PCLIB User's Guide

ARM® Cortex® M0+

Document Number: CM0PCLIBUG
Rev. 3, 04/2016

Review Draft

Contents

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

Chapter 1
Library

Chapter 2
Algorithms in detail

Chapter 1
Library

1.1 Introduction

1.1.1 Overview

This user's guide describes the Power Control Library (PCLIB) for the family of ARM Cortex M0+ core-based microcontrollers. This library contains optimized functions.

1.1.2 Data types

PCLIB 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

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

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

where the function is compiled from four parts:

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

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

Table 1-1. Input/output types

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

1.1.4 Supported compilers

PCLIB for the ARM Cortex M0+ core is written in C language. The library is built and tested using the following compilers:

- Kinetis Design Studio
- IAR Embedded Workbench
- Keil µVision

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

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

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

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

1.1.5 Special issues

1. The equations describing the algorithms are symbolic. If there is positive 1, the number is the closest number to 1 that the resolution of the used fractional type allows. If there are maximum or minimum values mentioned, check the range allowed by the type of the particular function version.
2. The library functions that round the result (the API contains Rnd) round to nearest (half up).

1.2 Library integration into project (Kinetis Design Studio)

This section provides a step-by-step guide on how to quickly and easily include PCLIB into an empty project or any SDK example or demo application projects using Kinetis Design Studio. This example uses the default installation path (C:\NXP\RTCESL\CM0_RTCESL_4.3_KDS). If you have a different installation path, use that path instead. If you want to use an existing SDK project (for example the hello_world project) see Memory-mapped divide and square root support . If not, continue with the next section.

1.2.1 New project (without SDK)

This example uses the NXP MKV10Z32xxx7 MCU, and the default installation path (C:\NXP\RTCESL\CM0_RTCESL_4.3_KDS) 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. Follow these steps to create a new project:

1. Launch Kinetis Design Studio.
2. Select File > New > Kinetis Design Studio Project so that the New Kinetis Design Studio Project dialog appears.
3. Type the name of the project, for example, MyProject01.
4. If you don't use the default location, untick the Use default location checkbox, and type the path where you want to create the project folder (for example, C:\KDSProjects\MyProject01) and click Next. See Figure 1-1.

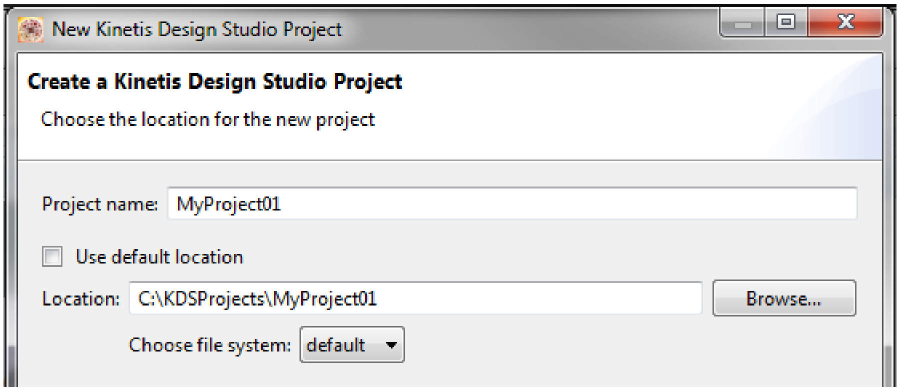

Figure 1-1. Project name and location

5. Expand the tree by clicking Processors, then Kinetis V > MKV1x > KV10Z, and click MKV10Z32xxx7. Click Finish. See Figure 1-2.

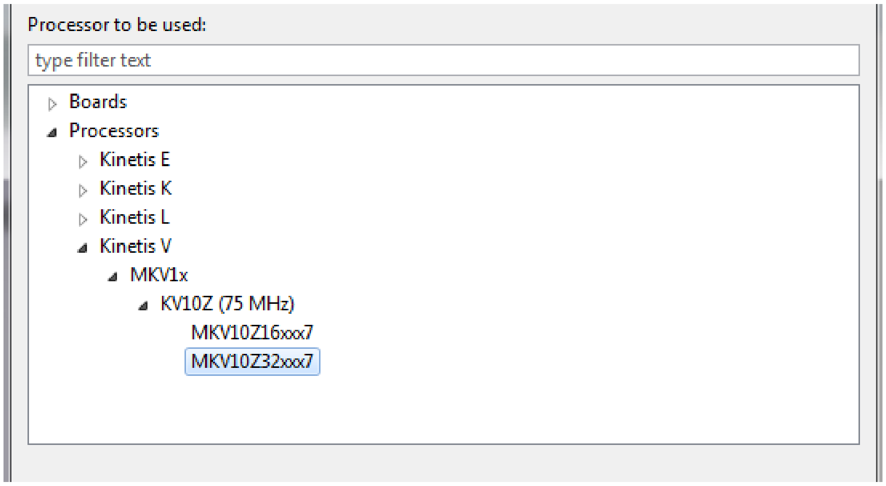

Figure 1-2. Processor selection

6. The newly-created project is now visible in the left-hand part of Kinetis Design Studio. See Figure 1-3.

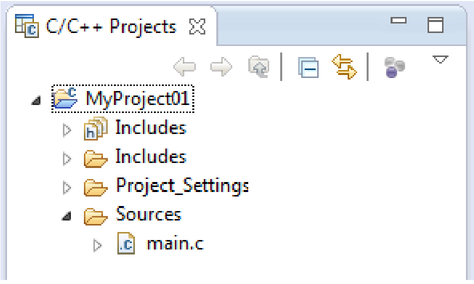

Figure 1-3. Project folder

1.2.2 Memory-mapped divide and square root support

Some Kinetis platforms contain a peripheral module dedicated for division and square root. This section shows how to turn the memory-mapped divide and square root (MMDVSQ) support on and off.

1. Right-click the MyProject01 or SDK project name node or in the left-hand part and 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 1-4.
3. In the right-hand part, under the Cross ARM C compiler node, click the Preprocessor node. See Figure 1-4.

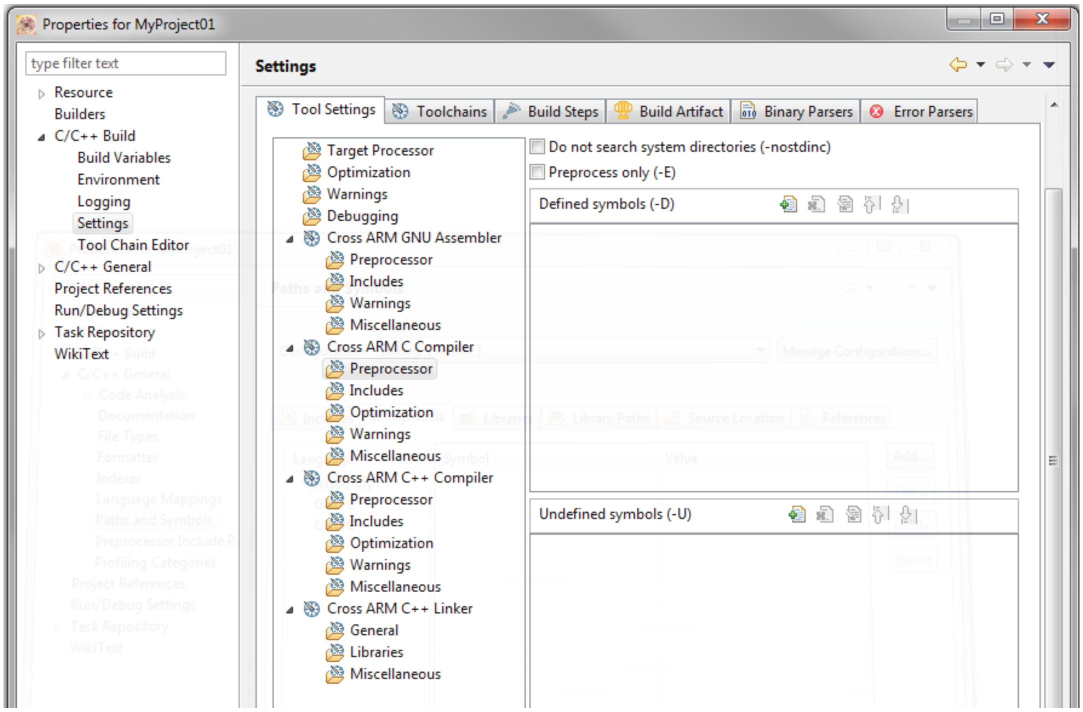

Figure 1-4. Defined symbols

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 1-5), type the following:
   - RTCESL_MMDVSQ_ON—to turn the hardware division and square root support on
   - RTCESL_MMDVSQ_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.

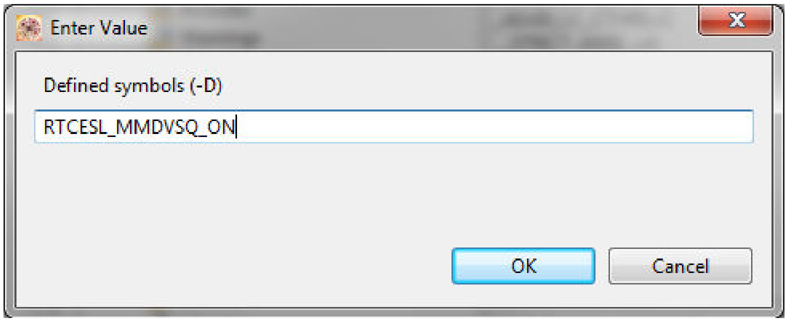

Figure 1-5. Symbol definition

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

See the device reference manual to verify whether the device contains the MMDVSQ module.

1.2.3 Library path variable

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

1. Right-click the MyProject01 or SDK project name node in the left-hand part and click Properties, or select Project > Properties from the menu. A project properties dialog appears.
2. Expand the Resource node and click Linked Resources. See Figure 1-6.

