main(void)
{
    a32M1 = ACC32(-5.5);       /* a32M1 = -5.5 */
    a32M2 = ACC32(3.1);        /* a32M2 = 3.1 */
    /* f16Result = sat(round(-a32M1 * f32M2)) */
    a32Result = MLIB_MulNegRndSat_A32(a32M1, a32M2);
}
```

2.44 MLIB_Neg

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

\[ MLIB_{Neg}(x) = -x \]

Figure 68. 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 overflow.

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

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Neg_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Negative value of a 16-bit fractional value. The output is within the range ((-1 ; 1)).</td>
</tr>
<tr>
<td>MLIB_Neg_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>Negative value of a 32-bit fractional value. The output is within the range ((-1 ; 1)).</td>
</tr>
</tbody>
</table>

2.44.2 Declaration

The available MLIB_Neg functions have the following declarations:

```c
frac16_t MLIB_Neg_F16(frac16_t f16Val)
frac32_t MLIB_Neg_F32(frac32_t f32Val)
```

2.44.3 Function use

The use of the MLIB_Neg function is shown in the following examples:
Fixed-point version:

```c
#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);
}
```

2.45 MLIB_NegSat

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

\[
\text{MLIB\_NegSat}(x) = -x
\]

Figure 69. Algorithm formula

2.45.1 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_NegSat_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Negative value of a 16-bit value. The output is within the range (-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_NegSat_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>Negative value of a 32-bit value. The output is within the range (-1 ; 1).</td>
</tr>
</tbody>
</table>

2.45.2 Declaration

The available MLIB_NegSat functions have the following declarations:

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

2.45.3 Function use

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

```c
#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);
}
```
2.46 **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:

![Figure 70. Algorithm formula](image)

### 2.46.1 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Rcp_A32s</td>
<td>frac16_t</td>
<td>acc32_t</td>
<td>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.</td>
</tr>
<tr>
<td>MLIB_Rcp1_A32s</td>
<td>frac16_t</td>
<td>acc32_t</td>
<td>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.</td>
</tr>
</tbody>
</table>

#### 2.46.2 Declaration

The available **MLIB\_Rcp** functions have the following declarations:

```c
acc32\_t MLIB\_Rcp\_A32s(frac16\_t f16Denom)
acc32\_t MLIB\_Rcp1\_A32s(frac16\_t f16Denom)
```

#### 2.46.3 Function use

The use of the **MLIB\_Rcp** function is shown in the following example:

```c
#include "mlib.h"

static acc32\_t a32Result;
static frac16\_t f16Denom;

void main(void)
{
    f16Denom = FRAC16(0.354);  /* f16Denom = 0.354 */
    /* a32Result = 1/f16Denom */
}```
2.47 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:

\[
MLIB_Rcp1Q(x) = \begin{cases} 
\text{max}, & x = 0 \\
\frac{1}{x}, & x > 0 
\end{cases}
\]

Figure 71. Algorithm formula

2.47.1 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Rcp1Q_A32s</td>
<td>frac16_t</td>
<td>acc32_t</td>
<td>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.</td>
</tr>
<tr>
<td>MLIB_Rcp1Q1_A32s</td>
<td>frac16_t</td>
<td>acc32_t</td>
<td>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.</td>
</tr>
</tbody>
</table>

2.47.2 Declaration

The available MLIB_Rcp1Q functions have the following declarations:

\[
\begin{align*}
\text{acc32_t } & \text{ MLIB_Rcp1Q_A32s(frac16_t f16Denom)} \\
\text{acc32_t } & \text{ MLIB_Rcp1Q1_A32s(frac16_t f16Denom)}
\end{align*}
\]

2.47.3 Function use

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

```c
#include "mlib.h"

static acc32_t a32Result;
static frac16_t f16Denom;

void main(void)
{
    f16Denom = FRAC16(0.354); /* f16Denom = 0.354 */
    a32Result = MLIB_Rcp1_A32s(f16Denom);
    /* a32Result = 1/f16Denom */
```
2.48 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:

\[ \text{MLIB}_\text{Rnd}(x) = \text{round}(x) \]  

