Document identifier: CM33GDFLIBUG
Rev. 5, 01 November 2021

GDFLIB User's Guide

ARM® Cortex® M33

Contents

Chapter 1
Library

1.1 Introduction

1.1.1 Overview

This user's guide describes the General Digital Filters Library (GDFLIB) for the family of ARM Cortex M33 core-based microcontrollers. This library contains optimized functions.

1.1.2 Data types

GDFLIB 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

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

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

where the function is compiled from four parts:

- MLIB—this is the library prefix
- Mac—the function name—Multiply-Accumulate
- F32—the function output type
- lss—the types of the function inputs; if all the inputs have the same type as the output, the inputs are not marked

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

Table 1. Input/output types

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

1.1.4 Supported compilers

GDFLIB 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 *gdflib.a* file.

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

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

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

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

1.1.5 Library configuration

GDFLIB 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 GDFLIB 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.

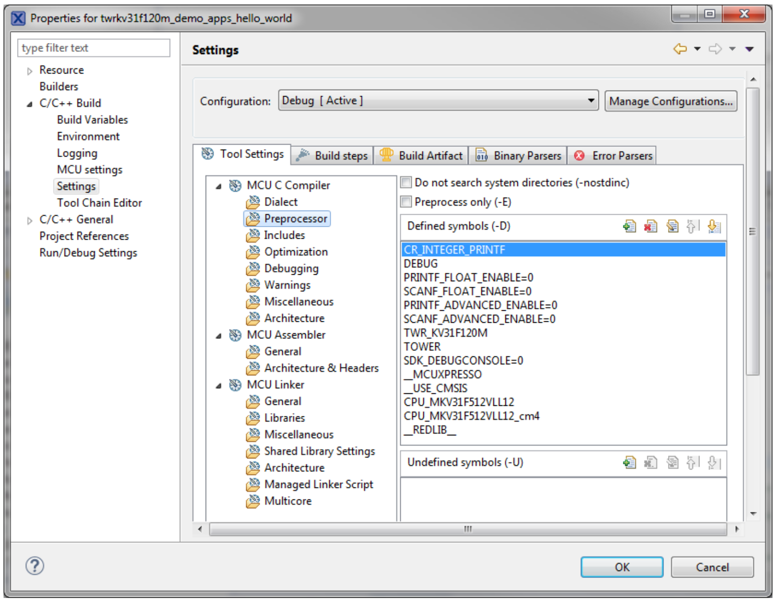

Figure 1. Defined symbols

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.

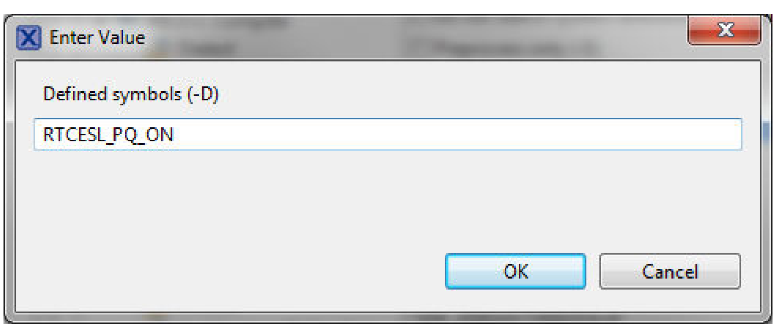

Figure 2. Symbol definition

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

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

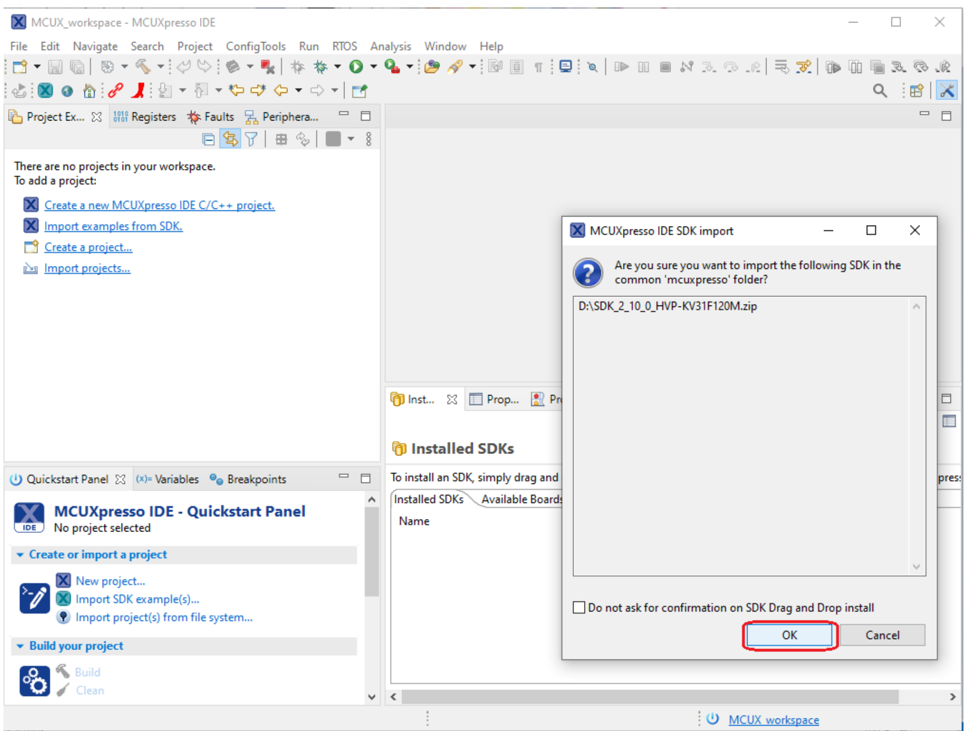

Figure 3. MCUXpresso IDE - imporing the SDK package to MCUXpresso IDE

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

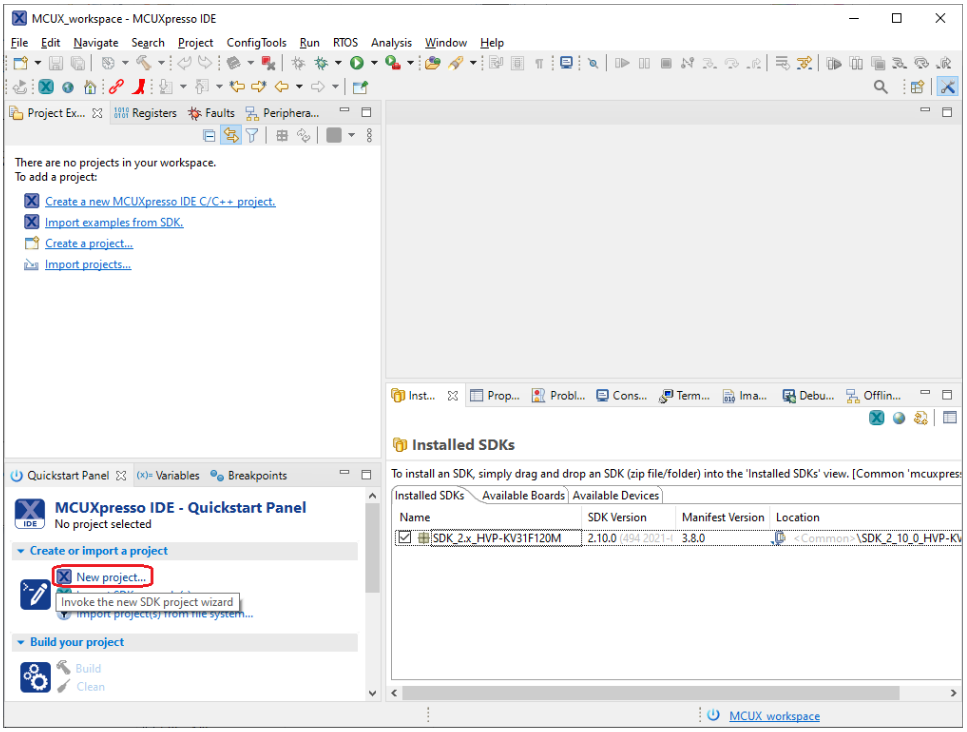

Figure 4. MCUXpresso IDE - create new project or Import SDK example(s)

Then select your board, and clik Next button.