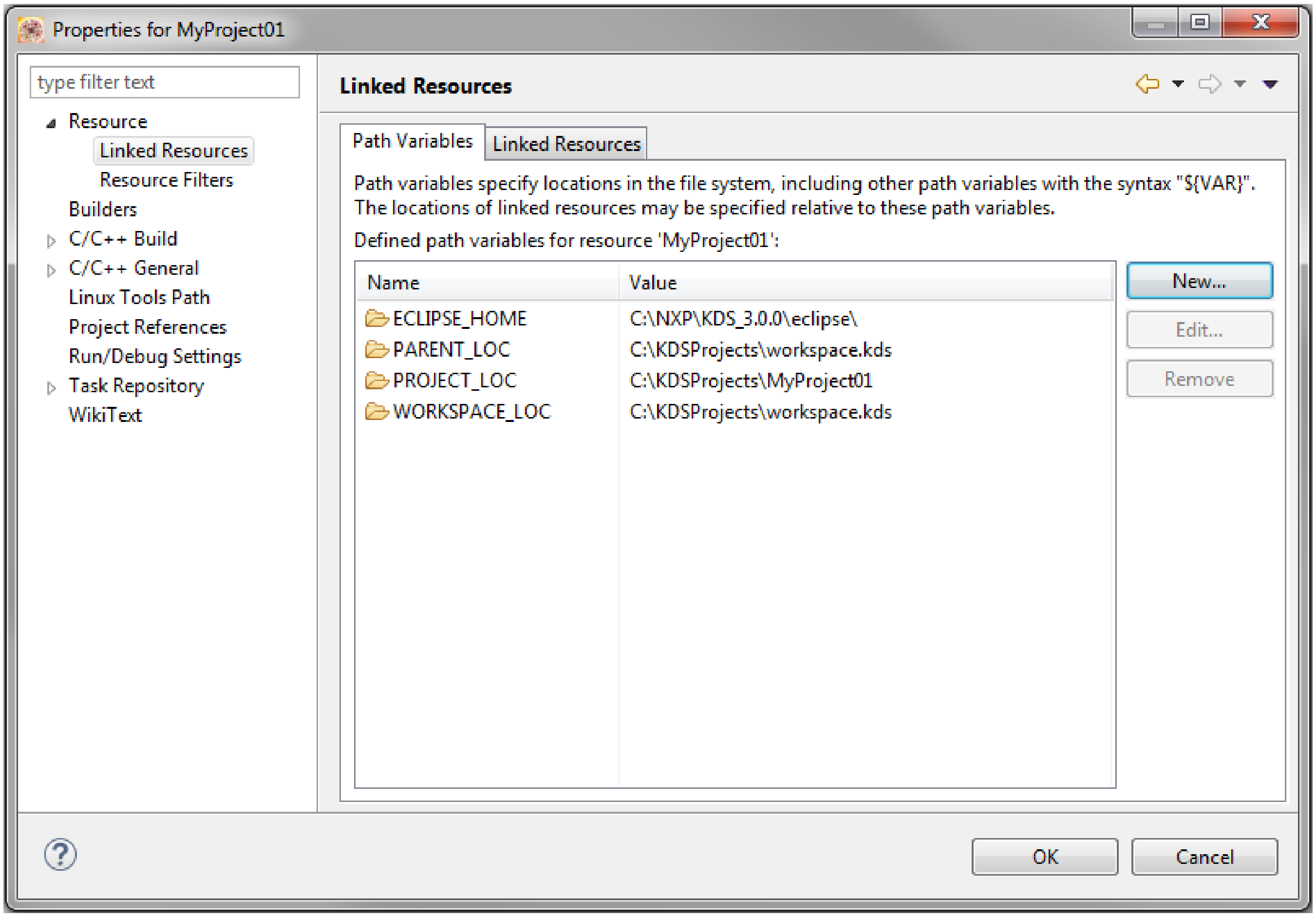

Figure 1-6. Project properties

3. Click the New… button in the right-hand side.
4. In the dialog that appears (see Figure 1-7), type this variable name into the Name box: RTCESL_LOC.
5. Select the library parent folder by clicking Folder…, or just type the following path into the Location box: C:\NXP\RTCESL\CM0_RTCESL_4.3_KDS. Click OK.

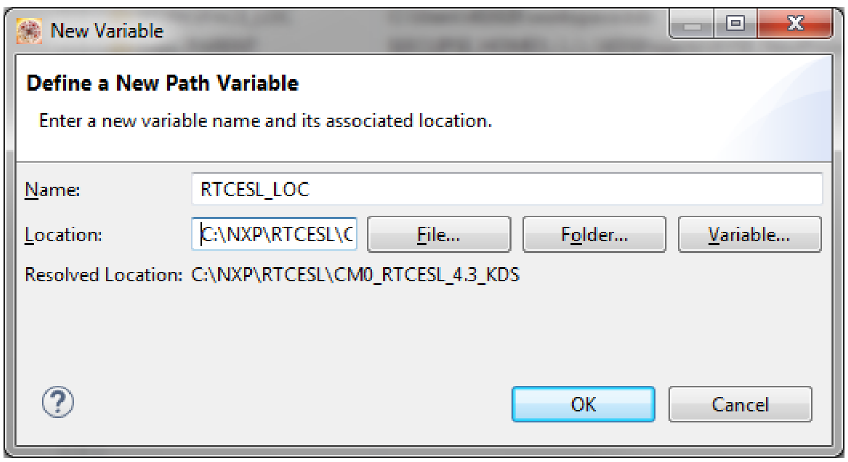

Figure 1-7. New variable

6. Create such variable for the environment. Expand the C/C++ Build node and click Environment.
7. Click the Add… button in the right-hand side.
8. In the dialog that appears (see Figure 1-8), type this variable name into the Name box: RTCESL_LOC.
9. Type the library parent folder path into the Value box: C:\NXP\RTCESL \CM0_RTCESL_4.3_KDS.
10. Tick the Add to all configurations box to use this variable in all configurations. See Figure 1-8.
11. Click OK.
12. In the previous dialog, click OK.

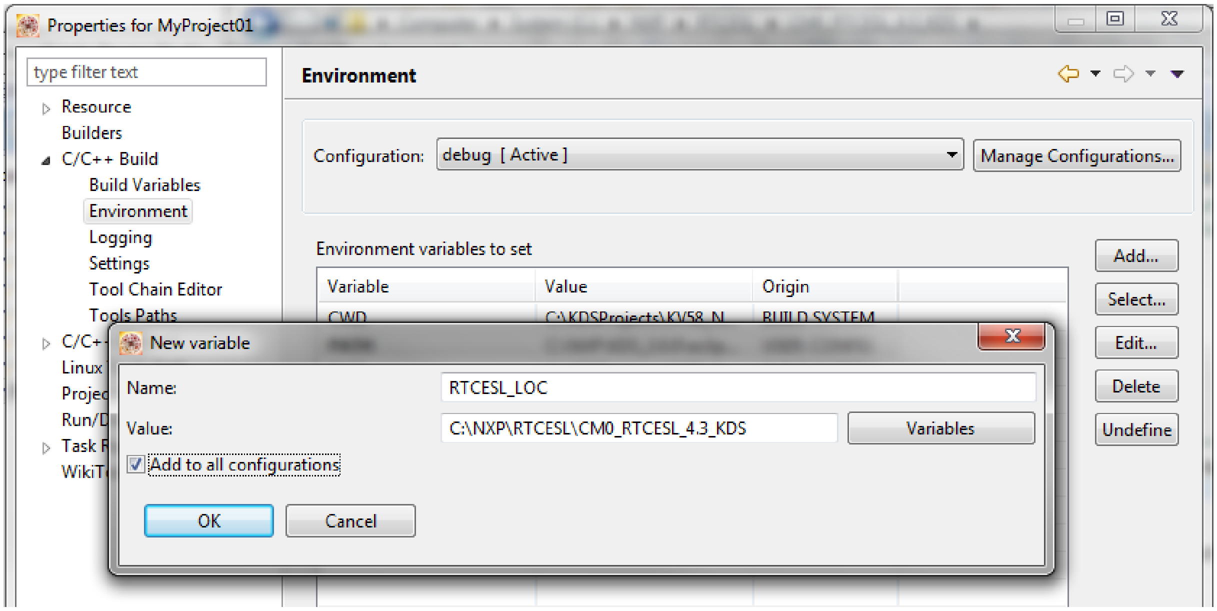

Figure 1-8. Environment variable

1.2.4 Library folder addition

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

1. Right-click the MyProject01 or SDK project name node in the left-hand part and click New > Folder, or select File > New > Folder from the menu. A dialog appears.
2. Click Advanced to show the advanced options.
3. To link the library source, select the option Link to alternate location (Linked Folder).
4. Click Variables..., select the RTCESL_LOC variable in the dialog, click OK, and/or type the variable name into the box. See Figure 1-9.
5. Click Finish, and you will see the library folder linked in the project. See Figure 1-10.