**Figure 72. 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 result may overflow.

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

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Rnd_F16l</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>Rounding of a 32-bit fractional value to a 16-bit fractional value. The output is within the range ([-1; 1)).</td>
</tr>
</tbody>
</table>

### 2.48.2 Declaration

The available MLIB_Rnd functions have the following declarations:

\[
\text{frac16}_t \ MLIB\_\text{Rnd}\_F16l(\text{frac32}_t \ f32Val)
\]

### 2.48.3 Function use

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

```c
#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);
}
```
2.49 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:

\[
\text{MLIB_RndSat}(x) = \text{round}(x)
\]

Figure 73. Algorithm formula

### 2.49.1 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_RndSat_F16l</td>
<td>frac32_t</td>
<td>frac16_t</td>
<td>Rounding of a 32-bit fractional value to a 16-bit fractional value. The output is within the range ([-1 ; 1)).</td>
</tr>
</tbody>
</table>

### 2.49.2 Declaration

The available **MLIB_RndSat** functions have the following declarations:

\[
\text{frac16_t MLIB_RndSat_F16l(frac32_t f32Val)}
\]

### 2.49.3 Function use

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

```c
#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);
}
```

2.50 MLIB_Sat

The **MLIB_Sat** functions return the fractional portion of the accumulator input. The output is saturated if necessary. See the following equation:
Figure 74. Algorithm formula

\[
MLIB_{\text{Sat}}(x) = \begin{cases} 
1, & x > 1 \\
-1, & x < -1 \\
x, & \text{else}
\end{cases}
\]

2.50.1 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 \texttt{MLIB\_Sat} function are shown in the following table.

Table 51. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{MLIB_Sat_F16a}</td>
<td>\texttt{acc32_t}</td>
<td>\texttt{frac16_t}</td>
<td>Saturation of a 32-bit accumulator value to a 16-bit fractional value. The output is within the range ((-1; 1)).</td>
</tr>
</tbody>
</table>

2.50.2 Declaration

The available \texttt{MLIB\_Sat} functions have the following declarations:

\[
\texttt{frac16\_t MLIB\_Sat\_F16a(acc32\_t a32Accum)}
\]

2.50.3 Function use

The use of the \texttt{MLIB\_Sat} function is shown in the following example:

```c
#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);
}
```

2.51 \texttt{MLIB\_Sh1L}

The \texttt{MLIB\_Sh1L} functions return the arithmetically one-time-shifted value to the left. The function does not saturate the output. See the following equation:

\[
MLIB\_Sh1L(x) = x \ll 1
\]

Figure 75. Algorithm formula
2.51.1 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MLIB_Sh1L_F16</code></td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Shift of a 16-bit fractional value by one time to the left. The output is within the range (-1 ; 1).</td>
</tr>
<tr>
<td><code>MLIB_Sh1L_F32</code></td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>Shift of a 32-bit fractional value by one time to the left. The output is within the range (-1 ; 1).</td>
</tr>
</tbody>
</table>

2.51.2 Declaration

The available `MLIB_Sh1L` functions have the following declarations:

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

2.51.3 Function use

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

```c
#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);
}
```

2.52 `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:

\[
MLIB\_Sh1LSat(x) = \begin{cases} 
1, & x > 0.5 \\
-1, & x < -0.5 \\
0, & \text{else} 
\end{cases}
\]

Figure 76. 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_Sh1LSat function are shown in the following table.

Table 53. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Sh1LSat_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Shift of a 16-bit fractional value by one time to the left. The output is within the range &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_Sh1LSat_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>Shift of a 32-bit fractional value by one time to the left. The output is within the range &lt;-1 ; 1).</td>
</tr>
</tbody>
</table>

2.52.2 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)

2.52.3 Function use

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

```c
#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);
}
```

2.53 MLIB_Sh1R

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

Figure 77. Algorithm formula

$$MLIB_{-}Sh1R(x) = x \gg 1$$

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 <-0.5 ; 0.5).