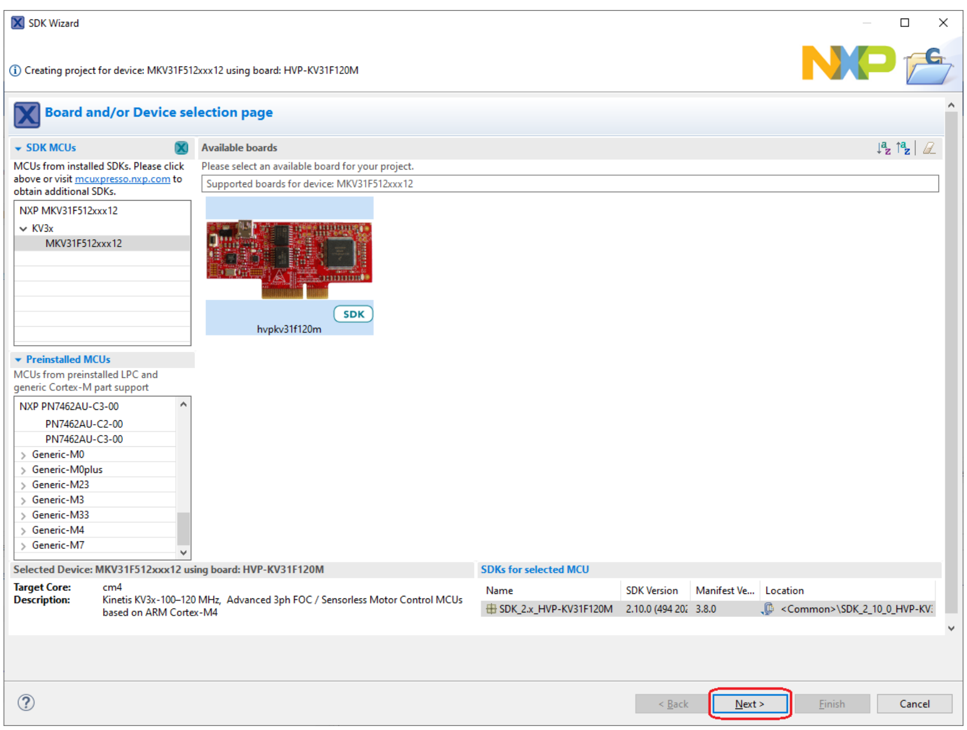

Figure 5. MCUXpresso IDE - selecting the board

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

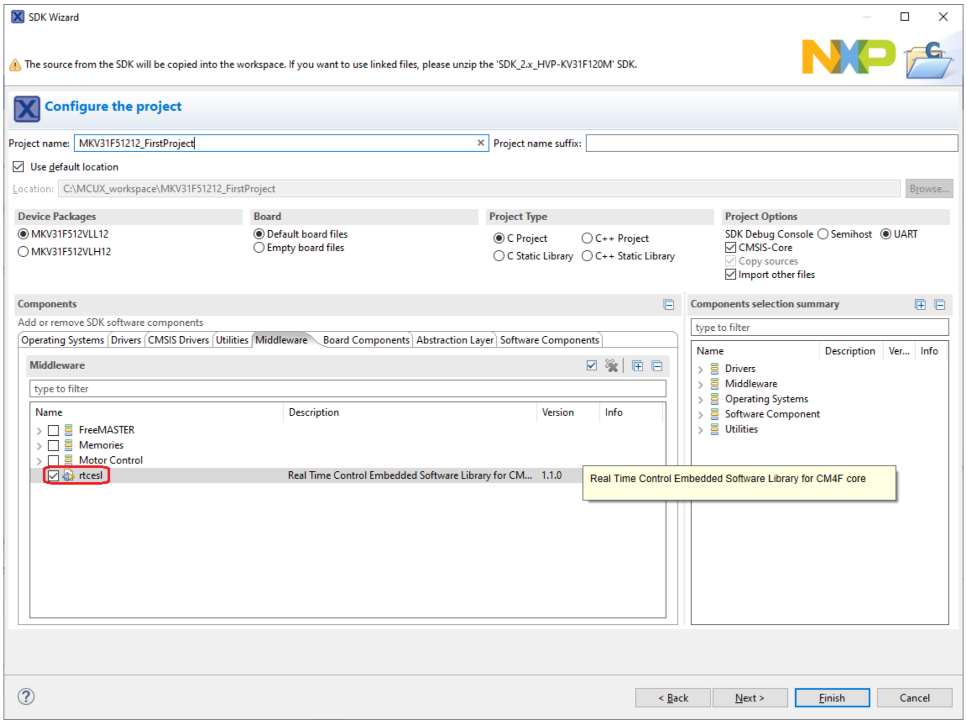

Figure 6. MCUXpresso IDE - selecting rtcesl component

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

```
#include "mlib.h"
#include "gdflib.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 GDFLIB 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.

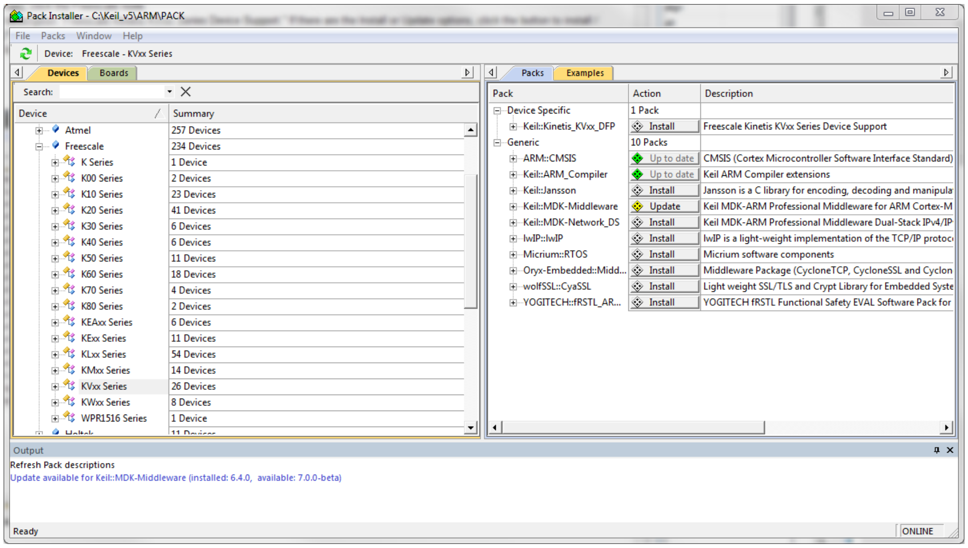

Figure 7. Pack Installer

New project (without MCUXpresso SDK)

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

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

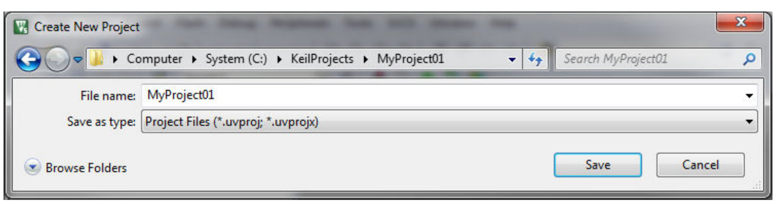

Figure 8. Create New Project dialog

4. In the next dialog, select the Software Packs in the very first box.
5. Type '' into the Search box, so that the device list is reduced to the devices.
6. Expand the node.
7. Click the LPC55s69 node, and then click OK. See Figure 9.

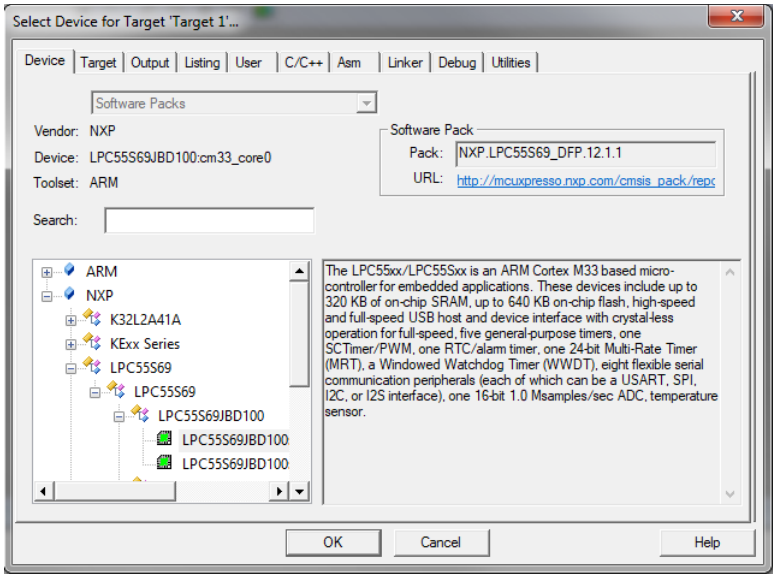

Figure 9. Select Device dialog

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

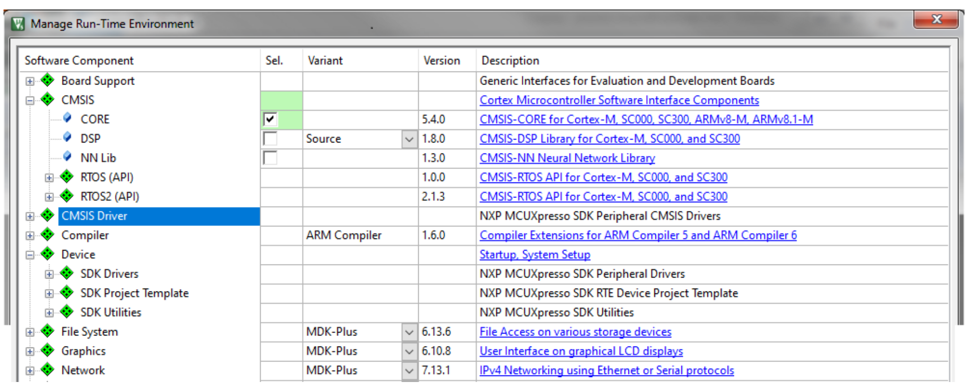

Figure 10. Manage Run-Time Environment dialog

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

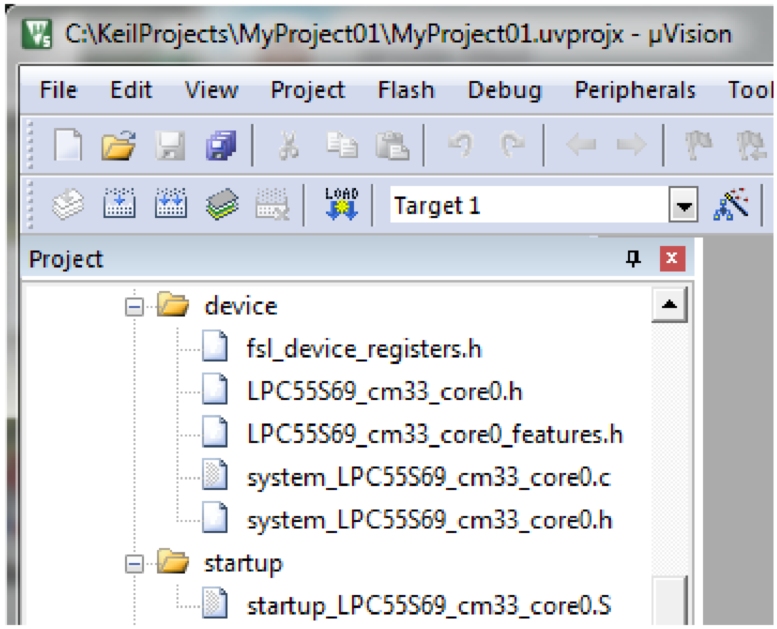

Figure 11. Project

11. In the main menu, go to Project > Options for Target 'Target1'…, and a dialog appears.
12. Select the Target tab.
13. Select Not Used in the Floating Point Hardware option. See Figure 11.

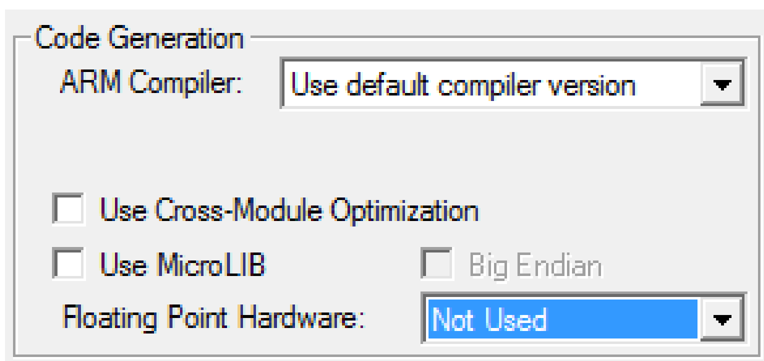

Figure 12. FPU

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.

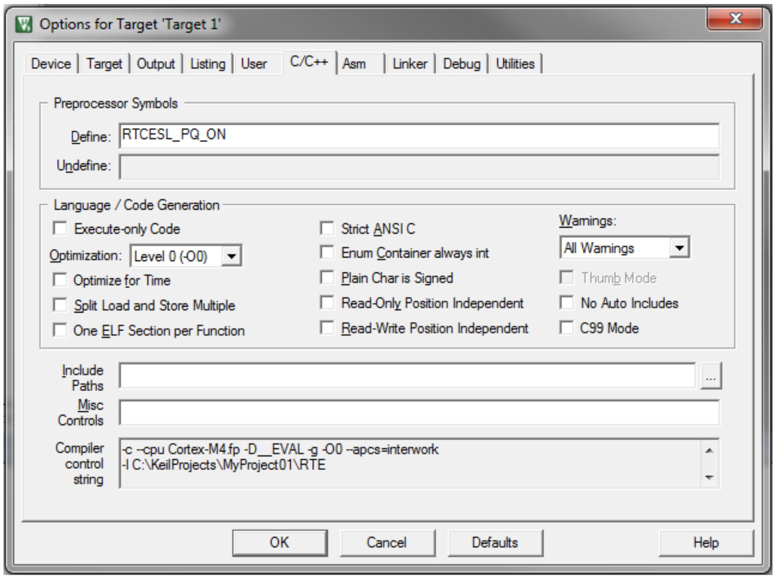

Figure 13. Preprocessor symbols

4. Click OK in the main dialog.
5. Ensure the PowerQuad moduel 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

GDFLIB requires MLIB to be included too. The following steps show how to include all dependent modules.

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.

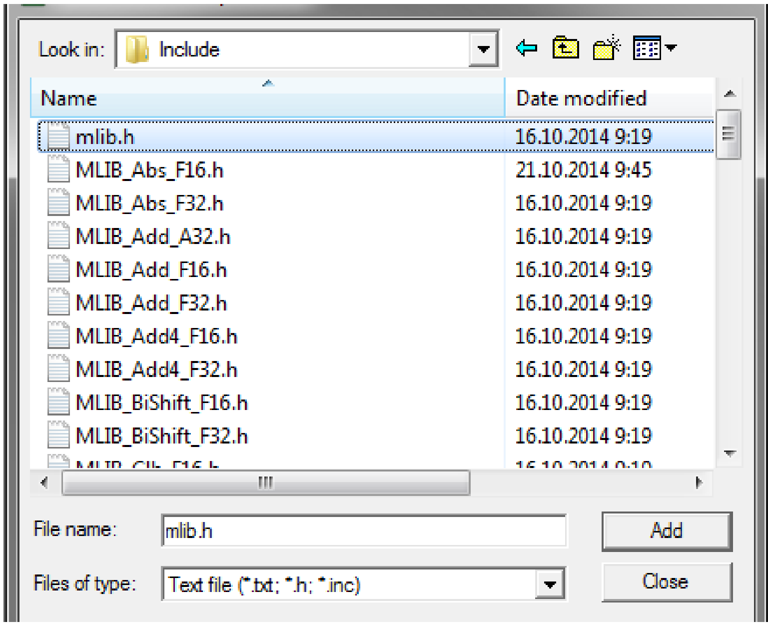

Figure 14. Adding .h files dialog

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.

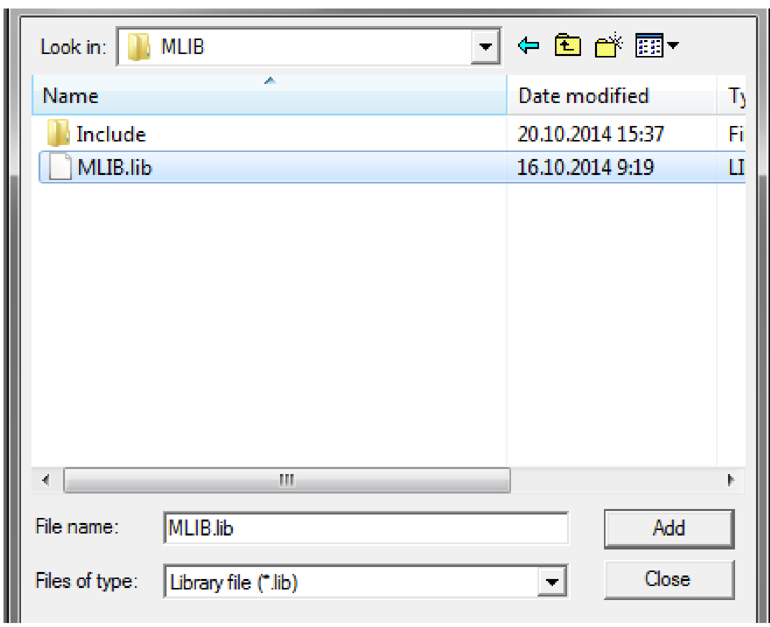

Figure 15. Adding .lib files dialog

6. Navigate into the library installation folder C:\NXP\RTCESL\CM33_RTCESL_4.7_KEIL\GDFLIB\Include, and select the *gdflib.h* file. If the file does not appear, set the Files of type filter to Text file. Click Add.
7. Navigate to the parent folder C:\NXP\RTCESL\CM33_RTCESL_4.7_KEIL\GDFLIB, and select the *gdflib.lib* file. If the file does not appear, set the Files of type filter to Library file. Click Add.
8. Now, all necessary files are in the project tree; see Figure 16. Click Close.

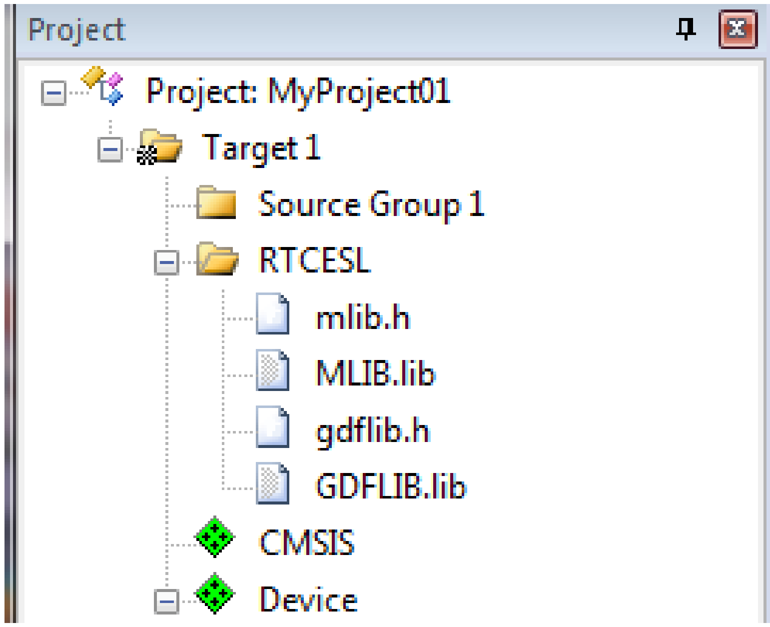

Figure 16. Project workspace

Library path setup

The following steps show the inclusion of all dependent modules.

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

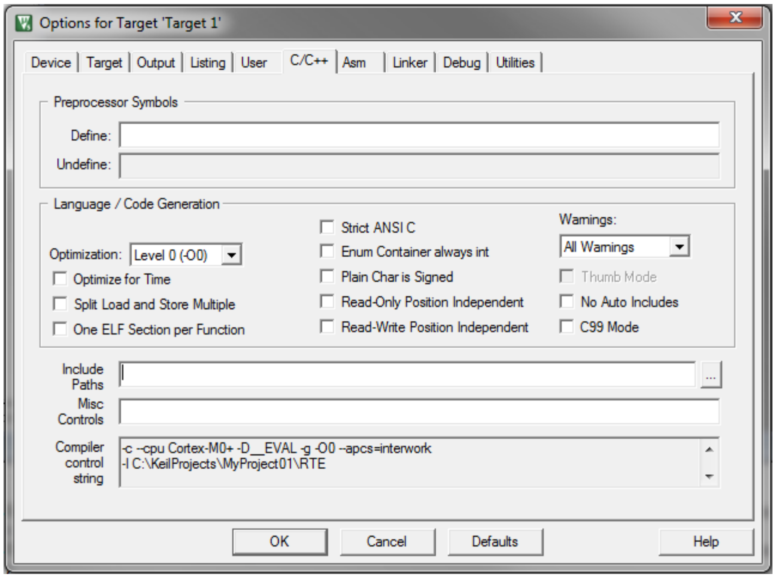

Figure 17. Library path addition

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

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

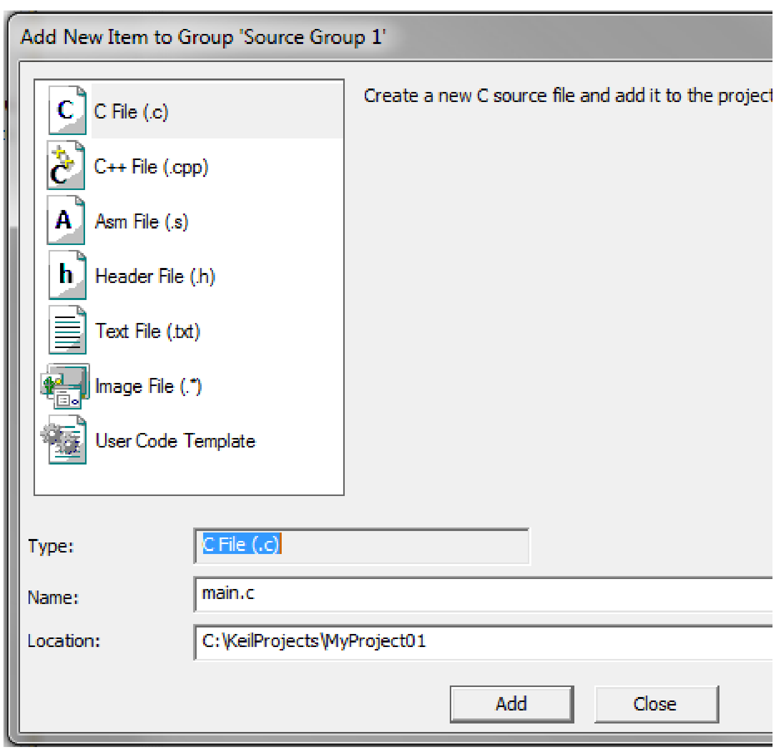

Figure 18. Adding new source file dialog

3. Click Add, and a new source file is created and opened up.

4. In the opened source file, include the following lines into the #include section, and create a main function:

```
#include "mlib.h"
#include "gdflib.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 GDFLIB 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.

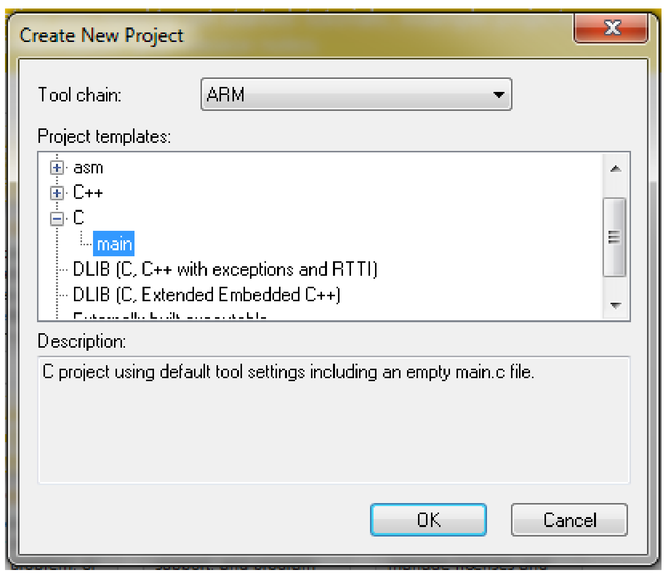

Figure 19. Create New Project dialog

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

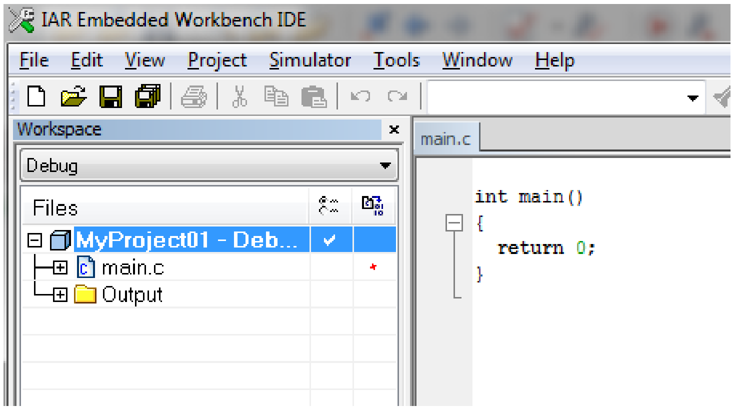

Figure 20. New project

5. In the main menu, go to Project > Options…, and a dialog appears.
6. In the Target tab, select the Device option, and click the button next to the dialog to select the MCU. In this example, select NXP > 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.

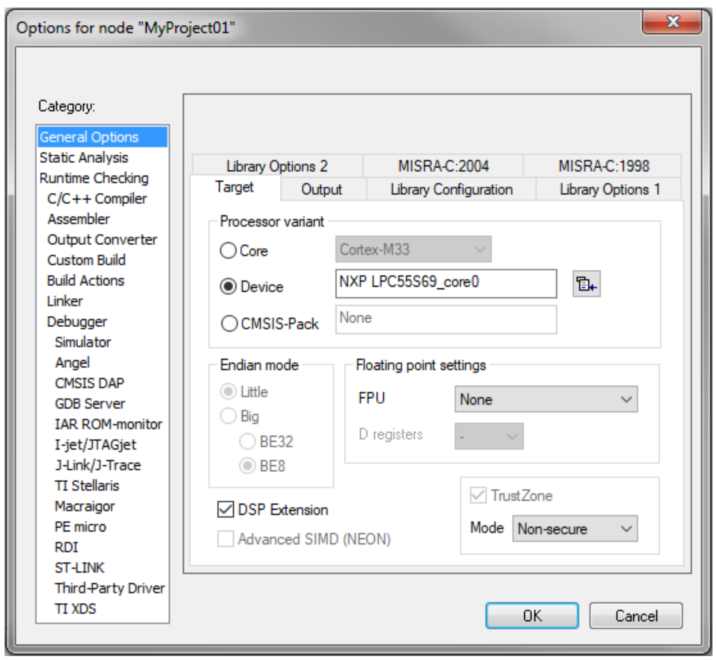

Figure 21. Options dialog

PowerQuad DSP Coprocessor and Accelerator support

Some LPC platforms (LPC55S6x) contain a hardware accelerator dedicated to common calculations in DSP applications. Only functions runing 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.

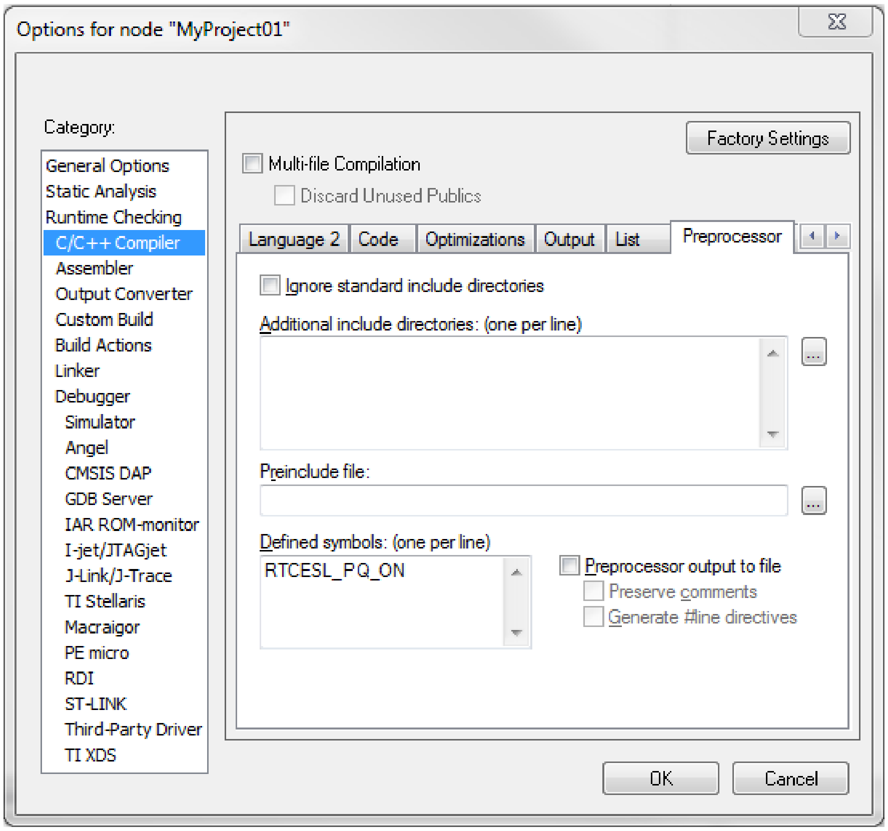

Figure 22. Defined symbols

5. Click OK in the main dialog.
6. Ensure the PowerQuad moduel 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.

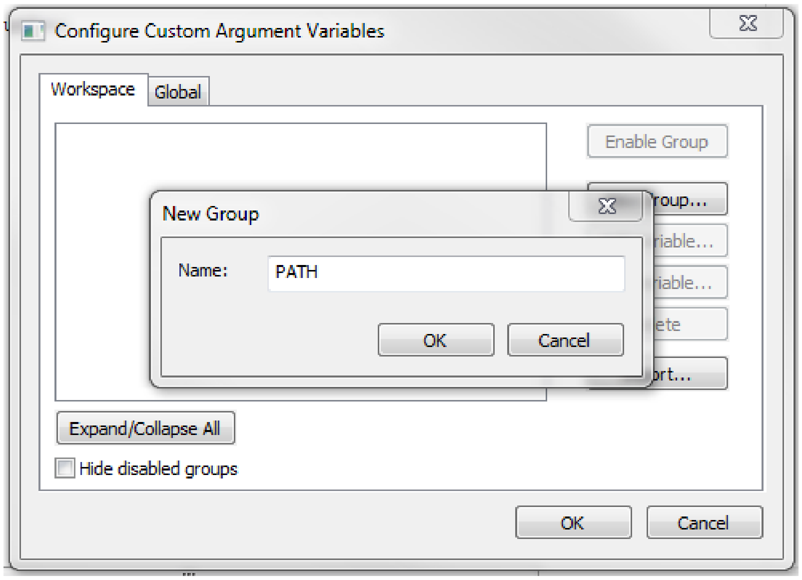

Figure 23. New Group

3. Click on the newly created group, and click the Add Variable button. A dialog appears.
4. Type this name: RTCESL_LOC
5. To set up the value, look for the library by clicking the '…' button, or just type the installation path into the box: C:\NXP\RTCESL\CM33_RTCESL_4.7_IAR. Click OK.
6. In the main dialog, click OK. See Figure 24.