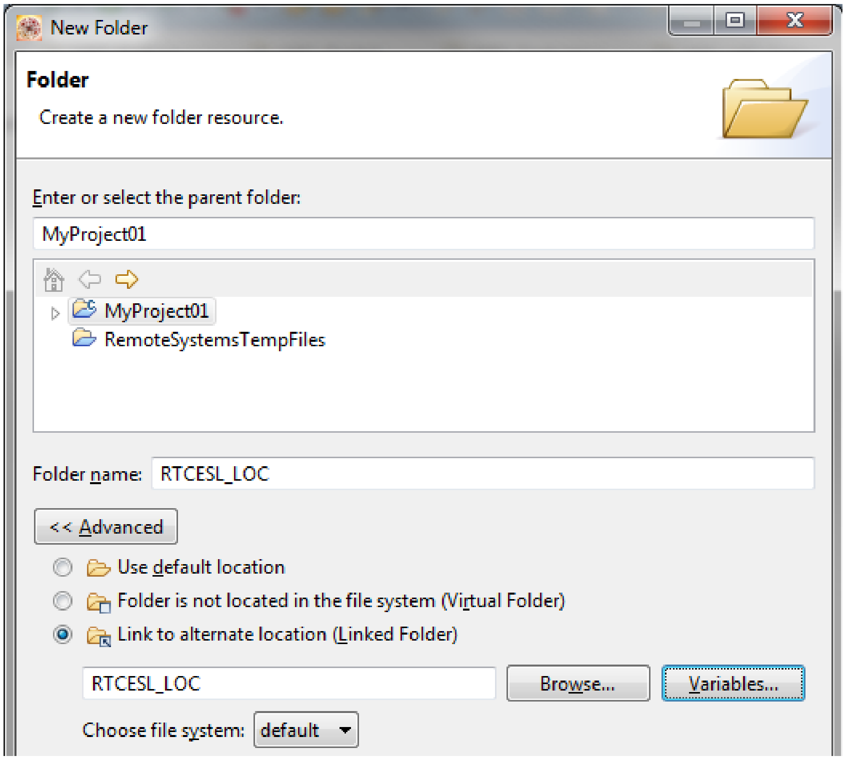

Figure 1-9. Folder link

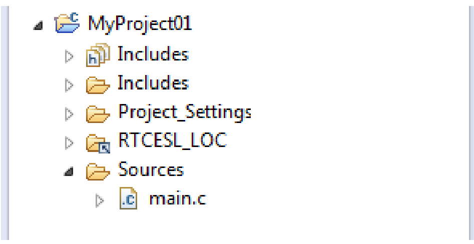

Figure 1-10. Projects libraries paths

1.2.5 Library path setup

1. Right-click the MyProject01 or SDK project name node in the left-hand part and click Properties, or select Project > Properties from the menu. A project properties dialog appears.
2. Expand the C/C++ General node, and click Paths and Symbols.
3. In the right-hand dialog, select the Library Paths tab. See Figure 1-12.
4. Click the Add… button on the right, and a dialog appears.
5. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box by adding the following (see Figure 1-11): ${RTCESL_LOC}\MLIB.
6. Click OK, and then click the Add… button.
7. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box by adding the following: ${RTCESL_LOC}\PCLIB.

8. Click OK, and the will be visible in the list. See Figure 1-12.

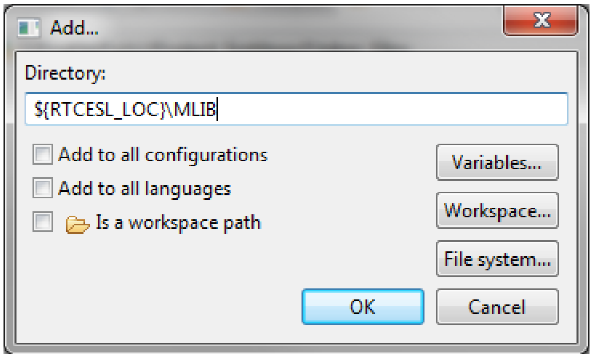

Figure 1-11. Library path inclusion

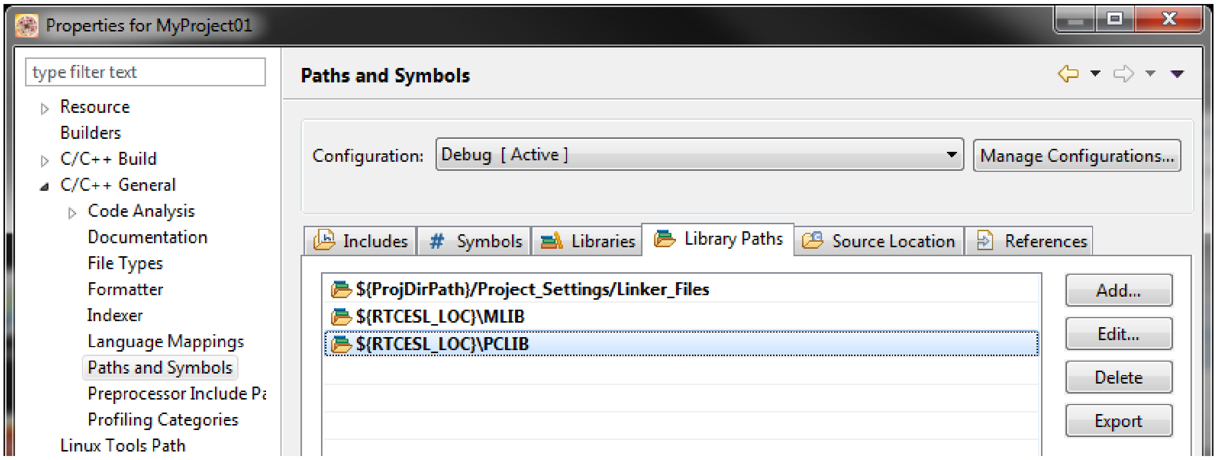

Figure 1-12. Library paths

9. After adding the library paths, add the library files. Click the Libraries tab. See Figure 1-14.
10. Click the Add… button on the right, and a dialog appears.
11. Type the following into the File text box (see Figure 1-13): :mlib.a
12. Click OK, and then click the Add… button.
13. Type the following into the File text box: :pclib.a
14. Click OK, and you will see the added in the list. See Figure 1-14.

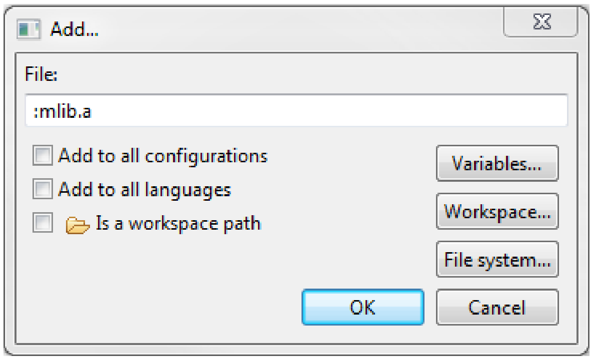

Figure 1-13. Library file inclusion

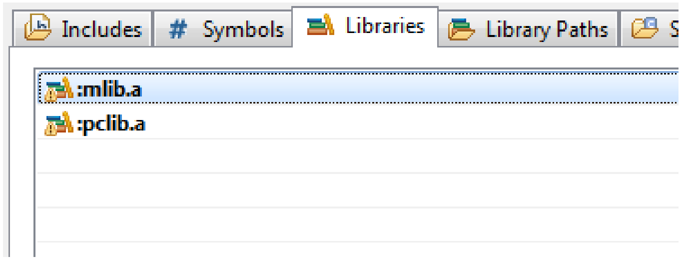

Figure 1-14. Libraries

15. In the right-hand dialog, select the Includes tab, and click GNU C in the Languages list. See Figure 1-16.
16. Click the Add… button on the right, and a dialog appears. See Figure 1-15.
17. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box to be: ${RTCESL_LOC}\MLIB\Include
18. Click OK, and then click the Add… button.
19. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box to be: ${RTCESL_LOC}\PCLIB\Include
20. Click OK, and you will see the added in the list. See Figure 1-16. Click OK.

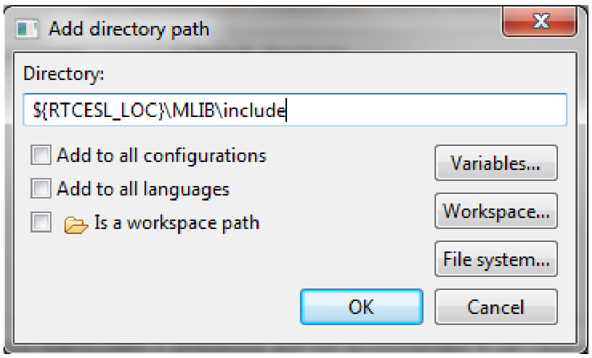

Figure 1-15. Library include path addition

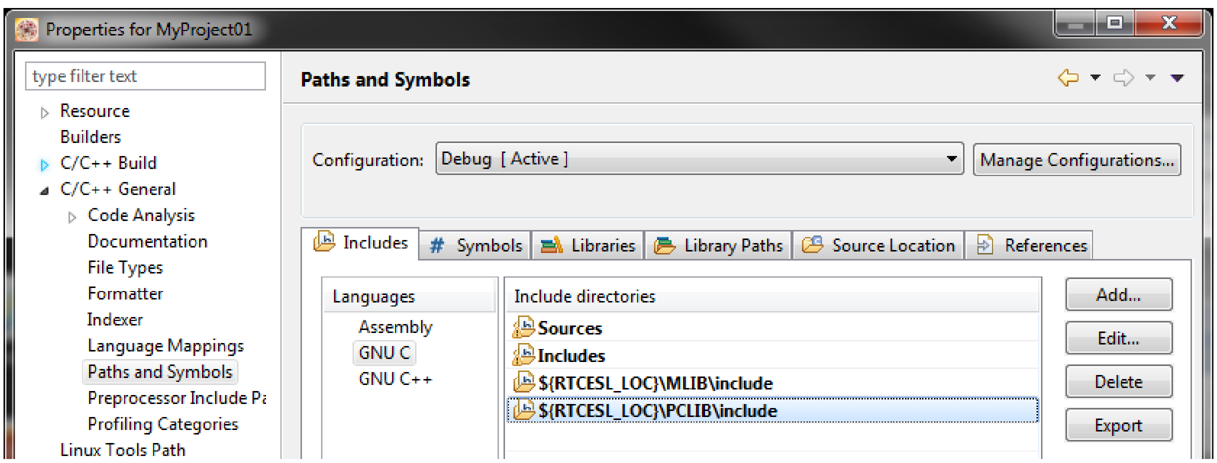

Figure 1-16. Compiler setting

Type the #include syntax into the code. Include the library into the *main.c* file. In the left-hand dialog, open the Sources folder of the project, and double-click the *main.c* file. After the *main.c* file opens up, include the following lines in the #include section:

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

When you click the Build icon (hammer), the project will be 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 PCLIB into an empty project or any SDK example or demo application projects using Keil µVision. This example uses the default installation path (C:\NXP\RTCESL\CM0_RTCESL_4.3_KEIL). If you have a different installation path, use that path instead. If any SDK project is intended to use (for example hello_world project) go to Memory-mapped divide and square root support chapter otherwise read next chapter.

1.3.1 NXP pack installation for new project (without SDK)

This example uses the NXP MKV10Z32xxx7 part, and the default installation path (C:\NXP\RTCESL\CM0_RTCESL_4.3_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 1-17.
7. When installed, the button has the "Up to date" title. Now close the Pack Installer.