The available versions of the MLIB_Sh1R function are shown in the following table.
Table 54. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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). |

2.53.2 Declaration

The available MLIB_Sh1R functions have the following declarations:

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

2.53.3 Function use

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

```c
#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);
}
```

2.54 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:

\[
\text{MLIB}_{\text{ShL}}(x, n) = x \ll n
\]

Figure 78. 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.

The available versions of the MLIB_ShL function are shown in the following table.
Table 55. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_ShL_F16</td>
<td>frac16_t</td>
<td>uint16_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td>MLIB_ShL_F32</td>
<td>frac32_t</td>
<td>uint16_t</td>
<td>frac32_t</td>
</tr>
</tbody>
</table>

2.54.2 Declaration

The available MLIB_ShL functions have the following declarations:

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

2.54.3 Function use

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

```c
#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);
}
```

2.55 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:

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

Figure 79. 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_ShLSat** function are shown in the following table.

### Table 56. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MLIB_ShLSat_F16</strong></td>
<td>frac16_t</td>
<td>uint16_t</td>
<td>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).</td>
</tr>
<tr>
<td><strong>MLIB_ShLSat_F32</strong></td>
<td>frac32_t</td>
<td>uint16_t</td>
<td>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).</td>
</tr>
</tbody>
</table>

### 2.55.2 Declaration

The available **MLIB_ShLSat** functions have the following declarations:

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

### 2.55.3 Function use

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

```c
#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);
}
```

### 2.56 MLIB_ShR

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

\[
MLIB_{\text{ShR}}(x, n) = x \gg n
\]

### Figure 80. 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 available versions of the MLIB_ShR function are shown in the following table.

Table 57. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_ShR_F16</td>
<td>frac16_t</td>
<td>uint16_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td>MLIB_ShR_F32</td>
<td>frac32_t</td>
<td>uint16_t</td>
<td>frac32_t</td>
</tr>
</tbody>
</table>

2.56.2 Declaration

The available MLIB_ShR functions have the following declarations:

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

2.56.3 Function use

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

```c
#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 = MLIB_ShR_F16(f16Val, u16Sh);
}
```

2.57 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:

```
MLIB_ShLBi(x, n) = x << n
```

Figure 81. Algorithm formula

2.57.1 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 58. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_ShLBi_F16</td>
<td>frac16_t</td>
<td>int16_t</td>
<td>frac16_t</td>
</tr>
<tr>
<td>MLIB_ShLBi_F32</td>
<td>frac32_t</td>
<td>int16_t</td>
<td>frac32_t</td>
</tr>
</tbody>
</table>

2.57.2 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)

2.57.3 Function use

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

```c
#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);
}
```

2.58 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:
2.58.1 Available versions

This function is available in the following versions:

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

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

Table 59. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_ShLBiSat_F16</td>
<td>frac16_t</td>
<td>int16_t</td>
<td>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 &lt;-15 ; 15&gt;. The output is within the range &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_ShLBiSat_F32</td>
<td>frac32_t</td>
<td>int16_t</td>
<td>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 &lt;-31 ; 31&gt;. The output is within the range &lt;-1 ; 1).</td>
</tr>
</tbody>
</table>

2.58.2 Declaration

The available MLIB_ShLBiSat functions have the following declarations:

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

2.58.3 Function use

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

```c
#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);
}
```
2.59 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:

\[ MLIB_{\text{ShRBi}}(x, n) = x \gg n \]

Figure 83. Algorithm formula

2.59.1 Available versions

This function is available in the following versions:

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

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

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_ShRBi_F16</td>
<td>frac16_t</td>
<td>int16_t</td>
<td>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 &lt;-15; 15&gt;. The output is within the range &lt;-1; 1).</td>
</tr>
<tr>
<td>MLIB_ShRBi_F32</td>
<td>frac32_t</td>
<td>int16_t</td>
<td>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 &lt;-31; 31&gt;. The output is within the range &lt;-1; 1).</td>
</tr>
</tbody>
</table>

2.59.2 Declaration

The available MLIB_ShRBi functions have the following declarations:

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

2.59.3 Function use

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

```c
#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 */
```
2.60 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:

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

Figure 84. Algorithm formula

2.60.1 Available versions

This function is available in the following versions:

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

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

Table 61. Function versions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_ShRBiSat_F16</td>
<td>frac16_t</td>
<td>int16_t</td>
<td>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).</td>
</tr>
<tr>
<td>MLIB_ShRBiSat_F32</td>
<td>frac32_t</td>
<td>int16_t</td>
<td>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).</td>
</tr>
</tbody>
</table>

2.60.2 Declaration

The available MLIB_ShRBiSat functions have the following declarations:

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

2.60.3 Function use

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

```c
#include "mlib.h"

static frac32_t f32Result, f32Val;
static int16_t i16Sh;

void main(void)
```
2.61 MLIB_Sign

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

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

Figure 85. Algorithm formula

### 2.61.1 Available versions

This function is available in the following versions:

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

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

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Sign_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Sign of a 16-bit fractional value. The output is within the range (-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_Sign_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>Sign of a 32-bit fractional value. The output is within the range (-1 ; 1).</td>
</tr>
</tbody>
</table>

### 2.61.2 Declaration

The available **MLIB_Sign** functions have the following declarations:

```c
frac16_t MLIB_Sign_F16(frac16_t f16Val)
frac32_t MLIB_Sign_F32(frac32_t f32Val)
```

### 2.61.3 Function use

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

**Fixed-point version:**

```c
#include "mlib.h"

static frac32_t f32In, f32Result;
void main(void)
{
    f32In = FRAC32(-0.95);  /* f32In = -0.95 */
    f32Result = MLIB_ShRBiSat_F32(f32Val, i16Sh);
    /* f32Result = sat(f32Val >> i16Sh) */
}
```
2.62 MLIB_Sub

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

\[
\text{MLIB}_{\text{Sub}}(a, b) = a - b
\]

Figure 86. Algorithm formula

2.62.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The result may overflow.
- 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Sub_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Subtraction of a 16-bit fractional subtrahend from a 16-bit fractional minuend. The output is within the range &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_Sub_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>Subtraction of a 32-bit fractional subtrahend from a 32-bit fractional minuend. The output is within the range &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_Sub_A32ss</td>
<td>frac16_t</td>
<td>acc32_t</td>
<td>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 &lt;-65536 ; 65536).</td>
</tr>
<tr>
<td>MLIB_Sub_A32as</td>
<td>acc32_t</td>
<td>acc32_t</td>
<td>Subtraction of a 16-bit fractional subtrahend from a 32-bit accumulator. The output may be out of the range &lt;-65536 ; 65536).</td>
</tr>
</tbody>
</table>

2.62.2 Declaration

The available MLIB_Sub functions have the following declarations:

```c
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)
```

2.62.3 Function use

The use of the MLIB_Sub function is shown in the following examples:
Fixed-point version:

```c
#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);
}
```

2.63 MLIB_SubSat

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

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

Figure 87. Algorithm formula

2.63.1 Available versions

This function is available in the following versions:

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

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

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_SubSat_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Subtraction of a 16-bit fractional subtrahend from a 16-bit fractional minuend. The output is within the range (-1, 1).</td>
</tr>
<tr>
<td>MLIB_SubSat_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>Subtraction of a 32-bit fractional subtrahend from a 32-bit fractional minuend. The output is within the range (-1, 1).</td>
</tr>
</tbody>
</table>

2.63.2 Declaration

The available MLIB_SubSat functions have the following declarations:

```c
frac16_t MLIB_SubSat_F16(frac16_t f16Min, frac16_t f16Sub)
frac32_t MLIB_SubSat_F32(frac32_t f32Min, frac32_t f32Sub)
```
2.63.3 Function use

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

```c
#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);
}
```

2.64 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:

```
MLIB_Sub4(a, b, c, d) = a - b - c - d
```

Figure 88. Algorithm formula

2.64.1 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Sub4_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Subtraction of three 16-bit fractional subtrahends from 16-bit fractional minuend. The output is within the range &lt;-1 ; 1).</td>
</tr>
<tr>
<td>MLIB_Sub4_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>Subtraction of three 32-bit fractional subtrahends from 32-bit fractional minuend. The output is within the range &lt;-1 ; 1).</td>
</tr>
</tbody>
</table>