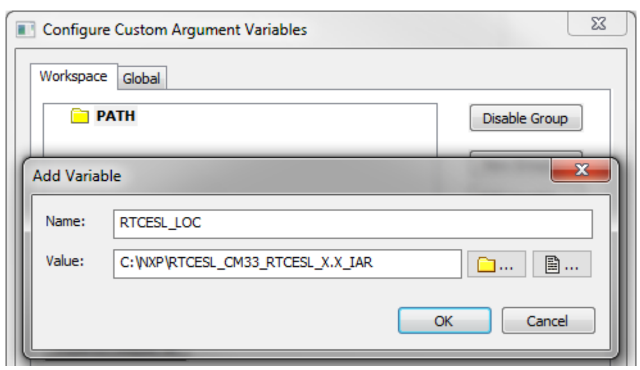

Figure 24. New variable

Linking the files into the project

GDFLIB requires MLIB to be included too. The following steps show the inclusion of all dependent modules.

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.

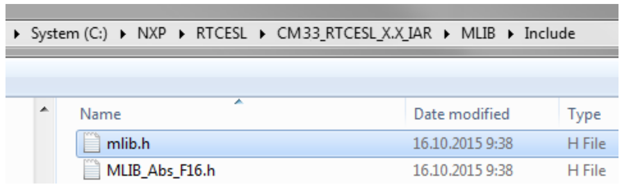

Figure 25. Add Files dialog

7. Click on the RTCESL node, go to Project > Add Group…, and create a GDFLIB subgroup.
8. Click on the newly created node GDFLIB, and go to the main menu Project > Add Files….
9. Navigate into the library installation folder C:\NXP\RTCESL\CM33_RTCESL_4.7_IAR\GDFLIB\Include, and select the *gdflib.h* file. (If the file does not appear, set the file-type filter to Source Files.) Click Open.
10. Navigate into the library installation folder C:\NXP\RTCESL\CM33_RTCESL_4.7_IAR\GDFLIB, and select the *gdflib.a* file. If the file does not appear, set the file-type filter to Library / Object files. Click Open.
11. Now you will see the files added in the workspace. See Figure 26.

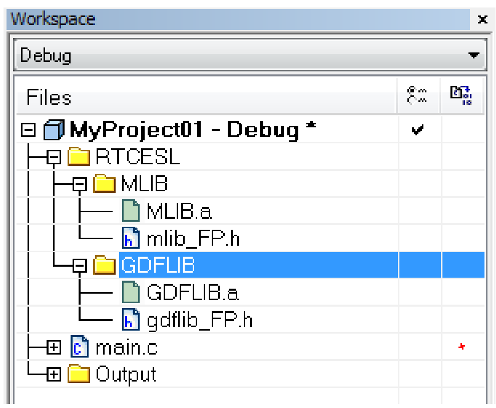

Figure 26. Project workspace

Library path setup

The following steps show the inclusion of all dependent modules:

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
   - $RTCESL_LOC$\GDFLIB\Include
5. Click OK in the main dialog. See Figure 27.

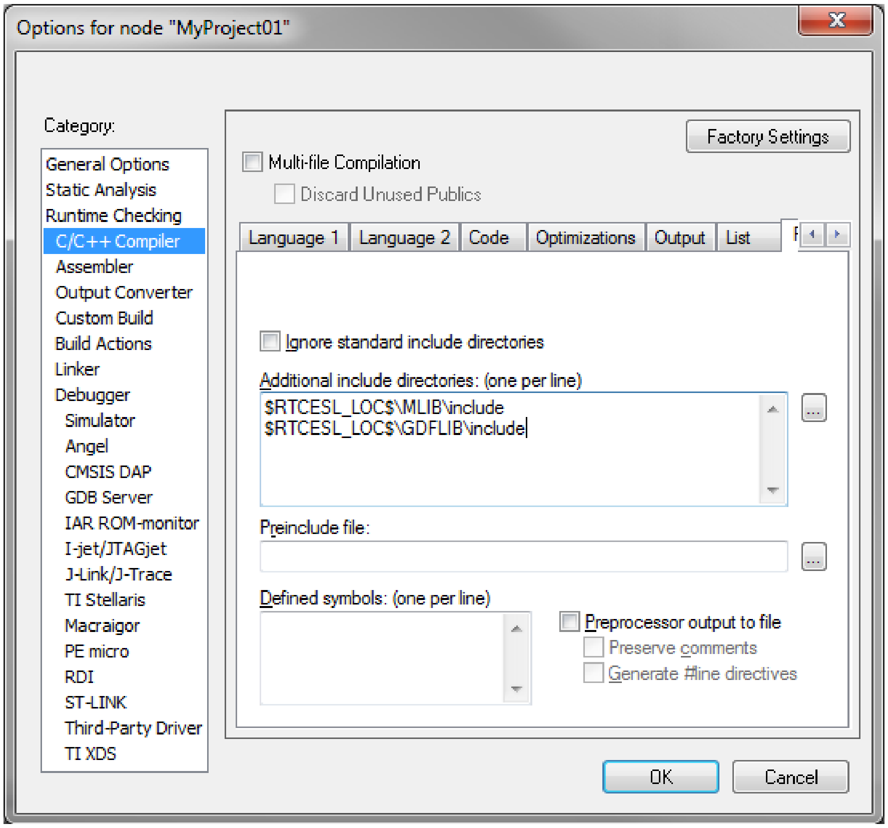

Figure 27. Library path adition

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

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

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

Chapter 2
Algorithms in detail

2.1 GDFLIB_FilterExp

The GDFLIB_FilterExp function calculates the exponential smoothing. The exponential filter is the simplest filter with only one tuning parameter, requiring to store only one variable - the filter output (it is used in the next step). For a proper use, it is recommended that the algorithm is initialized by the GDFLIB_FilterExpInit function, before using the GDFLIB_FilterExp function.

The filter calculation consists of the following equation:

$$y(k) = y(k-1) + A \cdot (x(k) - (k-1))$$

Figure 28.

where:

- x(k) is the actual value of the input signal
- y(k) is the actual filter output
- A is the filter constant (0 ; 1) (it defines the smoothness of the exponential filter)

The exponential filter tuning is based on these rules: for a small value of the filter constant there is a strong filtering effect (if A = 0 then the output equals the new input). For a high value of the filtering constant, there is a weak filtering effect (if A = 1 then the new input is ignored). The filter constant defines the ratio between the filter inputs and the last step output, used for the next calculation.

2.1.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The parameter uses the fraction type.

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

Table 2. Init function versions

Function name	Input type	Parameters	Result type	Description
GDFLIB_FilterExpInit_F16	frac16_t	GDFLIB_FILTER_EXP_T_F32*	void	The input argument is a 16-bit fractional value that represents the initial value of the filter at the current step. The input is within the range <-1 ; 1). The parameters' structure is pointed to by a pointer.

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

Table 3. Function versions

Function name	Input type	Parameters	Result type	Description
GDFLIB_FilterExp_F16	frac16_t	GDFLIB_FILTER_EXP_T_F32 *	frac16_t	The input argument is a 16-bit fractional value of the input signal to be filtered within the range <-1 ; 1). The parameters' structure is pointed to by a pointer. The function returns a 16-bit fractional value within the range <-1 ; 1).

2.1.2 GDFLIB_FILTER_EXP_T_F32

Variable name	Input type	Description
f32A	frac32_t	Filter constant value (filter parameter). It defines the smoothness of the exponential filter (high value = small filtering effect, low value = strong filtering effect). It is usually defined as: $$A = 1 - exp^{-\frac{T_S}{\tau}}$$ Where T_s is the sample time and τ is the filter time constant. The parameter is a 32-bit fractional value within the range <-0 ; 1). Set by the user.
f32AccK_1	frac32_t	Filter accumulator (last step output) value. The parameter is a 32-bit accumulator type within the range <-1.0 ; 1.0). Controlled by the algorithm.

2.1.3 Declaration

The available GDFLIB_FilterExpInit functions have the following declarations:

```
void GDFLIB_FilterExpInit_F16(frac16_t f16InitVal, GDFLIB_FILTER_EXP_T_F32 *psParam)
```

The available GDFLIB_FilterExp functions have the following declarations:

```
frac16_t GDFLIB_FilterExp_F16(frac16_t f16InX, GDFLIB_FILTER_EXP_T_F32 *psParam)
```

2.1.4 Function use

The use of the GDFLIB_FilterExpInit and GDFLIB_FilterExp functions is shown in the following examples:

Fixed-point version:

```
#include "gdflib.h"

static frac16_t f16Result;
static frac16_t f16InitVal, f16InX;
static GDFLIB_FILTER_EXP_T_F32 sFilterParam;

void Isr(void);

void main(void)
{
   f16InitVal = FRAC16(0.0);                 /* f16InitVal = 0.0 */

   /* Filter constant = 0.05 */
   sFilterParam.f32A = FRAC32(0.05);

   GDFLIB_FilterExpInit_F16(f16InitVal, &sFilterParam);
```

```
    f16InX =  FRAC16(0.5);
}

/* periodically called function */
void Isr(void)
{
    f16Result = GDFLIB_FilterExp_F16(f16InX, &sFilterParam);
}
```

2.2 GDFLIB_FilterIIR1

This function calculates the first-order direct form 1 IIR filter.

For a proper use, it is recommended that the algorithm is initialized by the GDFLIB_FilterIIR1Init function, before using the GDFLIB_FilterIIR1 function. The GDFLIB_FilterIIR1Init function initializes the buffer and coefficients of the first-order IIR filter.

The GDFLIB_FilterIIR1 function calculates the first-order infinite impulse response (IIR) filter. The IIR filters are also called recursive filters, because both the input and the previously calculated output values are used for calculation. This form of feedback enables the transfer of energy from the output to the input, which leads to an infinitely long impulse response (IIR). A general form of the IIR filter, expressed as a transfer function in the Z-domain, is described as follows:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}}$$

Figure 29.

where N denotes the filter order. The first-order IIR filter in the Z-domain is expressed as follows:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1}}$$

Figure 30.

which is transformed into a time-domain difference equation as follows:

$$y(k) = b_0 x(k) + b_1 x(k-1) - a_1 y(k-1)$$

Figure 31.

The filter difference equation is implemented in the digital signal controller directly, as given in Equation 3; this equation represents a direct-form 1 first-order IIR filter, as shown in Figure 32.

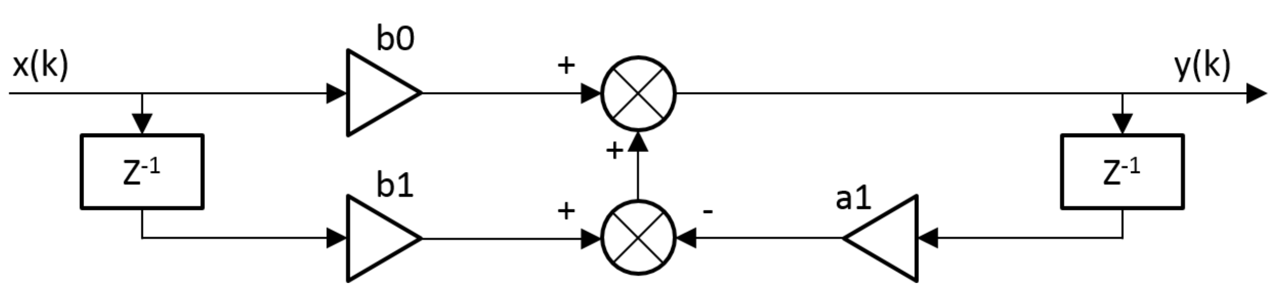

Figure 32. Direct form 1 first-order IIR filter

The coefficients of the filter shown in Figure 3-1 can be designed to meet the requirements for the first-order low-pass filter (LPF) or high-pass filter (HPF). The coefficient quantization error is not important in the case of a first-order filter due to a finite precision arithmetic. A higher-order LPF or HPF can be obtained by connecting a number of first-order filters in series. The number of connections gives the order of the resulting filter.