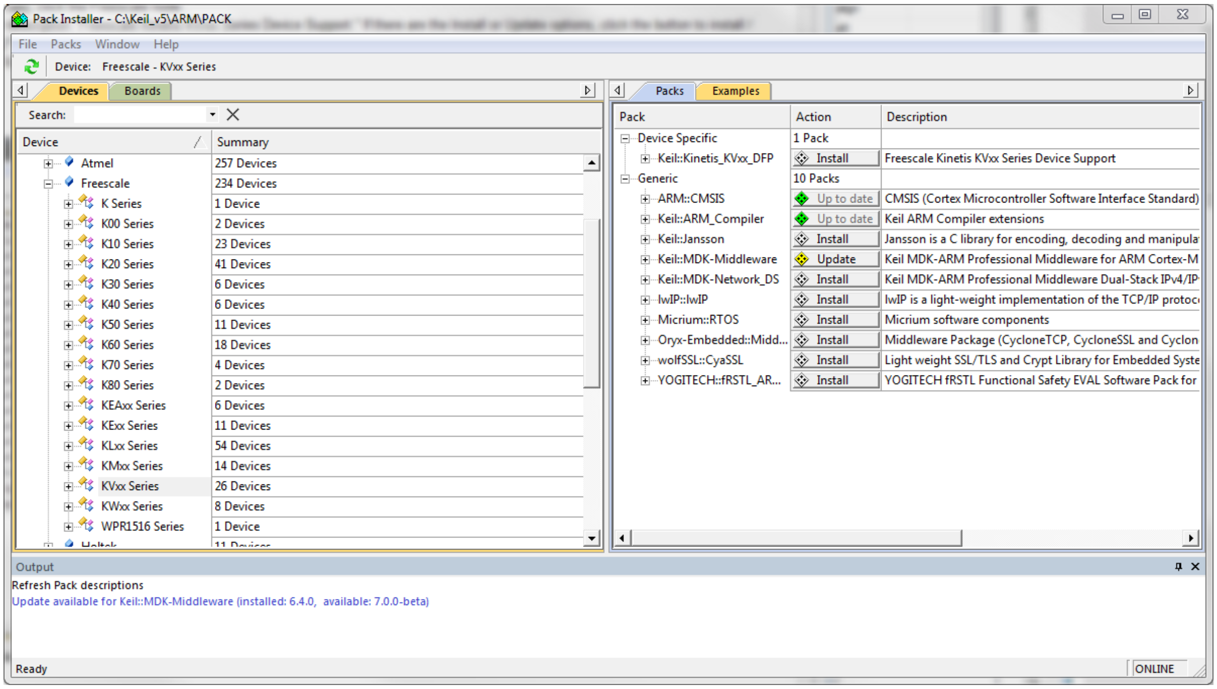

Figure 1-17. Pack Installer

1.3.2 New project (without 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 1-18.

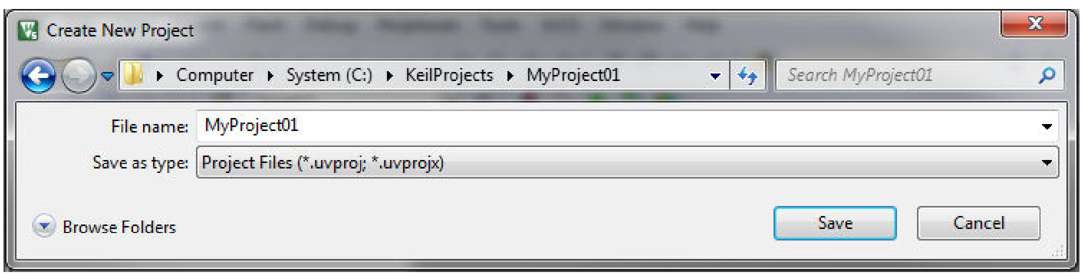

Figure 1-18. Create New Project dialog

4. In the next dialog, select the Software Packs in the very first box.
5. Type 'kv10' into the Search box, so that the device list is reduced to the KV10 devices.
6. Expand the KV10 node.

7. Click the MKV10Z32xxx7 node, and then click OK. See Figure 1-19.

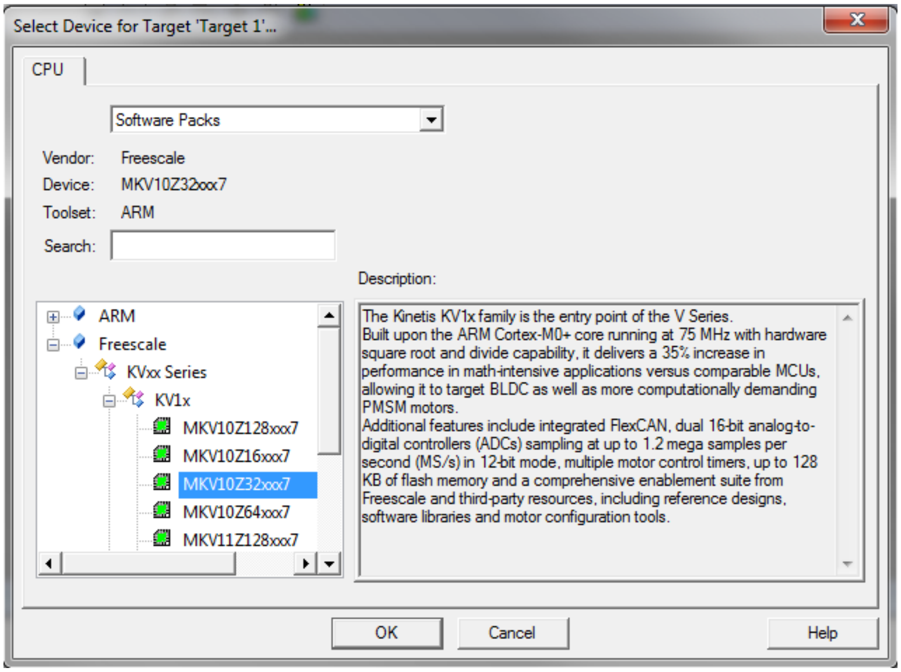

Figure 1-19. Select Device dialog

8. In the next dialog, expand the Device node, and tick the box next to the Startup node. See Figure 1-20.
9. Expand the CMSIS node, and tick the box next to the CORE node.

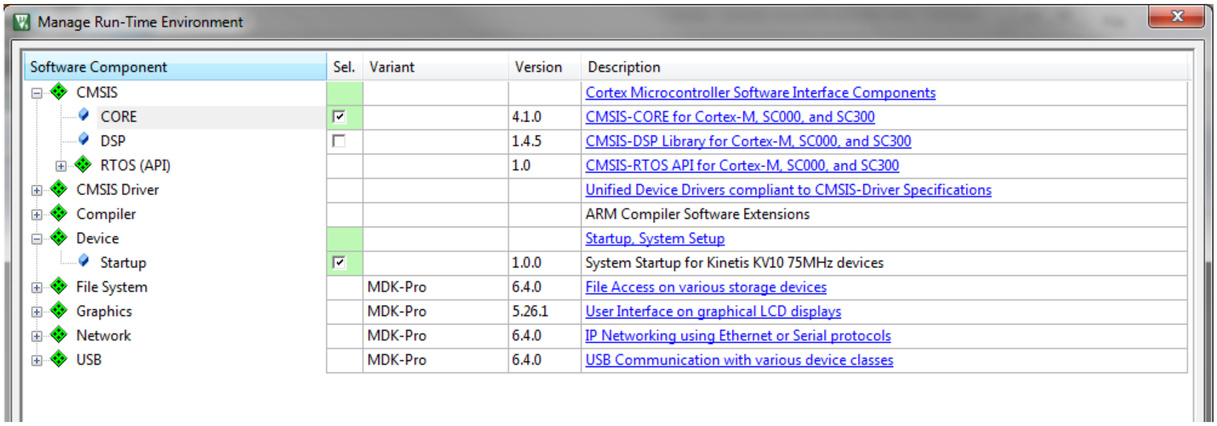

Figure 1-20. Manage Run-Time Environment dialog

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 1-21.

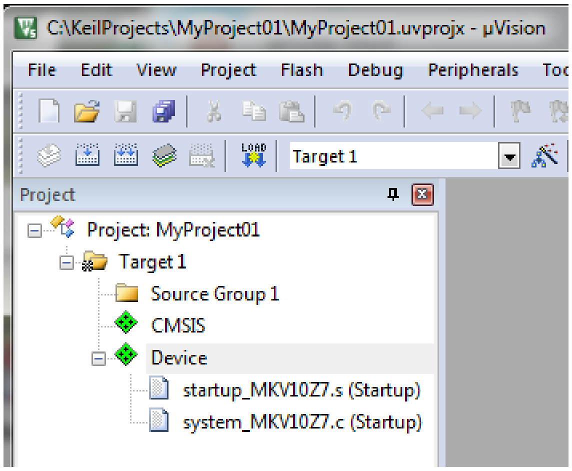

Figure 1-21. Project

1.3.3 Memory-mapped divide and square root support

Some Kinetis platforms contain a peripheral module dedicated for division and square root. This section shows how to turn the memory-mapped divide and square root (MMDVSQ) support on and off.

1. In the main menu, go to Project > Options for Target 'Target1'…, and a dialog appears.
2. Select the C/C++ tab. See Figure 1-22.
3. In the Include Preprocessor Symbols text box, type the following:
    - RTCESL_MMDVSQ_ON—to turn the hardware division and square root support on
    - RTCESL_MMDVSQ_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.