2.64.2 Declaration

The available MLIB_Sub4 functions have the following declarations:

```c
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)
```
2.64.3 Function use

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

```c
#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);
}
```

2.65 MLIB_Sub4Sat

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

\[
\text{MLIB\_Sub4\_Sat}(a, b, c, d) = \begin{cases} 
1, & a - b - c - d > 1 \\
-1, & a - b - c - d < -1 \\
a - b - c - d, & \text{else}
\end{cases}
\]

Figure 89. Algorithm formula

2.65.1 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>
<thead>
<tr>
<th>Function name</th>
<th>Input type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_Sub4_Sat_F16</td>
<td>frac16_t</td>
<td>frac16_t</td>
<td>Subtraction of three 16-bit fractional subtrahends from 16-bit fractional minuend. The output is within the range (-1 &lt; 1).</td>
</tr>
<tr>
<td>MLIB_Sub4_Sat_F32</td>
<td>frac32_t</td>
<td>frac32_t</td>
<td>Subtraction of three 32-bit fractional subtrahends from 32-bit fractional minuend. The output is within the range (-1 &lt; 1).</td>
</tr>
</tbody>
</table>
2.65.2 Declaration
The available MLIB_Sub4Sat functions have the following declarations:

\[
\begin{align*}
\text{frac16_t } & \text{ MLIB_Sub4Sat_F16}(\text{frac16_t } f16Min, \text{frac16_t } f16Sub1, \text{frac16_t } f16Sub2, \text{frac16_t } f16Sub3) \\
\text{frac32_t } & \text{ MLIB_Sub4Sat_F32}(\text{frac32_t } f32Min, \text{frac32_t } f32Sub1, \text{frac32_t } f32Sub2, \text{frac32_t } f32Sub3)
\end{align*}
\]

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

```c
#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:

```c
typedef unsigned short bool_t;
```

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

Table 67. Data storage

<table>
<thead>
<tr>
<th>Value</th>
<th>Unused</th>
<th>Logical</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FALSE</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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:

```c
typedef unsigned char uint8_t;
```

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

Table 68. Data storage

<table>
<thead>
<tr>
<th>Value</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>255</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td></td>
<td>F</td>
</tr>
</tbody>
</table>

Table continues on the next page...
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 \(0 \leq 65535\). Its definition is as follows:

\[
\text{typedef unsigned short uint16_t;}
\]

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

<table>
<thead>
<tr>
<th>Table 68. Data storage (continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
</tr>
<tr>
<td>124</td>
</tr>
<tr>
<td>159</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 69. Data storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
</tr>
<tr>
<td>65535</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>15518</td>
</tr>
<tr>
<td>40768</td>
</tr>
<tr>
<td>9</td>
</tr>
</tbody>
</table>
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 70. Data storage

<table>
<thead>
<tr>
<th>Value</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>4294967295</td>
<td>F F F F F F F F</td>
</tr>
<tr>
<td>2147483648</td>
<td>8 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>55977296</td>
<td>0 3 5 6 2 5 5 0</td>
</tr>
<tr>
<td>3451051828</td>
<td>C D B 2 D F 3 4</td>
</tr>
</tbody>
</table>

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 71. Data storage

<table>
<thead>
<tr>
<th>Value</th>
<th>Sign</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>127</td>
<td>0 1 1 1 1 1 1 1</td>
<td></td>
</tr>
<tr>
<td>-128</td>
<td>1 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>0 0 1 1 1 1 0 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>C</td>
</tr>
</tbody>
</table>

Table continues on the next page...
### 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:

```c
typedef short int16_t;
```

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

![Table 72. Data storage](image)

<table>
<thead>
<tr>
<th>Value</th>
<th>Sign</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>32767</td>
<td>0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 F</td>
<td></td>
</tr>
<tr>
<td>-32768</td>
<td>1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 F</td>
<td></td>
</tr>
<tr>
<td>15518</td>
<td>0 0 1 1 1 1 0 0 1 0 0 1 1 1 1 0</td>
<td></td>
</tr>
<tr>
<td>-24768</td>
<td>1 0 0 1 1 1 1 1 0 1 0 0 0 0 0 0</td>
<td></td>
</tr>
</tbody>
</table>

### 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:

```c
typedef long int32_t;
```

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

![Table 73. Data storage](image)

<table>
<thead>
<tr>
<th>Value</th>
<th>Sign</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>32767</td>
<td>0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 F</td>
<td></td>
</tr>
<tr>
<td>-32768</td>
<td>1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 F</td>
<td></td>
</tr>
<tr>
<td>15518</td>
<td>0 0 1 1 1 1 0 0 1 0 0 1 1 1 1 0</td>
<td></td>
</tr>
<tr>
<td>-24768</td>
<td>1 0 0 1 1 1 1 1 0 1 0 0 0 0 0 0</td>
<td></td>
</tr>
</tbody>
</table>

*Table continues on the next page...*
Table 73. Data storage (continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>31</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>2147483647</td>
<td>7</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>-2147483648</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>55977296</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>-843915468</td>
<td>C</td>
<td>D</td>
<td>B</td>
<td>2</td>
<td>D</td>
<td>F</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

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:

```c
typedef char frac8_t;
```

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

Table 74. Data storage

<table>
<thead>
<tr>
<th>Value</th>
<th>Sign</th>
<th>Fractional</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.99219</td>
<td>0 1 1 1 1 1 1 1</td>
<td></td>
</tr>
<tr>
<td>-1.0</td>
<td>1 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>0.46875</td>
<td>0 0 1 1 1 0 0 0</td>
<td></td>
</tr>
<tr>
<td>-0.75781</td>
<td>1 0 0 1 1 1 1 1</td>
<td></td>
</tr>
</tbody>
</table>

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:

```c
typedef short frac16_t;
```

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

Table 75. Data storage

<table>
<thead>
<tr>
<th>Value</th>
<th>Sign</th>
<th>Fractional</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9997</td>
<td>0</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>-1.0</td>
<td>1</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0.47357</td>
<td>0</td>
<td>0 1 1 1 1 0 0 1 0 0 1 1 1 1 0</td>
</tr>
<tr>
<td>-0.75586</td>
<td>1</td>
<td>0 0 1 1 1 1 1 0 1 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

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:

```c
typedef long frac32_t;
```

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

Table 76. Data storage

<table>
<thead>
<tr>
<th>Value</th>
<th>S</th>
<th>Fractional</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9999999995</td>
<td>7</td>
<td>F F F F F</td>
</tr>
</tbody>
</table>

To store a real number as frac32_t, use the FRAC32 macro.
Table 76. Data storage (continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.02606645970</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-0.3929787632</td>
<td>C</td>
<td>D</td>
<td>B</td>
<td>2</td>
<td>D</td>
<td>F</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:

```c
typedef short acc16_t;
```

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

Table 77. Data storage
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:

```c
typedef long acc32_t;
```

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

<table>
<thead>
<tr>
<th>Value</th>
<th>31</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>65535.999969</td>
<td>7</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>-65536.0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1.0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>23.789734</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>B</td>
<td>E</td>
<td>5</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>-1171.306793</td>
<td>F</td>
<td>D</td>
<td>B</td>
<td>6</td>
<td>5</td>
<td>8</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

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:

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

```c
#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:

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

```c
#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:

```c
#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>.

```c
#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:

```c
#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>.

```c
#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:

```c
#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}>.

```c
#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:

```c
#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>.

```c
#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:

```c
#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}>.

```c
#include "mlib.h"
static acc32_t a32Val;
void main(void)
{  
```
{ 
a32Val = ACC32(-13.654437);    /* a32Val = -13.654437 */
}

Limited warranty and liability — Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, ICODE, JOP, LIFE, VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFAREMOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, Altiview, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetics, Layerscape, Magniv, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Convergence, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQi, Vybrid, AirFast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-MD, ULINK-PLUS, ULINKpro, μVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org. M, M Mobileye and other Mobileye trademarks or logos appearing herein are trademarks of Mobileye Vision Technologies Ltd. in the United States, the EU and/or other jurisdictions.

© NXP B.V. 2021.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 01 November 2021
Document identifier: CM33MLIBUG