The filter coefficients must be defined before calling this function. As some coefficients can be greater than 1 (and lesser than 2), the coefficients are scaled down (divided) by 2.0 for the fractional version of the algorithm. For faster calculation, the A coefficient is sign-inverted. The function returns the filtered value of the input in the step k, and stores the input and the output values in the step k into the filter buffer.

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 <-1 ; 1).

The available versions of the GDFLIB_FilterIIR1Init function are shown in the following table:

Table 4. Init function versions

Function name	Parameters	Result type	Description
GDFLIB_FilterIIR1Init_F16	GDFLIB_FILTER_IIR1_T_F32*	void	Filter initialization (reset) function. The parameters' structure is pointed to by a pointer.

The available versions of the GDFLIB_FilterIIR1 function are shown in the following table:

Table 5. Function versions

Function name	Input type	Parameters	Result type	Description
GDFLIB_FilterIIR1_F16	frac16_t	GDFLIB_FILTER_IIR1_T_F32*	frac16_t	The input argument is a 16-bit fractional value of the input signal to be filtered within the range <-1 ; 1). The parameters' structure is pointed to by a pointer. The function returns a 16-bit fractional value within the range <-1 ; 1).

2.2.2 GDFLIB_FILTER_IIR1_T_F32

Variable name	Input type	Description
sFltCoeff	GDFLIB_FILTER_IIR1_COEFF_T_F32*	Substructure containing filter coefficients.
f32FltBfrY[1]	frac32_t	Internal buffer of y-history. Controlled by the algorithm.
f16FltBfrX[1]	frac16_t	Internal buffer of x-history. Controlled by the algorithm.

2.2.3 GDFLIB_FILTER_IIR1_COEFF_T_F32

Variable name	Type	Description
f32B0	frac32_t	B0 coefficient of the IIR1 filter. Set by the user, and must be divided by 2.
f32B1	frac32_t	B1 coefficient of the IIR1 filter. Set by the user, and must be divided by 2.
f32A1	frac32_t	A1 (sign-inverted) coefficient of the IIR1 filter. Set by the user, and must be divided by -2 (negative two).

2.2.4 Declaration

The available GDFLIB_FilterIIR1Init functions have the following declarations:

```
void GDFLIB_FilterIIR1Init_F16(GDFLIB_FILTER_IIR1_T_F32 *psParam)
```

The available GDFLIB_FilterIIR1 functions have the following declarations:

```
frac16_t GDFLIB_FilterIIR1_F16(frac16_t f16InX, GDFLIB_FILTER_IIR1_T_F32 *psParam)
```

2.2.5 Calculation of filter coefficients

There are plenty of methods for calculating the coefficients. The following example shows the use of Matlab to set up a low-pass filter with the 500 Hz sampling frequency, and 240 Hz stopped frequency with a 20 dB attenutation. Maximum passband ripple is 3 dB at the cut-off frequency of 50 Hz.

```
% sampling frequency 500 Hz, low pass
Ts = 1 / 500

% cut-off frequency 50 Hz
Fc = 50

% max. passband ripple 3 dB
Rp = 3

% stopped frequency 240Hz
Fs = 240

% attenuation 20 dB
Rs = 20

% checking order of the filter
n = buttord(2 * Ts * Fc, 2 * Ts * Fs, Rp, Rs)
% n = 1, i.e. the filter is achievable with the 1st order

% getting the filter coefficients
[b, a] = butter(n, 2 * Ts * Fc, 'low');

% the coefs are:
% b0 = 0.245237275252786, b1 = 0.245237275252786
% a0 = 1.0000, a1 = -0.509525449494429
```

The filter response is shown in Figure 33.

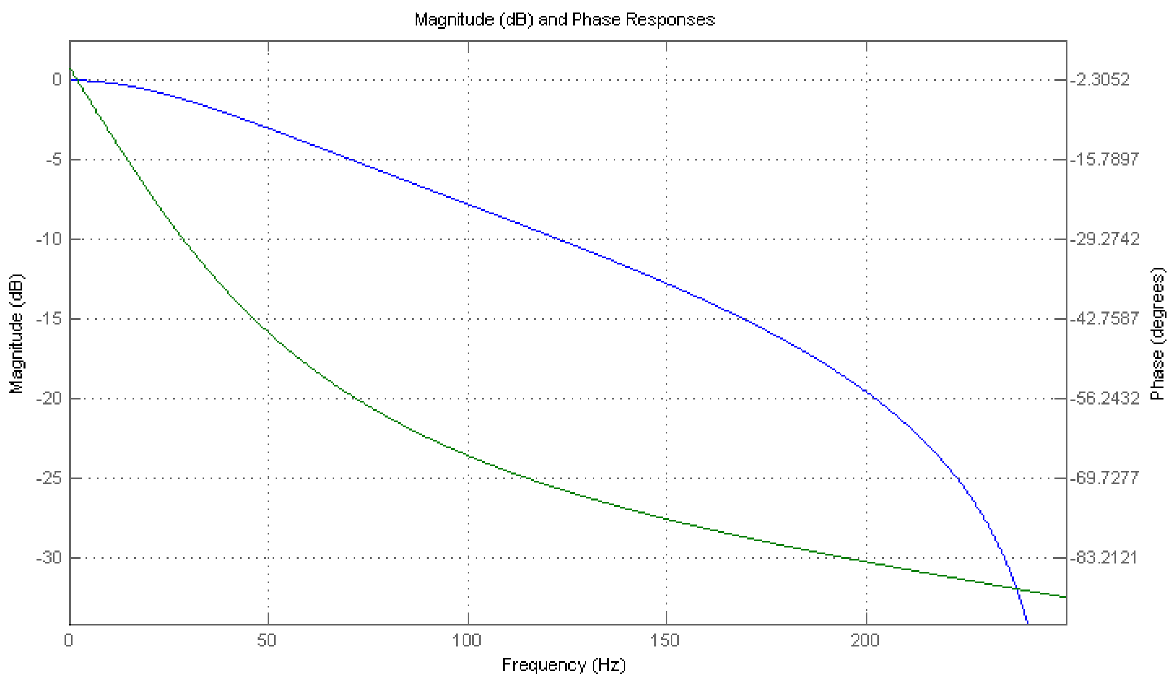

Figure 32. Filter response

2.2.6 Function use

The use of the GDFLIB_FilterIIR1Init and GDFLIB_FilterIIR1 functions is shown in the following examples. The filter uses the above-calculated coefficients:

Fixed-point version:

```
#include "gdflib.h"

static frac16_t f16Result;
static frac16_t f16InX;
static GDFLIB_FILTER_IIR1_T_F32 sFilterParam;

void Isr(void);

void main(void)
{
   sFilterParam.sFltCoeff.f32B0 = FRAC32(0.245237275252786 / 2.0);
   sFilterParam.sFltCoeff.f32B1 = FRAC32(0.245237275252786 / 2.0);
   sFilterParam.sFltCoeff.f32A1 = FRAC32(-0.509525449494429 / -2.0);

   GDFLIB_FilterIIR1Init_F16(&sFilterParam);

   f16InX = FRAC16(0.1);
}

/* periodically called function */
void Isr(void)
{
```

```
    f16Result = GDFLIB_FilterIIR1_F16(f16InX, &sFilterParam);
}
```

2.3 GDFLIB_FilterIIR2

This function calculates the second-order direct-form 1 IIR filter.

For a proper use, it is recommended that the algorithm is initialized by the GDFLIB_FilterIIR2Init function, before using the GDFLIB_FilterIIR2 function. The GDFLIB_FilterIIR2Init function initializes the buffer and coefficients of the second-order IIR filter.

The GDFLIB_FilterIIR2 function calculates the second-order infinite impulse response (IIR) filter. The IIR filters are also called recursive filters, because both the input and the previously calculated output values are used for calculation. This form of feedback enables the transfer of energy from the output to the input, which leads to an infinitely long impulse response (IIR). A general form of the IIR filter, expressed as a transfer function in the Z-domain, is described as follows:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}}$$

Figure 33.

where N denotes the filter order. The second-order IIR filter in the Z-domain is expressed as follows:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

Figure 34.

which is transformed into a time-domain difference equation as follows:

$$y(k) = b_0 x(k) + b_1 x(k-1) + b_2 x(k-2) - a_1 y(k-1) - a_2 y(k-2)$$

Figure 35.

The filter difference equation is implemented in the digital signal controller directly, as given in Equation 3; this equation represents a direct-form 1 second-order IIR filter, as depicted in Figure 36.

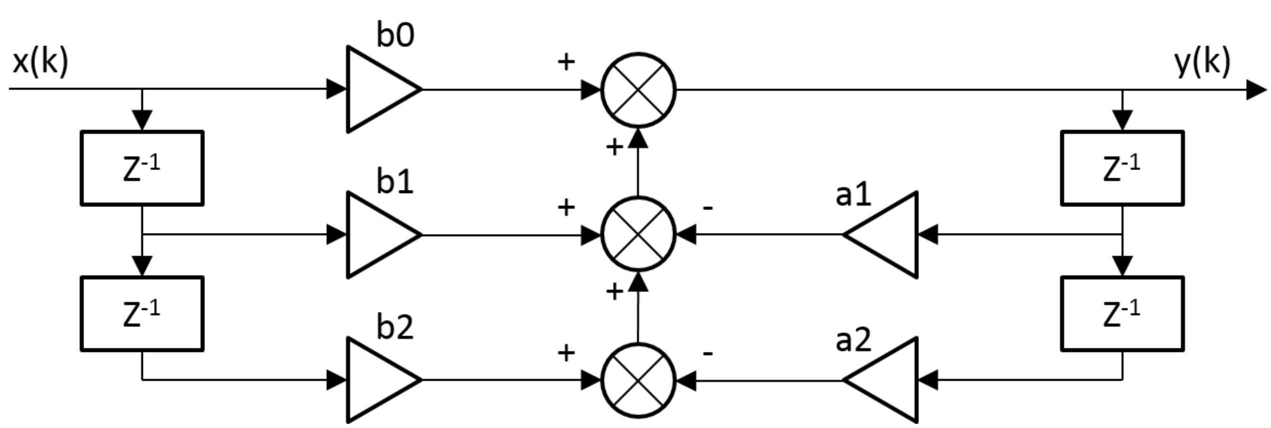

Figure 36. Direct-form 1 second-order IIR filter

The coefficients of the filter depicted in Figure 3-1 can be designed to meet the requirements for the second-order low-pass filter (LPF), high-pass filter (HPF), band-pass filter (BPF) or band-stop filter (BSF). The coefficient quantization error can be neglected in the case of a second-order filter due to a finite precision arithmetic. A higher-order LPF or HPF can be obtained by connecting a number of second-order filters in series. The number of connections gives the order of the resulting filter.

The filter coefficients must be defined before calling this function. As some coefficients can be greater than 1 (and lesser than 2), the coefficients are scaled down (divided) by 2.0 for the fractional version of the algorithm. For faster calculation, the A coefficients are sign-inverted. The function returns the filtered value of the input in the step k, and stores the input and output values in the step k into the filter buffer.

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 available versions of the GDFLIB_FilterIIR2Init function are shown in the following table:

Table 6. Init function versions

Function name	Parameters	Result type	Description
GDFLIB_FilterIIR2Init_F16	GDFLIB_FILTER_IIR2_T_F32*	void	Filter initialization (reset) function. The parameters' structure is pointed to by a pointer. If PowerQuad based function used the Init function must be called prior to FilterIIR2_F16 function to transfer IIR2 parameters from fraction to float, without the Init function required parameters will not be used for the IIR2 calculations.

The available versions of the GDFLIB_FilterIIR2 function are shown in the following table:

Table 7. Function versions

Function name	Input type	Parameters	Result type	Description
GDFLIB_FilterIIR2_F16	frac16_t	GDFLIB_FILTER_IIR2_T_F32*	frac16_t	Input argument is a 16-bit fractional value of the input signal to be filtered within the range <-1 ; 1). The parameters' structure is pointed to by a pointer. The function returns a 16-bit fractional value within the range <-1 ; 1). If PowerQuad based function used the Init function must be called prior to FilterIIR2_F16 function to transfer IIR2 parameters from fraction to float, without the Init function required parameters will not be used for the IIR2 calculations.

2.3.2 GDFLIB_FILTER_IIR2_T_F32

Variable name	Input type	Description
sFltCoeff	GDFLIB_FILTER_IIR2_COEFF_T_F32*	Substructure containing filter coefficients.
f32FltBfrY[2]	frac32_t	Internal buffer of y-history. Controlled by the algorithm.
f16FltBfrX[2]	frac16_t	Internal buffer of x-history. Controlled by the algorithm.