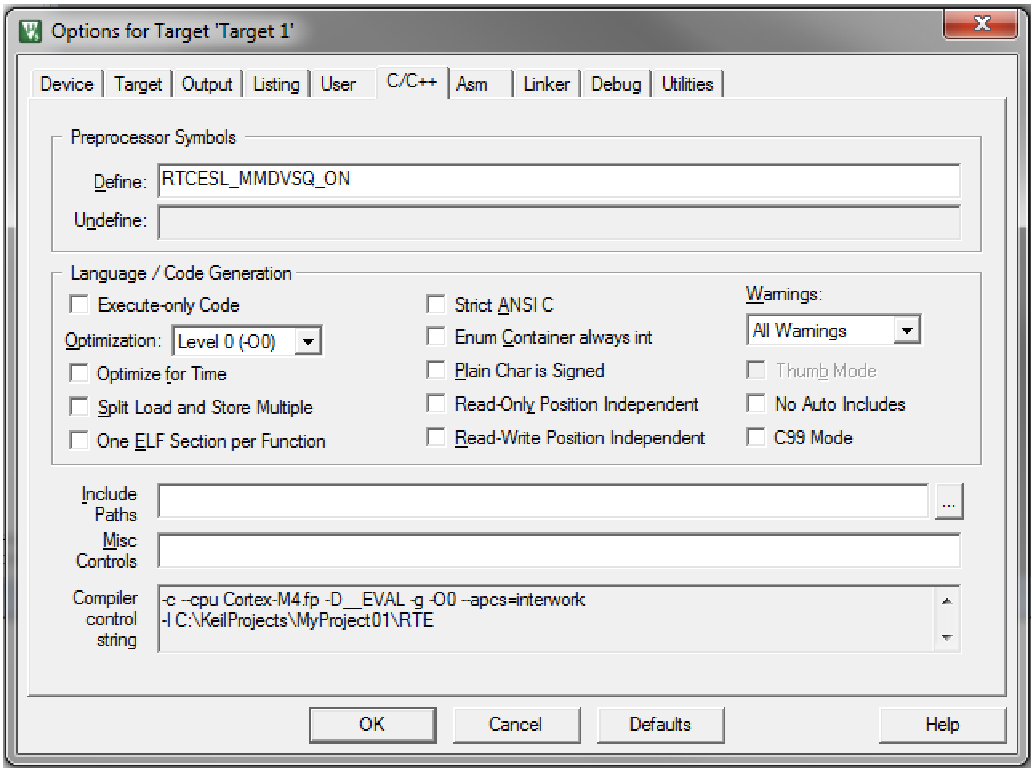

Figure 1-22. Preprocessor symbols

4. Click OK in the main dialog.

See the device reference manual to verify whether the device contains the MMDVSQ module.

1.3.4 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 \CM0_RTCESL_4.3_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 1-23.

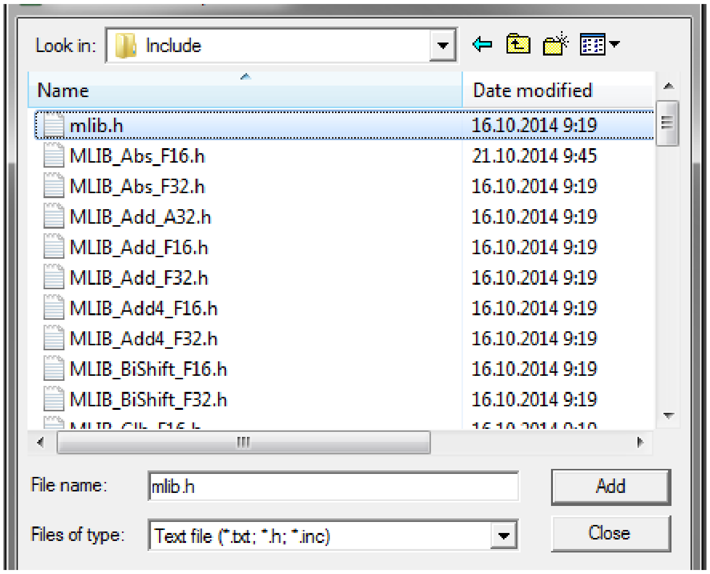

Figure 1-23. Adding .h files dialog

5. Navigate to the parent folder C:\NXP\RTCESL\CM0_RTCESL_4.3_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 1-24.

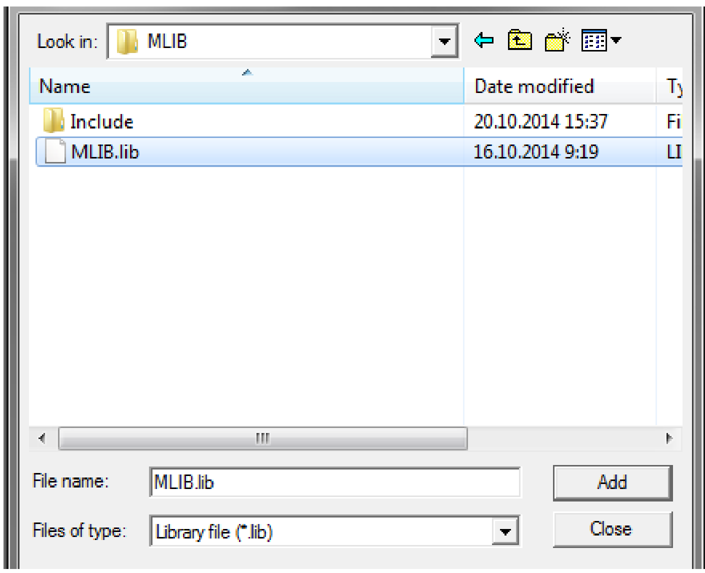

Figure 1-24. Adding .lib files dialog

6. Navigate into the library installation folder C:\NXP\RTCESL \CM0_RTCESL_4.3_KEIL\PCLIB\Include, and select the *pclib.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\CM0_RTCESL_4.3_KEIL\PCLIB, and select the *pclib.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 1-25. Click Close.

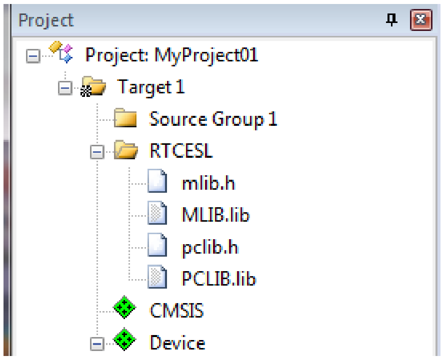

Figure 1-25. Project workspace

1.3.5 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 1-26.
3. In the Include Paths text box, type the following (if there are more paths, they must be separated by ';') or add by clicking the … button next to the text box:
   - "C:\NXP\RTCESL\CM0_RTCESL_4.3_KEIL\MLIB\Include"
   - "C:\NXP\RTCESL\CM0_RTCESL_4.3_KEIL\PCLIB\Include"
4. Click OK.
5. Click OK in the main dialog.

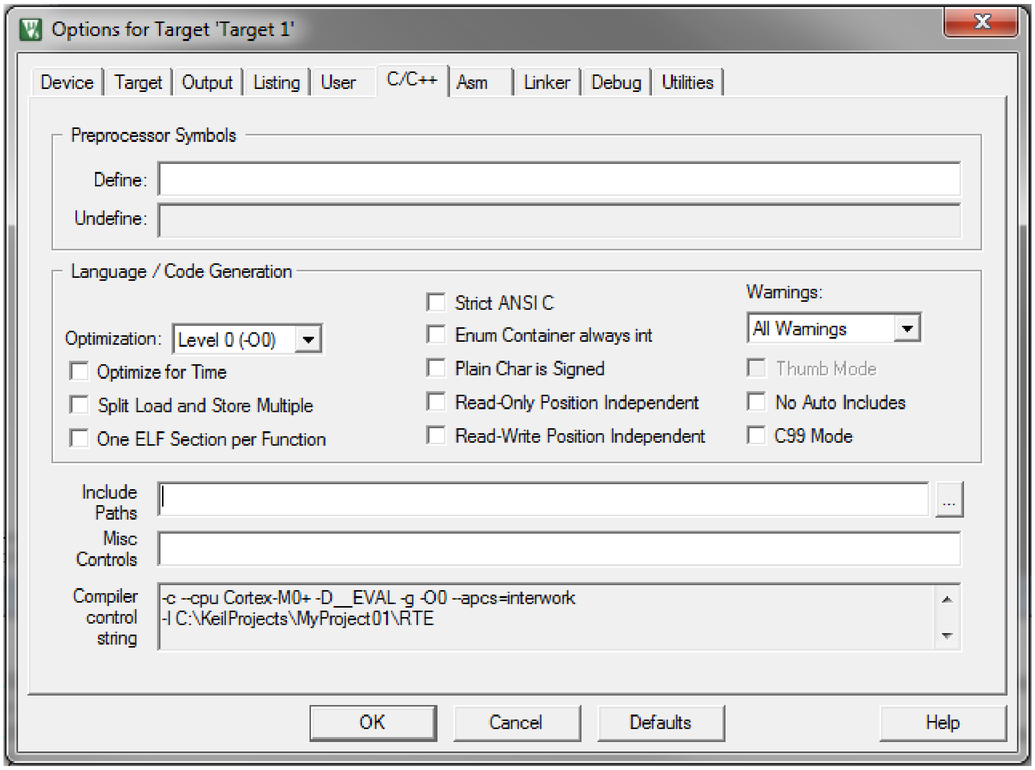

Figure 1-26. Library path addition

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 1-27.

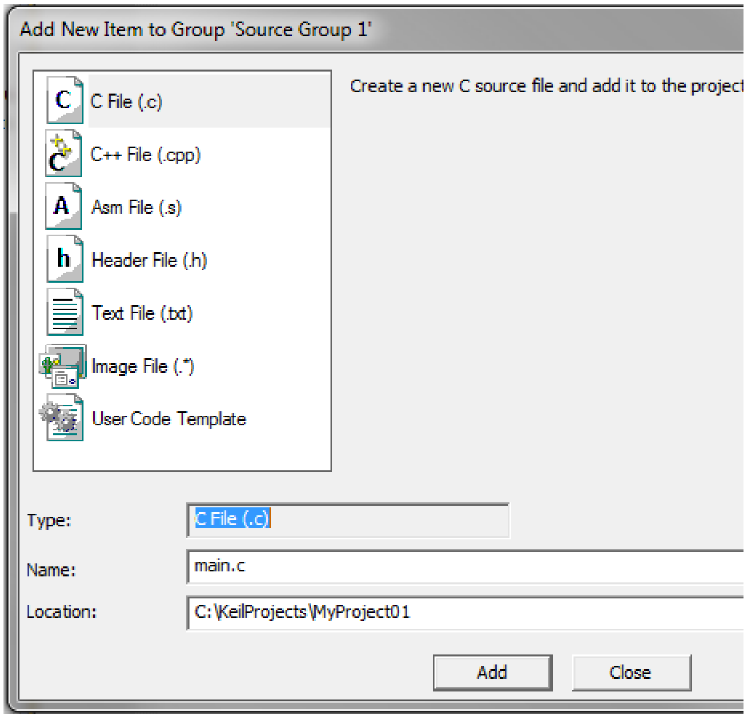

Figure 1-27. Adding new source file dialog

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 "pclib.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 PCLIB into an empty project or any SDK example or demo application projects using IAR Embedded Workbench. This example uses the default installation path (C:\NXP\RTCESL\CM0_RTCESL_4.3_IAR). If you have a different installation path, use that path instead. If any SDK project is intended to use (for example hello_world project) go to Memory-mapped divide and square root support chapter otherwise read next chapter.