2.3.3 GDFLIB_FILTER_IIR2_COEFF_T_F32

Variable name	Type	Description
f32B0	frac32_t	B0 coefficient of the IIR2 filter. Set by the user, and must be divided by 2.
f32B1	frac32_t	B1 coefficient of the IIR2 filter. Set by the user, and must be divided by 2.
f32B2	frac32_t	B2 coefficient of the IIR2 filter. Set by the user, and must be divided by 2.
f32A1	frac32_t	A1 (sign-inverted) coefficient of the IIR2 filter. Set by the user, and must be divided by -2 (negative two).
f32A2	frac32_t	A2 (sign-inverted) coefficient of the IIR2 filter. Set by the user, and must be divided by -2 (negative two).

2.3.4 Declaration

The available GDFLIB_FilterIIR2Init functions have the following declarations:

```
void GDFLIB_FilterIIR2Init_F16(GDFLIB_FILTER_IIR2_T_F32 *psParam)
```

The available GDFLIB_FilterIIR2 functions have the following declarations:

```
frac16_t GDFLIB_FilterIIR2_F16(frac16_t f16InX, GDFLIB_FILTER_IIR2_T_F32 *psParam)
```

2.3.5 Calculation of filter coefficients

There are plenty of methods for calculating the coefficients. The following example shows the use of Matlab to set up a stopband filter with the 1000 Hz sampling frequency, 100 Hz stop frequency with 10 dB attenuation, and 30 Hz bandwidth. Maximum passband ripple is 3 dB.

```
% sampling frequency 1000 Hz, stop band
Ts = 1 / 1000

% center stop frequency 100 Hz
Fc = 50

% attenuation 10 dB
Rs = 10

% bandwidth 30 Hz
Fbw = 30

% max. passband ripple 3 dB
Rp = 3

% checking order of the filter
n = buttord(2 * Ts * [Fc - Fbw /2 Fc + Fbw / 2], 2 * Ts * [Fc - Fbw Fc + Fbw], Rp, Rs)
% n = 2, i.e. the filter is achievable with the 2nd order

% getting the filter coefficients
[b, a] = butter(n / 2, 2 * Ts * [Fc - Fbw /2 Fc + Fbw / 2], 'stop')

% the coefs are:
```

```
% b0 = 0.913635972986238, b1 = -1.745585863109291, b2 = 0.913635972986238
% a0 = 1.0000, a1 = -1.745585863109291, a2 = 0.827271945972476
```

The filter response is shown in Figure 37.

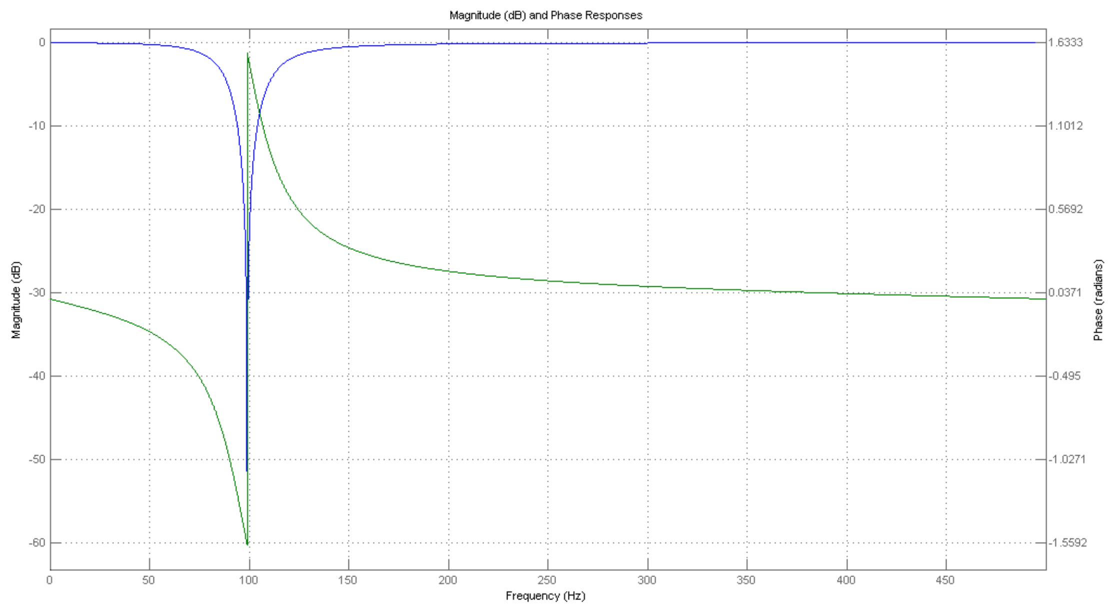

Figure 36. Filter response

2.3.6 Function use

The use of the GDFLIB_FilterIIR2Init and GDFLIB_FilterIIR2 functions is shown in the following examples. The filter uses the above-calculated coefficients:

Fixed-point version:

```
#include "gdflib.h"

static frac16_t f16Result;
static frac16_t f16InX;
static GDFLIB_FILTER_IIR2_T_F32 sFilterParam;

void Isr(void);

void main(void)
{
   sFilterParam.sFltCoeff.f32B0 = FRAC32(0.913635972986238 / 2.0);
   sFilterParam.sFltCoeff.f32B1 = FRAC32(-1.745585863109291 / 2.0);
   sFilterParam.sFltCoeff.f32B2 = FRAC32(0.913635972986238 / 2.0);
   sFilterParam.sFltCoeff.f32A1 = FRAC32(-1.745585863109291 / -2.0);
   sFilterParam.sFltCoeff.f32A2 = FRAC32(0.827271945972476 / -2.0);

   GDFLIB_FilterIIR2Init_F16(&sFilterParam);

   f16InX = FRAC16(0.1);
}
```

```
/* periodically called function */
void Isr(void)
{
     f16Result = GDFLIB_FilterIIR2_F16(f16InX, &sFilterParam);
}
```

2.4 GDFLIB_FilterIIR3

This function calculates the third-order direct-form 1 IIR filter.

For a proper use, it is recommended to initialize the algorithm by the GDFLIB_FilterIIR3Init function before using the GDFLIB_FilterIIR3 function. The GDFLIB_FilterIIR3Init function initializes the buffer and coefficients of the third-order IIR filter.

The GDFLIB_FilterIIR3 function calculates the third-order infinite impulse response (IIR) filter. The IIR filters are also called recursive filters because both the input and the previously calculated output values are used for calculation. This form of feedback enables the transfer of energy from the output to the input, which leads to an infinitely long impulse response (IIR). A general form of the IIR filter (expressed as a transfer function in the Z-domain) is described as follows:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}}$$

Figure 37.

where N denotes the filter order. The third-order IIR filter in the Z-domain is expressed as follows:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}}{1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}}$$

Figure 38.

which is transformed into a time-domain difference equation as follows:

$$y(k) = b_0 x(k) + b_1 x(k-1) + b_2 x(k-2) + b_3 x(k-3) - a_1 y(k-1) - a_2 y(k-2) - a_3 y(k-3)$$

Figure 39.

The filter difference equation is implemented in the digital signal controller directly, as given in Equation 3. This equation represents a direct-form 1 third-order IIR filter, as depicted in Figure 40.

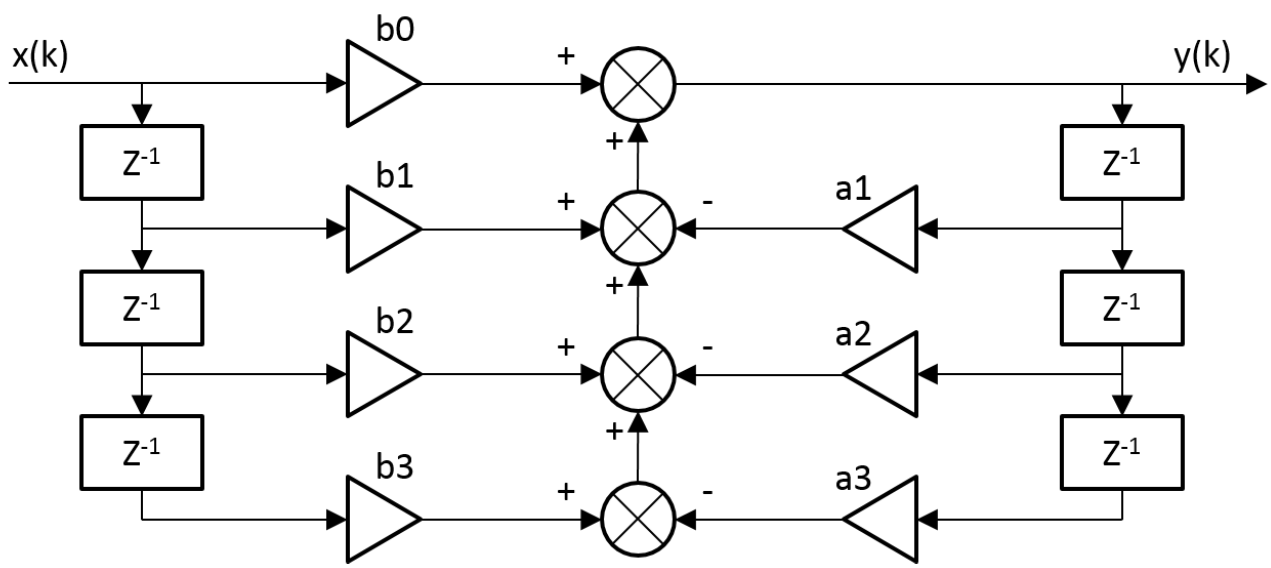

Figure 40. Direct-form 1 third-order IIR filter

The coefficients of the filter depicted in Figure 3-1 can be designed to meet the requirements for the third-order low-pass filter (LPF) or high-pass filter (HPF). The coefficient quantization error can be neglected in the case of a third-order filter due to a finite precision arithmetic. A higher-order LPF or HPF can be obtained by connecting a number of third-order filters in series. The number of connections gives the order of the resulting filter.

Define the filter coefficients before calling this function. As some coefficients can be greater than 1 (and lesser than 4), the coefficients are scaled down (divided) by 4.0 for the fractional version of the algorithm. For a faster calculation, the A coefficients are sign-inverted. The function returns the filtered value of the input in the step k, and stores the input and output values in the step k into the filter buffer.

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 available versions of the GDFLIB_FilterIIR3Init function are shown in the following table:

Table 8. Init function versions

Function name	Parameters	Result type	Description
GDFLIB_FilterIIR3Init_F16	GDFLIB_FILTER_IIR3_T_F32*	void	Filter initialization (reset) function. The parameters' structure is pointed to by a pointer.

The available versions of the GDFLIB_FilterIIR3 function are shown in the following table:

Table 9. Function versions

Function name	Input type	Parameters	Result type	Description
GDFLIB_FilterIIR3_F16	frac16_t	GDFLIB_FILTER_IIR3_T_F32*	frac16_t	Input argument is a 16-bit fractional value of the input signal to be filtered within the range <-1 ; 1). The parameters' structure

Table continues on the next page...

Table 9. Function versions

Function name	Input type	Parameters	Result type	Description
				is pointed to by a pointer. The function returns a 16-bit fractional value within the range <-1 ; 1).

2.4.2 GDFLIB_FILTER_IIR3_T_F32

Variable name	Input type	Description
sFltCoeff	GDFLIB_FILTER_IIR3_COEFF_T_F32*	Substructure containing filter coefficients.
f32FltBfrY[3]	frac32_t	Internal buffer of y-history. Controlled by the algorithm.
f16FltBfrX[3]	frac16_t	Internal buffer of x-history. Controlled by the algorithm.

2.4.3 GDFLIB_FILTER_IIR3_COEFF_T_F32

Variable name	Type	Description
f32B0	frac32_t	B0 coefficient of the IIR3 filter. Set by the user, and must be divided by 4.
f32B1	frac32_t	B1 coefficient of the IIR3 filter. Set by the user, and must be divided by 4.
f32B2	frac32_t	B2 coefficient of the IIR3 filter. Set by the user, and must be divided by 4.
f32B3	frac32_t	B3 coefficient of the IIR3 filter. Set by the user, and must be divided by 4 (negative four).
f32A1	frac32_t	A1 (sign-inverted) coefficient of the IIR3 filter. Set by the user. Must be divided by -4 (negative four).
f32A2	frac32_t	A2 (sign-inverted) coefficient of the IIR3 filter. Set by the user. Must be divided by -4 (negative four).
f32A3	frac32_t	A3 (sign-inverted) coefficient of the IIR3 filter. Set by the user. Must be divided by -4 (negative four).