1.4.1 New project (without SDK)

This example uses the NXP MKV10Z32xxx7 part, and the default installation path (C:\NXP\RTCESL\CM0_RTCESL_4.3_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 1-28.

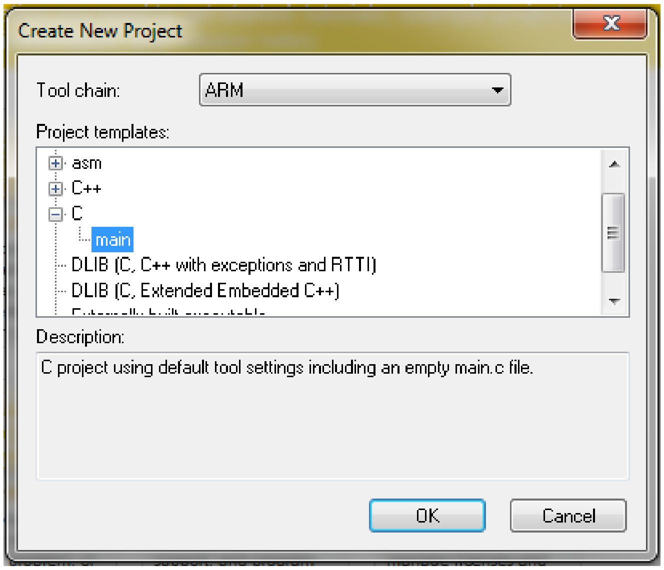

Figure 1-28. Create New Project dialog

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 1-29.

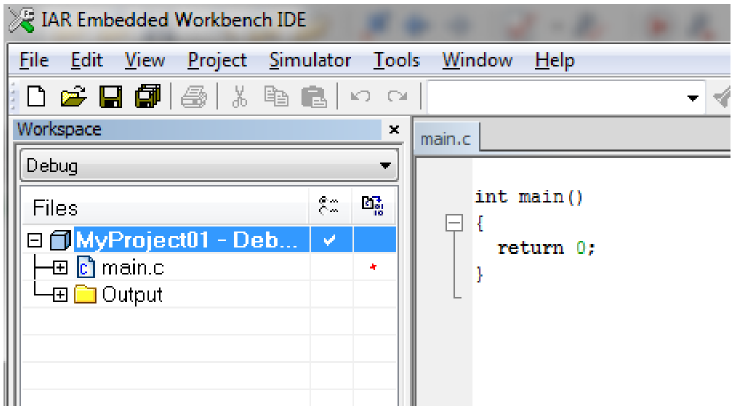

Figure 1-29. New project

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 > KV1x > NXP MKV10Z32xxx7 Click OK. See Figure 1-30.

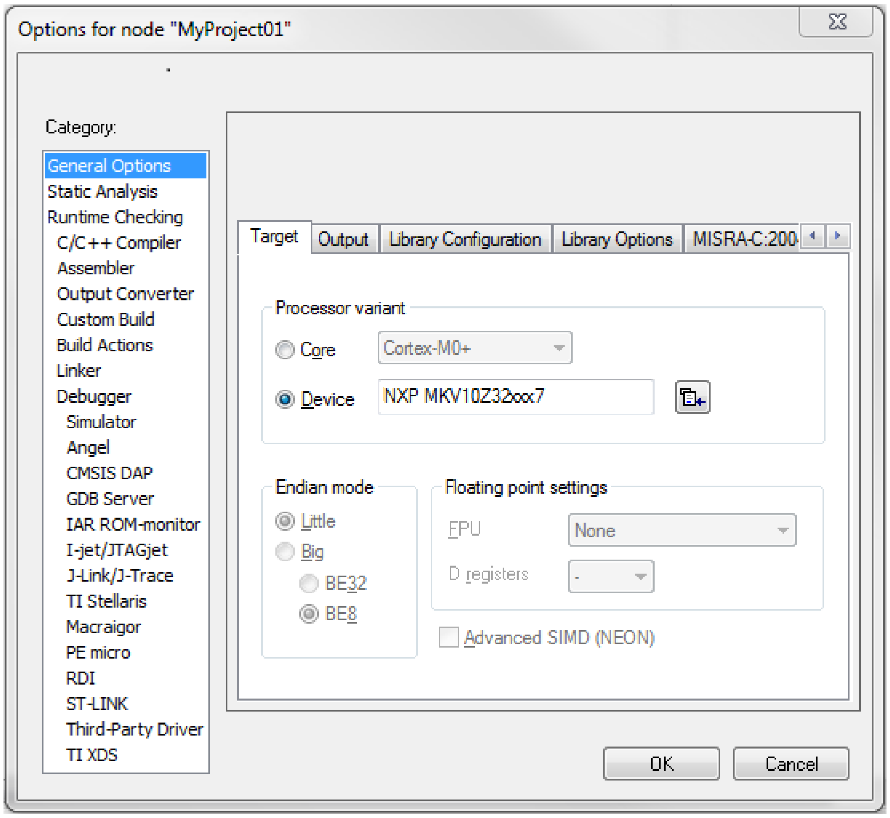

Figure 1-30. Options dialog

1.4.2 Memory-mapped divide and square root support

Some Kinetis platforms contain a peripheral module dedicated to division and square root. This section shows how to turn the memory-mapped divide and square root (MMDVSQ) support 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; use the arrow icons for navigation).
4. In the text box (at the Defined symbols: (one per line)), type the following (See Figure 1-31):
   - RTCESL_MMDVSQ_ON—to turn the hardware division and square root support on
   - RTCESL_MMDVSQ_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.

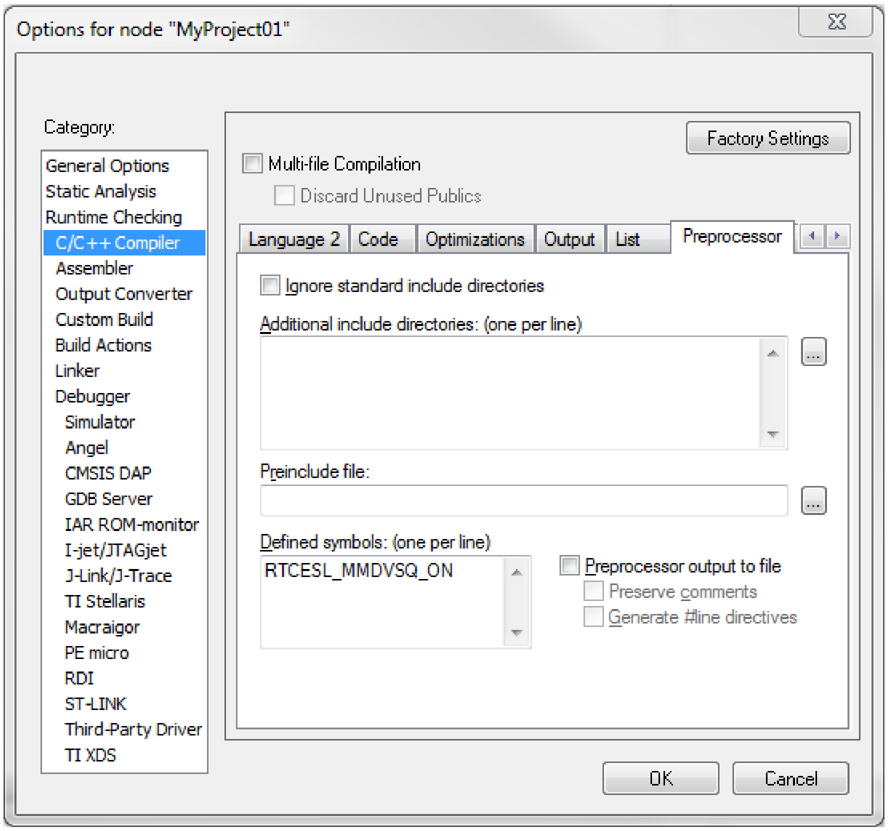

Figure 1-31. Defined symbols

5. Click OK in the main dialog.

See the device reference manual to verify whether the device contains the MMDVSQ module.

1.4.3 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 1-32.

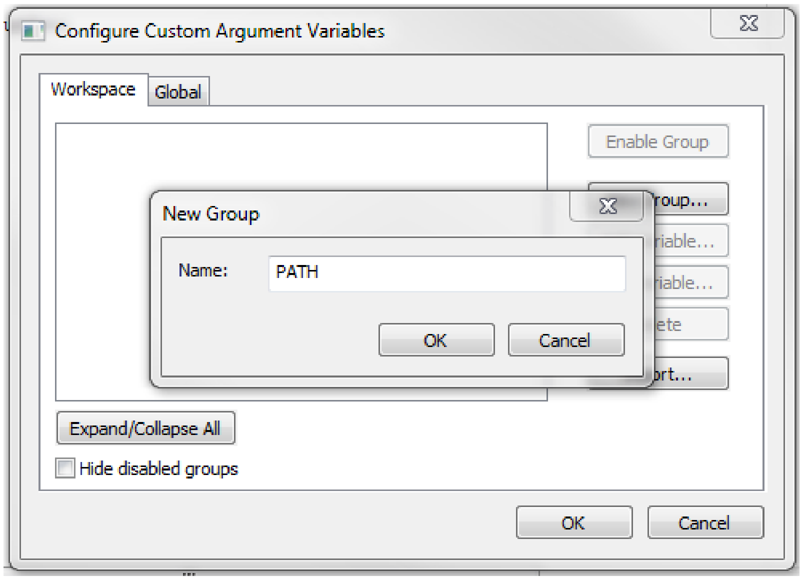

Figure 1-32. New Group

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\CM0_RTCESL_4.3_IAR. Click OK.
6. In the main dialog, click OK. See Figure 1-33.