2.4.4 Declaration

The available GDFLIB_FilterIIR3Init functions have the following declarations:

```
void GDFLIB_FilterIIR3Init_F16(GDFLIB_FILTER_IIR3_T_F32 *psParam)
```

The available GDFLIB_FilterIIR3 functions have the following declarations:

```
frac16_t GDFLIB_FilterIIR3_F16(frac16_t f16InX, GDFLIB_FILTER_IIR3_T_F32 *psParam)
```

2.4.5 Calculation of filter coefficients

There are plenty of methods for calculating the coefficients. The following example shows the use of Matlab to set up a high-pass filter with the 10000 Hz sampling frequency and 200 Hz stop frequency with 60 dB attenuation. The ripple is 3 dB at the cut-off frequency of 2000 Hz.

```
% sampling frequency 10000 Hz, high pass
Ts = 1 / 10000

% cut-off frequency 2 KHz
Fc = 2000

% attenuation 60 dB
Rs = 60

% stop frequency 200 Hz
Fs = 200

% max. passband ripple 3 dB
Rp = 3

% checking order of the filter
n = buttord(2 * Ts * Fc, 2 * Ts * Fs, Rp, Rs)
% n = 3, i.e. the filter is achievable with the 3rd order

% getting the filter coefficients
[b, a] = butter(n, 2* Ts * Fc, 'high')

% the coefs are:
% b0 = 0.256915601248463, b1 = -0.770746803745390, b2 = 0.770746803745390,
% b3 = -0.256915601248463
% a0 = 1.0000, a1 = -0.577240524806303, a2 = 0.421787048689562, a3 = -0.056297236491843
```

The filter response is shown in Figure 41.

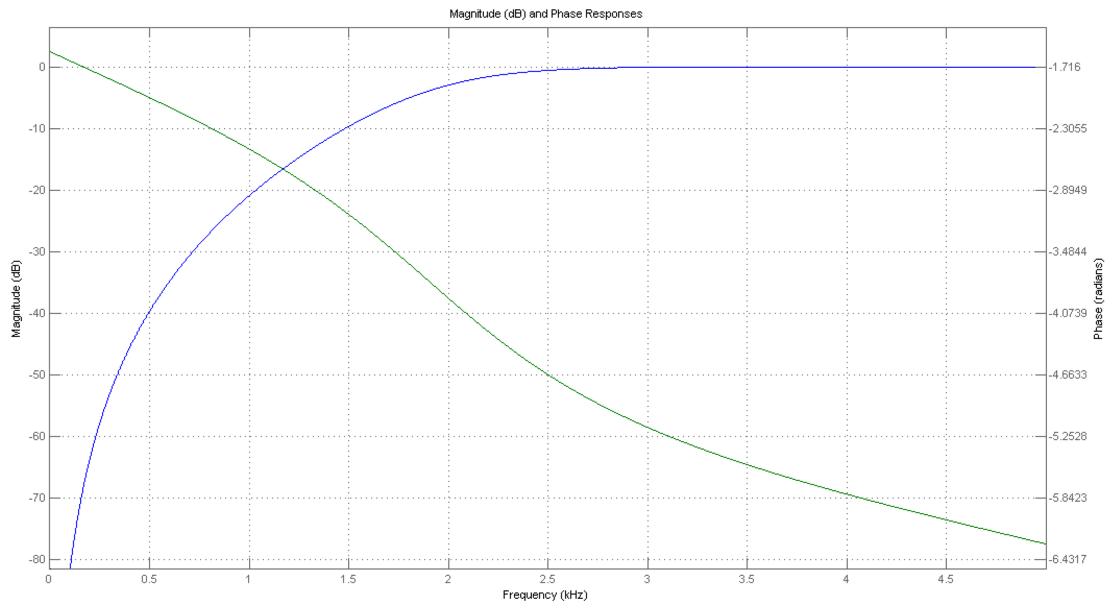

Figure 40. Filter response

2.4.6 Function use

The use of the GDFLIB_FilterIIR3Init and GDFLIB_FilterIIR3 functions is shown in the following examples. The filter uses the above-calculated coefficients:

Fixed-point version:

```
#include "gdflib.h"

static frac16_t f16Result;
static frac16_t f16InX;
static GDFLIB_FILTER_IIR3_T_F32 sFilterParam;

void Isr(void);

void main(void)
{
   sFilterParam.sFltCoeff.f32B0 = FRAC32(0.256915601248463 / 4.0);
   sFilterParam.sFltCoeff.f32B1 = FRAC32(-0.770746803745390 / 4.0);
   sFilterParam.sFltCoeff.f32B2 = FRAC32(0.770746803745390 / 4.0);
   sFilterParam.sFltCoeff.f32B3 = FRAC32(-0.256915601248463 / 4.0);
   sFilterParam.sFltCoeff.f32A1 = FRAC32(-0.577240524806303 / -4.0);
   sFilterParam.sFltCoeff.f32A2 = FRAC32(0.421787048689562 / -4.0);
   sFilterParam.sFltCoeff.f32A3 = FRAC32(-0.056297236491843 / -4.0);

   GDFLIB_FilterIIR3Init_F16(&sFilterParam);

   f16InX = FRAC16(0.1);
}

/* periodically called function */
void Isr(void)
```

```
{
     f16Result = GDFLIB_FilterIIR3_F16(f16InX, &sFilterParam);
}
```

2.5 GDFLIB_FilterIIR4

This function calculates the fourth-order direct-form 1 IIR filter.

For a proper use, it is recommended to initialize the algorithm by the GDFLIB_FilterIIR4Init function, before using the GDFLIB_FilterIIR4 function. The GDFLIB_FilterIIR4Init function initializes the buffer and coefficients of the fourth-order IIR filter.

The GDFLIB_FilterIIR4 function calculates the fourth-order infinite impulse response (IIR) filter. The IIR filters are also called recursive filters, because both the input and the previously calculated output values are used for calculation. This form of feedback enables the transfer of energy from the output to the input, which leads to an infinitely long impulse response (IIR). A general form of the IIR filter (expressed as a transfer function in the Z-domain) is described as follows:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}}$$

Figure 41.

where N denotes the filter order. The fourth-order IIR filter in the Z-domain is expressed as follows:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + b_4 z^{-4}}{1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + a_4 z^{-4}}$$

Figure 42.

which is transformed into a time-domain difference equation as follows:

$$y(k) = b_0 x(k) + b_1 x(k-1) + b_2 x(k-2) + b_3 x(k-3) + b_4 x(k-4) - a_1 y(k-1) - a_2 y(k-2) - a_3 y(k-3) - a_4 y(k-4)$$

Figure 43.

The filter difference equation is implemented directly in the digital signal controller, as given in Equation 3; this equation represents a direct-form 1 fourth-order IIR filter, as shown in Figure 44.

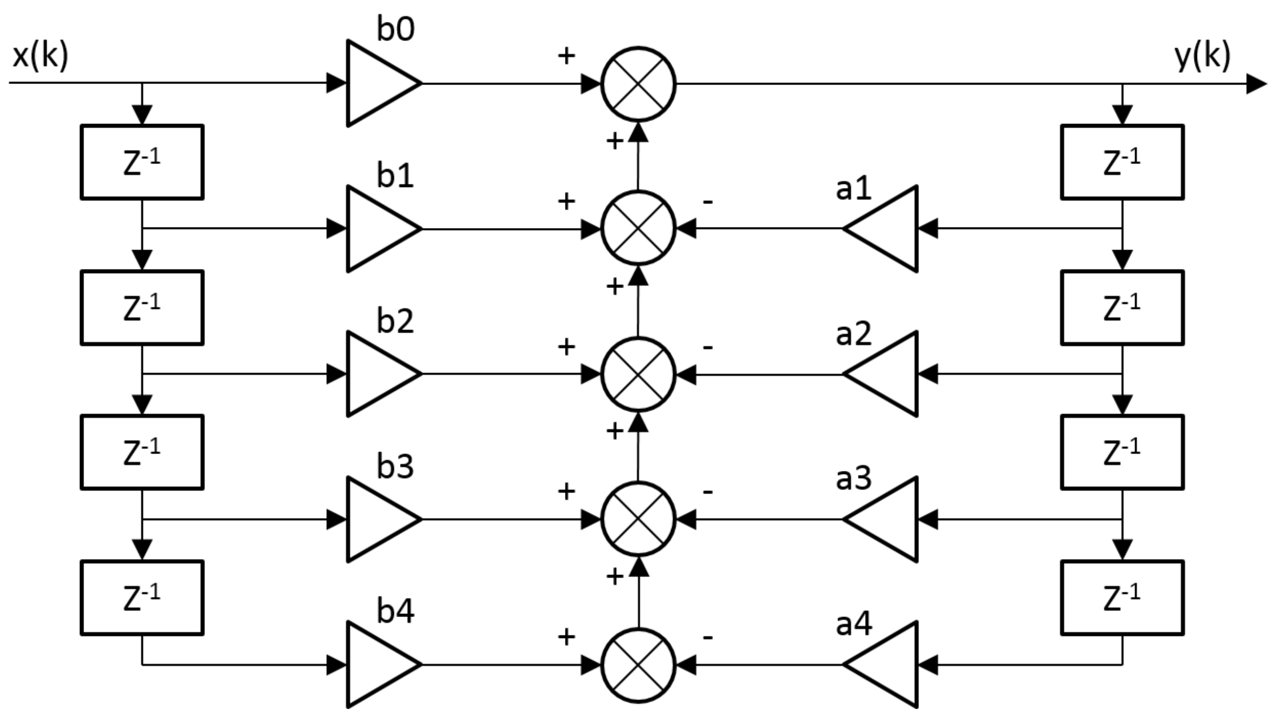

Figure 44. Direct-form 1 fourth-order IIR filter

The coefficients of the filter shown in Figure 3-1 can be designed to meet the requirements for the fourth-order low-pass filter (LPF), high-pass filter (HPF), band-pass filter (BPF), or band-stop filter (BSF). The coefficient quantization error can be ignored in the case of a fourth-order filter due to a finite precision arithmetic. A higher-order LPF or HPF can be obtained by connecting a number of fourth-order filters in series. The number of connections gives the order of the resulting filter.

Define the filter coefficients before calling this function. As some coefficients can be greater than 1 (and lesser than 8), the coefficients are scaled down (divided) by 8.0 for the fractional version of the algorithm. For a faster calculation, the A coefficients are sign-inverted. The function returns the filtered value of the input in step k, and stores the input and output values in the step k into the filter buffer.

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 available versions of the GDFLIB_FilterIIR4Init function are shown in the following table:

Table 10. Init function versions

Function name	Parameters	Result type	Description
GDFLIB_FilterIIR4Init_F16	GDFLIB_FILTER_IIR4_T_F32*	void	Filter initialization (reset) function. The parameters' structure is pointed to by a pointer.

The available versions of the GDFLIB_FilterIIR4 function are shown in the following table:

Table 11. Function versions

Function name	Input type	Parameters	Result type	Description
GDFLIB_FilterIIR4_F16	frac16_t	GDFLIB_FILTER_IIR4_T_F32*	frac16_t	Input argument is a 16-bit fractional value of the input signal to be filtered within the range <-1 ; 1). The parameters' structure is pointed to by a pointer. The function returns a 16-bit fractional value within the range <-1 ; 1).

2.5.2 GDFLIB_FILTER_IIR4_T_F32

Variable name	Input type	Description
sFltCoeff	GDFLIB_FILTER_IIR4_COEFF_T_F32*	Substructure containing filter coefficients.
f32FltBfrY[4]	frac32_t	Internal buffer of y-history. Controlled by the algorithm.
f16FltBfrX[4]	frac16_t	Internal buffer of x-history. Controlled by the algorithm.

2.5.3 GDFLIB_FILTER_IIR4_COEFF_T_F32

Variable name	Type	Description
f32B0	frac32_t	B0 coefficient of the IIR4 filter. Set by the user, and must be divided by 8.
f32B1	frac32_t	B1 coefficient of the IIR4 filter. Set by the user, and must be divided by 8.
f32B2	frac32_t	B2 coefficient of the IIR4 filter. Set by the user, and must be divided by 8.
f32B3	frac32_t	B3 coefficient of the IIR4 filter. Set by the user, and must be divided by 8.
f32B4	frac32_t	B4 coefficient of the IIR4 filter. Set by the user, and must be divided by 8.
f32A1	frac32_t	A1 (sign-inverted) coefficient of the IIR4 filter. Set by the user, and must be divided by -8 (negative eight).
f32A2	frac32_t	A2 (sign-inverted) coefficient of the IIR4 filter. Set by the user, and must be divided by -8 (negative eight).
f32A3	frac32_t	A3 (sign-inverted) coefficient of the IIR4 filter. Set by the user, and must be divided by -8 (negative eight).
f32A4	frac32_t	A4 (sign-inverted) coefficient of the IIR4 filter. Set by the user, and must be divided by -8 (negative eight).

2.5.4 Declaration

The available GDFLIB_FilterIIR4Init functions have the following declarations:

```
void GDFLIB_FilterIIR4Init_F16(GDFLIB_FILTER_IIR4_T_F32 *psParam)
```

The available GDFLIB_FilterIIR4 functions have the following declarations:

```
frac16_t GDFLIB_FilterIIR4_F16(frac16_t f16InX, GDFLIB_FILTER_IIR4_T_F32 *psParam)
```

2.5.5 Calculation of filter coefficients

There are plenty of methods for the coefficients calculation. The following example shows the use of Matlab to set up a band-pass filter with the 10000 Hz sampling frequency, 1000 Hz pass frequency, and 250 Hz bandwidth. The maximum passband ripple is 3 dB, and the attenuation is 20 dB.

```
% sampling frequency 10000 Hz, band pass
Ts = 1 / 10000

% center pass frequency 2000 Hz
Fc = 2000

% attenuation 20 dB
Rs = 20

% bandwidth 250 Hz
Fbw = 250

% max. passband ripple 3 dB
Rp = 3

% checking order of the filter
n = buttord(2 * Ts * [Fc - Fbw /2 Fc + Fbw / 2], 2 * Ts * [Fc - Fbw Fc + Fbw], Rp, Rs)
% n = 4, i.e. the filter is achievable with the 4th order

% getting the filter coefficients
[b, a] = butter(n / 2, 2 * Ts * [Fc - Fbw /2 Fc + Fbw / 2])

% the coefs are:
% b0 = 0.005542717210281, b1 = 0, b2 = -0.011085434420561, b3 = 0, b4 =
0.005542717210281
% a0 = 1.0000, a1 = -1.171272075750262, a2 = 2.122554479822350, a3 =
-1.047780658093187,
% a4 = 0.800802646665706
```

The filter response is shown in Figure 45.

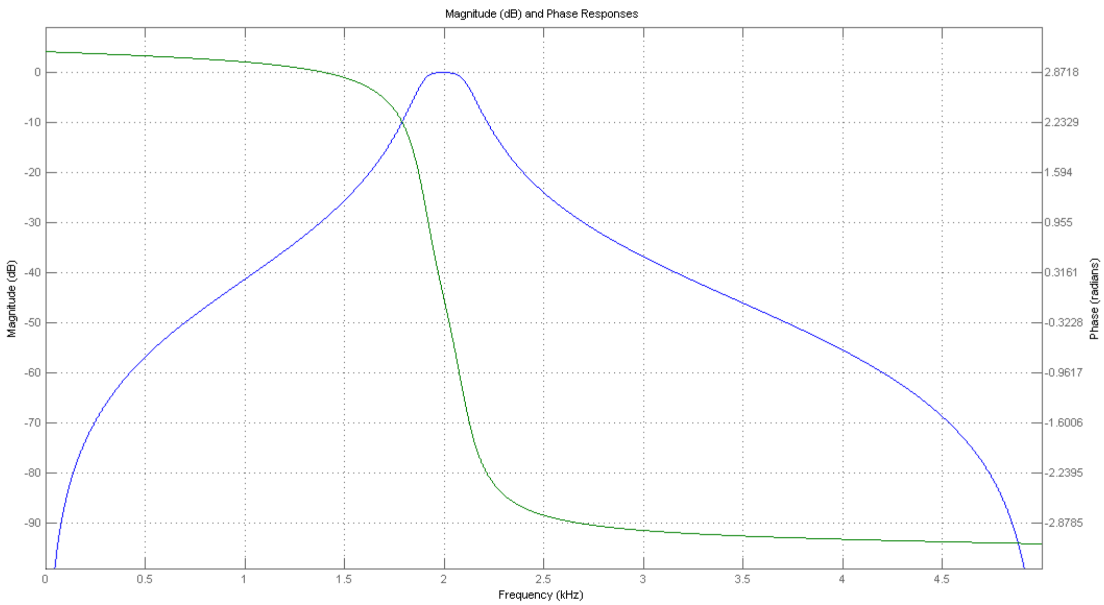

Figure 44. Filter response

2.5.6 Function use

The use of the GDFLIB_FilterIIR4Init and GDFLIB_FilterIIR4 functions is shown in the following examples. The filter uses the above-calculated coefficients:

Fixed-point version:

```
#include "gdflib.h"

static frac16_t f16Result;
static frac16_t f16InX;
static GDFLIB_FILTER_IIR4_T_F32 sFilterParam;

void Isr(void);

void main(void)
{
   sFilterParam.sFltCoeff.f32B0 = FRAC32(0.005542717210281 / 8.0);
   sFilterParam.sFltCoeff.f32B1 = FRAC32(0.0 / 8.0);
   sFilterParam.sFltCoeff.f32B2 = FRAC32(-0.011085434420561 / 8.0);
   sFilterParam.sFltCoeff.f32B3 = FRAC32(0.0 / 8.0);
   sFilterParam.sFltCoeff.f32B4 = FRAC32(0.005542717210281 / 8.0);
   sFilterParam.sFltCoeff.f32A1 = FRAC32(-1.171272075750262 / -8.0);
   sFilterParam.sFltCoeff.f32A2 = FRAC32(2.122554479822350 / -8.0);
   sFilterParam.sFltCoeff.f32A3 = FRAC32(-1.047780658093187 / -8.0);
   sFilterParam.sFltCoeff.f32A4 = FRAC32(0.800802646665706 / -8.0);

   GDFLIB_FilterIIR4Init_F16(&sFilterParam);

   f16InX = FRAC16(0.1);
}
```

```
/* periodically called function */
void Isr(void)
{
     f16Result = GDFLIB_FilterIIR4_F16(f16InX, &sFilterParam);
}
```

2.6 GDFLIB_FilterMA

The GDFLIB_FilterMA function calculates a recursive form of a moving average filter. For a proper use, it is recommended that the algorithm is initialized by the GDFLIB_FilterMAInit function, before using the GDFLIB_FilterMA function.

The filter calculation consists of the following equations:

$$acc(k) = acc(k-1) + x(k)$$

Figure 45.

$$y(k) = \frac{acc(k)}{n_p}$$

Figure 46.

$$acc(k) \leftarrow acc(k) - y(k)$$

Figure 47.

where:

- x(k) is the actual value of the input signal
- acc(k) is the internal filter accumulator
- y(k) is the actual filter output
- n_p is the number of points in the filter window

The size of the filter window (number of filtered points) must be defined before calling this function, and must be equal to or greater than 1.

The function returns the filtered value of the input at step k, and stores the difference between the filter accumulator and the output at step k into the filter accumulator.

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 parameters use the accumulator types.

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

Table 12. Function versions

Function name	Input type	Parameters	Result type	Description
GDFLIB_FilterMAInit_F16	frac16_t	GDFLIB_FILTER_MA_T_A32*	void	Input argument is a 16-bit fractional value that represents the initial value of the filter at the current step. The input is within the range <-1 ; 1). The parameters' structure is pointed to by a pointer.

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

Table 13. Function versions

Function name	Input type		Result type	Description
	Value	Parameter		
GDFLIB_FilterMA_F16	frac16_t	GDFLIB_FILTER_MA_T_A32 *	frac16_t	Input argument is a 16-bit fractional value of the input signal to be filtered within the range <-1 ; 1). The parameters' structure is pointed to by a pointer. The function returns a 16-bit fractional value within the range <-1 ; 1).

2.6.2 GDFLIB_FILTER_MA_T_A32

Variable name	Input type	Description
a32Acc	acc32_t	Filter accumulator. The parameter is a 32-bit accumulator type within the range <-65536.0 ; 65536.0). Controlled by the algorithm.
u16Sh	uint16_t	Number of samples for averaging filtered points (size of the window) defined as a number of shifts: $n_p = 2^{u16Sh}$ $u16Sh = \log_2 n_p$ The parameter is a 16-bit unsigned integer type within the range <0 ; 15>. Set by the user.

2.6.3 Declaration

The available GDFLIB_FilterMAInit functions have the following declarations:

```
void GDFLIB_FilterMAInit_F16(frac16_t f16InitVal, GDFLIB_FILTER_MA_T_A32 *psParam)
```

The available GDFLIB_FilterMA functions have the following declarations:

```
frac16_t GDFLIB_FilterMA_F16(frac16_t f16InX, GDFLIB_FILTER_MA_T_A32 *psParam)
```

2.6.4 Function use

The use of GDFLIB_FilterMAInit and GDFLIB_FilterMA functions is shown in the following examples:

Fixed-point version:

```
#include "gdflib.h"

static frac16_t f16Result;
static frac16_t f16InitVal, f16InX;
static GDFLIB_FILTER_MA_T_A32 sFilterParam;

void Isr(void);

void main(void)
{
    f16InitVal = FRAC16(0.0);                /* f16InitVal = 0.0 */
```

```
    /* Filter window = 2 ^ 2 = 4 points */
    sFilterParam.u16Sh = 2;

    GDFLIB_FilterMAInit_F16(f16InitVal, &sFilterParam);

    f16InX =  FRAC16(0.8);
}

/* periodically called function */
void Isr(void)
{
    f16Result = GDFLIB_FilterMA_F16(f16InX, &sFilterParam);
}
```

Appendix A
Library types

A.1 bool_t

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

```
typedef unsigned short bool_t;
```

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

Table 14. Data storage

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

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

A.2 uint8_t

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

```
typedef unsigned char uint8_t;
```

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

Table 15. Data storage

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

Table continues on the next page...

Table 15. Data storage (continued)

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

A.3 uint16_t

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

```
typedef unsigned short uint16_t;
```

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

Table 16. Data storage

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

A.4 uint32_t

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

```
typedef unsigned long uint32_t;
```

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

Table 17. Data storage

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

A.5 int8_t

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

```
typedef char int8_t;
```

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

Table 18. Data storage

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

Table continues on the next page...

Table 18. Data storage (continued)

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

A.6 int16_t

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

```
typedef short int16_t;
```

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

Table 19. Data storage

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

A.7 int32_t

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

```
typedef long int32_t;
```

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

Table 20. Data storage

Table continues on the next page...

Table 20. Data storage (continued)

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

A.8 frac8_t

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

```
typedef char frac8_t;
```

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

Table 21. Data storage

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

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

A.9 frac16_t

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

```
typedef short frac16_t;
```

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

Table 22. Data storage

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

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

A.10 frac32_t

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

```
typedef long frac32_t;
```

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

Table 23. Data storage

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

Table continues on the next page...

Table 23. Data storage (continued)

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

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

A.11 acc16_t

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

```
typedef short acc16_t;
```

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

Table 24. Data storage

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

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

A.12 acc32_t

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

```
typedef long acc32_t;
```

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

Table 25. Data storage

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

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

A.13 FALSE

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

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

```
#include "mlib.h"

static bool_t bVal;

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

A.14 TRUE

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

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

```
#include "mlib.h"

static bool_t bVal;

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

A.15 FRAC8

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

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

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

```
#include "mlib.h"

static frac8_t f8Val;

void main(void)
{
  f8Val = FRAC8(0.187);               /* f8Val = 0.187 */
}
```

A.16 FRAC16

The FRAC16 macro serves to convert a real number to the frac16_t type. Its definition is as follows:

```
#define FRAC16(x) ((frac16_t)((x) < 0.999969482421875 ? ((x) >= -1 ? (x)*0x8000 : 0x8000) : 0x7FFF))
```

The input is multiplied by 32768 (=2^{15}). The output is limited to the range <0x8000 ; 0x7FFF>, which corresponds to <-1.0 ; 1.0-2^{-15}>.

```
#include "mlib.h"

static frac16_t f16Val;

void main(void)
{
  f16Val = FRAC16(0.736);               /* f16Val = 0.736 */
}
```

A.17 FRAC32

The FRAC32 macro serves to convert a real number to the frac32_t type. Its definition is as follows:

```
#define FRAC32(x) ((frac32_t)((x) < 1 ? ((x) >= -1 ? (x)*0x80000000 : 0x80000000) : 0x7FFFFFFF))
```

The input is multiplied by 2147483648 (=2^{31}). The output is limited to the range <0x80000000 ; 0x7FFFFFFF>, which corresponds to <-1.0 ; 1.0-2^{-31}>.

```
#include "mlib.h"

static frac32_t f32Val;

void main(void)
{
  f32Val = FRAC32(-0.1735667);                /* f32Val = -0.1735667 */
}
```

A.18 ACC16

The ACC16 macro serves to convert a real number to the acc16_t type. Its definition is as follows:

```
#define ACC16(x) ((acc16_t)((x) < 255.9921875 ? ((x) >= -256 ? (x)*0x80 : 0x8000) : 0x7FFF))
```

The input is multiplied by 128 (=2^7). The output is limited to the range <0x8000 ; 0x7FFF> that corresponds to <-256.0 ; 255.9921875>.

```
#include "mlib.h"

static acc16_t a16Val;

void main(void)
{
  a16Val = ACC16(19.45627);               /* a16Val = 19.45627 */
}
```

A.19 ACC32

The ACC32 macro serves to convert a real number to the acc32_t type. Its definition is as follows:

```
#define ACC32(x) ((acc32_t)((x) < 65535.999969482421875 ? ((x) >= -65536 ? (x)*0x8000 : 0x80000000) :
0x7FFFFFFF))
```

The input is multiplied by 32768 (=2^{15}). The output is limited to the range <0x80000000 ; 0x7FFFFFFF>, which corresponds to <-65536.0 ; 65536.0-2^{-15}>.

```
#include "mlib.h"

static acc32_t a32Val;

void main(void)
```

```
{
  a32Val = ACC32(-13.654437);                /* a32Val = -13.654437 */
}
```

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: CM33GDFLIBUG