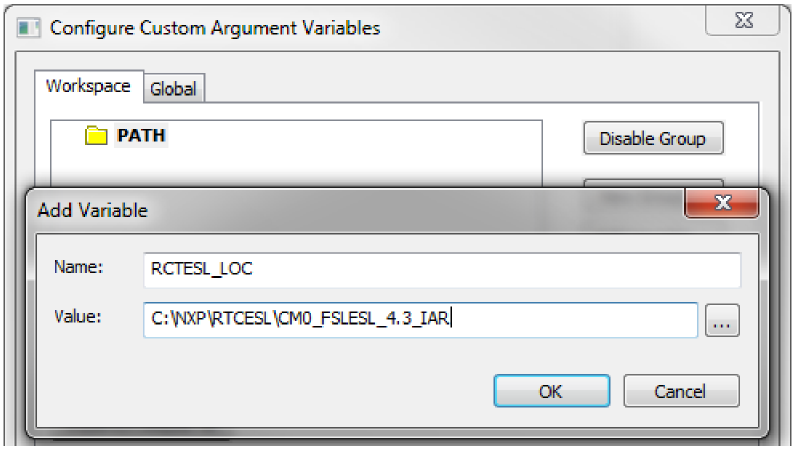

Figure 1-33. New variable

1.4.4 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 1-35.
5. Navigate into the library installation folder C:\NXP\RTCESL \CM0_RTCESL_4.3_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 1-34.
6. Navigate into the library installation folder C:\NXP\RTCESL \CM0_RTCESL_4.3_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.

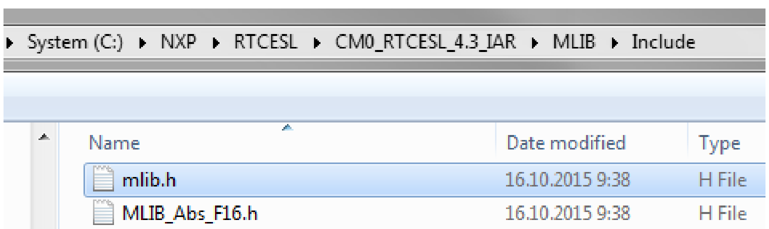

Figure 1-34. Add Files dialog

7. Click on the RTCESL node, go to Project > Add Group…, and create a PCLIB subgroup.
8. Click on the newly created node PCLIB, and go to the main menu Project > Add Files…

9. Navigate into the library installation folder C:\NXP\RTCESL \CM0_RTCESL_4.3_IAR\PCLIB\Include, and select the *pclib.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 \CM0_RTCESL_4.3_IAR\PCLIB, and select the *pclib.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 1-35.

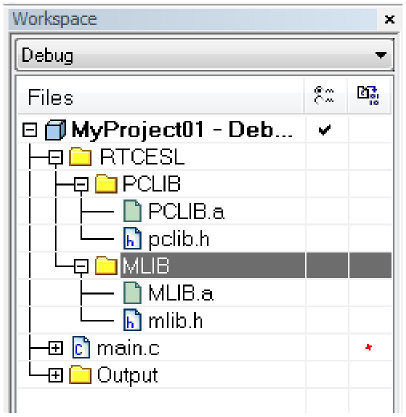

Figure 1-35. Project workspace

1.4.5 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
   - $RTCESL_LOC$\PCLIB\Include
5. Click OK in the main dialog. See Figure 1-36.

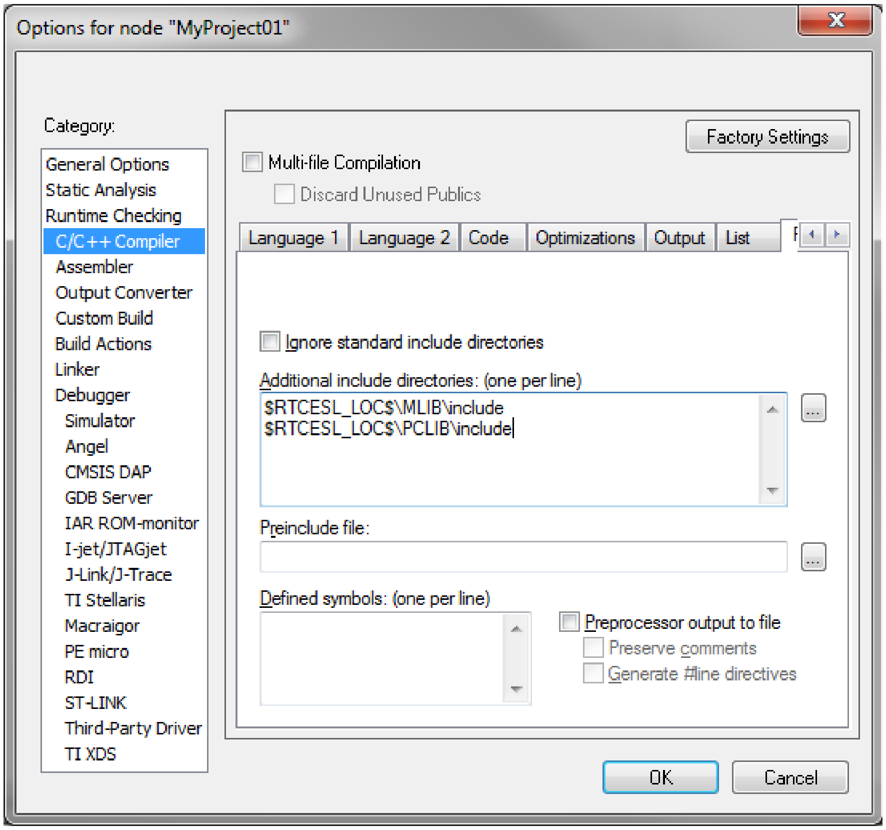

Figure 1-36. Library path adition

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 "pclib.h"
```

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

Chapter 2
Algorithms in detail

2.1 PCLIB_Ctrl2P2Z

The PCLIB_Ctrl2P2Z function calculates the compensation block for the controller, which consists of two poles and two zeroes. The s-domain transfer function equation for two-pole two-zero control law is as follows:

$$\frac{y[s]}{x[s]} = \frac{(s-Z1)(s-Z2)}{(s-P1)(s-P2)}$$

Equation 1.

where y[s] is the output, and x[s] is the input to the system. This control law has two poles (P1 and P2) and two zeroes (Z1 and Z2). The value or the placement of these poles and zeroes in the bode plot affects the stability and performance of the control loop and the system. The z-domain controller Gc(z) at sampling time Ts is expressed using the Tustin method as follows:

$$\frac{y[t]}{x[t]} = \frac{(b2 \cdot z^{-2} + b1 \cdot z^{-1} + b0)}{(1 - a2 \cdot z^{-2} - a1 \cdot z^{-1})}$$

Equation 2.

$$y[t] - a1 \cdot y[t] \cdot z^{-1} - a2 \cdot y[t] \cdot z^{-2} = b0 \cdot x[t] + b1 \cdot x[t] \cdot z^{-1} + b2 \cdot x[t] \cdot z^{-2}$$

Equation 3.

where:

- y[t] = y[n] is the present output
- $y[t] \cdot z^{-1} = y[n-1]$ is the previous output
- $y[t] \cdot z^{-2} = y[n-2]$ is the previous to previous output
- x[t] = x[n] is the present error
- $x[t] \cdot z^{-1} = x[n-1]$ is the previous error
- $x[t] \cdot z^{-2} = x[n-2]$ is the previous to previous error
- b0, b1, b2, a1, and a2 are the control coefficients and functions of Z1, Z2, P1, P2, and sampling time Ts.

$$y[n] = a2 \cdot y[n-2] + a1 \cdot y[n-1] + b2 \cdot x[n-2] + b1 \cdot x[n-1] + b0 \cdot x[n]$$

Equation 4.

For a proper use of this function, it is recommended to initialize the function's data by the PCLIB_Ctrl2P2ZInit function, before using the function. This function clears the internal buffers of the 2P2Z controller. You must call this function when you want the 2P2Z controller to be initialized. The init function must not be called together with PCLIB_Ctrl2P2Z, unless a periodic clearing of buffers is required.

2.1.1 Available versions

The available versions of the PCLIB_Ctrl2P2ZInit function are shown in the following table:

Table 2-1. Init function versions

<table>
<tr><th>Function name</th><th>Input type</th><th>Parameters</th><th>Result type</th></tr>
<tr><td rowspan="2">PCLIB_Ctrl2P2ZInit_F16</td><td>frac16_t</td><td>PCLIB_CTRL_2P2Z_T_F16 *</td><td>void</td></tr>
<tr><td colspan="3">The inputs are a 16-bit fractional initial value and a pointer to the controller's parameters structure. It clears the internal delay parameter buffers of the controller.</td></tr>
</table>

The available versions of the PCLIB_Ctrl2P2Z function are shown in the following table:

Table 2-2. Function versions

<table>
<tr><th>Function name</th><th>Input type</th><th>Parameters</th><th>Result type</th></tr>
<tr><td rowspan="2">PCLIB_Ctrl2P2Z_F16</td><td>frac16_t</td><td>PCLIB_CTRL_2P2Z_T_F16 *</td><td>frac16_t</td></tr>
<tr><td colspan="3">The error input is a 16-bit fractional value within the range <-1 ; 1). The parameters are pointed to by an input pointer. The function returns a 16-bit fractional value in the range <-1 ; 1).</td></tr>
</table>

2.1.2 PCLIB_CTRL_2P2Z_T_F16

Variable name	Type	Description
f16CoeffB0	frac16_t	Control coefficient for the present error. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16CoeffB1	frac16_t	Control coefficient for the past error. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16CoeffB2	frac16_t	Control coefficient for the past to past error. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16CoeffA1	frac16_t	Control coefficient for the past result. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.

Table continues on the next page...

Variable name	Type	Description
f16CoeffA2	frac16_t	Control coefficient for the past to past result. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16DelayX1	frac16_t	Delay parameter for the past error. Controlled by the algorithm.
f16DelayX2	frac16_t	Delay parameter for the past to past error. Controlled by the algorithm.
f16DelayY1	frac16_t	Delay parameter for the past result. Controlled by the algorithm.
f16DelayY2	frac16_t	Delay parameter for the past to past result. Controlled by the algorithm.

2.1.3 Declaration

The available PCLIB_Ctrl2P2Z functions have the following declarations:

```
void PCLIB_Ctrl2P2ZInit_F16(PCLIB_CTRL_2P2Z_T_F16 *psParam)
frac16_t PCLIB_Ctrl2P2Z_F16(frac16_t f16InErr, PCLIB_CTRL_2P2Z_T_F16 *psParam)
```

2.1.4 Function use

The use of the PCLIB_Ctrl2P2ZInit_F16 and PCLIB_Ctrl2P2Z functions is shown in the following example:

```
#include "pclib.h"

static frac16_t f16Result, f16InErr;
static PCLIB_CTRL_2P2Z_T_F16 sParam;

void Isr(void);

void main(void)
{
    f16InErr = FRAC16(-0.4);
    sParam.f16CoeffB0 = FRAC16(0.1);
    sParam.f16CoeffB1 = FRAC16(0.2);
    sParam.f16CoeffB2 = FRAC16(0.15);
    sParam.f16CoeffA1 = FRAC16(0.1);
    sParam.f16CoeffA2 = FRAC16(0.25);

    PCLIB_Ctrl2P2ZInit_F16(&sParam);
}

/* Periodical function or interrupt */
void Isr()
{
   f16Result = PCLIB_Ctrl2P2Z_F16(f16InErr, &sParam);
}
```

2.2 PCLIB_Ctrl3P3Z

The PCLIB_Ctrl3P3Z function calculates the compensation block for the controller, which consists of three poles and three zeroes. The s-domain transfer function equation for the three-pole three-zero control law is as follows:

$$\frac{y[s]}{x[s]} = \frac{(s-Z1)(s-Z2)(s-Z3)}{(s-P1)(s-P2)(s-P3)}$$

Equation 5.

where y[s] is the output and x[s] is the input to the system. This control law has three poles (P1, P2, and P3) and three zeroes (Z1, Z2, and Z3). The value or the placement of these poles and zeroes in the bode plot affects the stability and performance of the control loop and the system. The z-domain controller Gc(z) at sampling time Ts is expressed using the Tustin method as follows:

$$\frac{y[t]}{x[t]} = \frac{(b3 \cdot z^{-3} + b2 \cdot z^{-2} + b1 \cdot z^{-1} + b0)}{(1 - a3 \cdot z^{-3} - a2 \cdot z^{-2} - a1 \cdot z^{-1})}$$

Equation 6.

$$y[t] - a1 \cdot y[t] \cdot z^{-1} - a2 \cdot y[t] \cdot z^{-2} - a3 \cdot y[t] \cdot z^{-3} = b0 \cdot x[t] + b1 \cdot x[t] \cdot z^{-1} + b2 \cdot x[t] \cdot z^{-2} + b3 \cdot x[t] \cdot z^{-3}$$

Equation 7.

where:

- y[t] = y[n] is the present output
- $y[t] \cdot z^{-1}$ = y[n-1] is the previous output
- $y[t] \cdot z^{-2}$ = y[n-2] is the previous to previous output
- $y[t] \cdot z^{-3}$ = y[n-3] is the previous to previous to previous output
- x[t] = x[n] is the present error
- $x[t] \cdot z^{-1}$ = x[n-1] is the previous error
- $x[t] \cdot z^{-2}$ = x[n-2] is the previous to previous error
- $x[t] \cdot z^{-3}$ = x[n-3] is the previous to previous to previous error
- b0, b1, b2, b3 a1, a2, and a3 are the control coefficients and functions of Z1, Z2, Z3, P1, P2, P3, and sampling time Ts.

$$y[n] = a3 \cdot y[n-3] + a2 \cdot y[n-2] + a1 \cdot y[n-1] + b3 \cdot x[n-3] + b2 \cdot x[n-2] + b1 \cdot x[n-1] + b0 \cdot x[n]$$

Equation 8.

For a proper use of this function, it is recommended to initialize the function's data by the PCLIB_Ctrl3P3ZInit function, before using the function. This function clears the internal buffers of the 3P3Z controller. You must call this function when you want the 3P3Z controller to be initialized. The init function must not be called together with PCLIB_Ctrl3P3Z, unless a periodic clearing of buffers is required.

2.2.1 Available versions

The available versions of the PCLIB_Ctrl3P3ZInit function are shown in the following table:

Table 2-3. Init function versions

Function name	Input type	Parameters	Result type
PCLIB_Ctrl3P3ZInit_F16	frac16_t	PCLIB_CTRL_3P3Z_T_F16 *	void
	The inputs are a 16-bit fractional initial value and a pointer to the controller's parameters structure. It clears the internal delay parameter buffers of the controller.		

The available versions of the PCLIB_Ctrl3P3Z function are shown in the following table:

Table 2-4. Function versions

Function name	Input type	Parameters	Result type
PCLIB_Ctrl3P3Z_F16	frac16_t	PCLIB_CTRL_3P3Z_T_F16 *	frac16_t
	The error input is a 16-bit fractional value within the range <-1 ; 1). The parameters are pointed to by an input pointer. The function returns a 16-bit fractional value in the range <-1 ; 1).		

2.2.2 PCLIB_CTRL_3P3Z_T_F16

Variable name	Input type	Description
f16CoeffB0	frac16_t	Control coefficient for the present error. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16CoeffB1	frac16_t	Control coefficient for the past error. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16CoeffB2	frac16_t	Control coefficient for the past to past error. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16CoeffB3	frac16_t	Control coefficient for the past to past to past error. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16CoeffA1	frac16_t	Control coefficient for the past result. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16CoeffA2	frac16_t	Control coefficient for the past to past result. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16CoeffA3	frac16_t	Control coefficient for the past to past to past result. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16DelayX1	frac16_t	Delay parameter for the past error. Controlled by the algorithm.
f16DelayX2	frac16_t	Delay parameter for the past to past error. Controlled by the algorithm.
f16DelayX3	frac16_t	Delay parameter for the past to past to past error. Controlled by the algorithm.
f16DelayY1	frac16_t	Delay parameter for the past result. Controlled by the algorithm.
f16DelayY2	frac16_t	Delay parameter for the past to past result. Controlled by the algorithm.

Table continues on the next page...

Variable name	Input type	Description
f16DelayY3	frac16_t	Delay parameter for the past to past to past result. Controlled by the algorithm.

2.2.3 Declaration

The available PCLIB_Ctrl3P3Z functions have the following declarations:

```
void PCLIB_Ctrl3P3ZInit_F16(PCLIB_CTRL_3P3Z_T_F16 *psParam)
frac16_t PCLIB_Ctrl3P3Z_F16(frac16_t f16InErr, PCLIB_CTRL_3P3Z_T_F16 *psParam)
```

2.2.4 Function use

The use of the PCLIB_Ctrl3P3ZInit_F16 and PCLIB_Ctrl3P3Z functions is shown in the following example:

```
#include "pclib.h"

static frac16_t f16Result, f16InErr;
static PCLIB_CTRL_3P3Z_T_F16 sParam;

void Isr(void);

void main(void)
{
    f16InErr = FRAC16(-0.4);
    sParam.f16CoeffB0 = FRAC16(0.1);
    sParam.f16CoeffB1 = FRAC16(0.2);
    sParam.f16CoeffB2 = FRAC16(0.15);
    sParam.f16CoeffB3 = FRAC16(0.12);
    sParam.f16CoeffA1 = FRAC16(0.1);
    sParam.f16CoeffA2 = FRAC16(0.25);
    sParam.f16CoeffA3 = FRAC16(0.35);

    PCLIB_Ctrl3P3ZInit_F16(&sParam);
}

/* Periodical function or interrupt */
void Isr()
{
   f16Result = PCLIB_Ctrl3P3Z_F16(f16InErr, &sParam);
}
```

2.3 PCLIB_CtrlPI

The PCLIB_CtrlPI function calculates the Proportional-Integral (PI) compensation block for any given control system in power-control and motor-control applications. The integral output of the controller is also limited, and the limit values (IntegralUpperLimit and IntegralLowerLimit) are defined by the user. The controller output is also limited, and the limit values (UpperLimit and LowerLimit) are defined by the user. The integral state is limited by the controller limits in the same way as the controller output.

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

$$y(t) = Kp \cdot e(t) + \int_0^t Ki \cdot e(t) \cdot dt$$

Equation 9.

The above equation can be rewritten into the discrete time domain by approximating the integral term. The integral term is approximated by the Backward Euler method, also known as backward rectangular or right-hand approximation, as follows:

$$y_I(n) = y_I(n-1) + Ki \cdot Ts \cdot e(n)$$

Equation 10.

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

$$y(n) = Kp \cdot e(n) + y_I(n-1) + Ki \cdot Ts \cdot e(n)$$

Equation 11.

where:

- e(n) is the input error
- y(n) is the controller output
- Kp is the proportional gain
- Ki is the integral gain
- $y_I(n-1)$ is the previous integral output
- Ts is the sampling time

Rewritten as follows:

$$y(n) = Kp \cdot e(n) + y_I(n-1) + K_I \cdot e(n)$$

Equation 12.

$$K_I = Ki \cdot Ts$$

Equation 13.

For a proper use of this function, it is recommended to initialize the function's data by the PCLIB_CtrlPIInit functions, before using this function. This function clears the internal buffers of a PI controller. You must call this function when you want the PI controller to be initialized. The init function must not be called together with PCLIB_CtrlPI, unless a periodic clearing of buffers is required.

2.3.1 Available versions

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

Table 2-5. Init function versions

Function name	Input type	Parameters	Result type
PCLIB_CtrlPIInit_F16	frac16_t	PCLIB_CTRL_PI_T_F16 *	void
	The inputs are a 16-bit fractional initial value and a pointer to the controller's parameters structure. It clears the internal integral accumulator buffer.		

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

Table 2-6. Function versions

Function name	Input type	Parameters	Result type
PCLIB_CtrlPI_F16	frac16_t	PCLIB_CTRL_PI_T_F16 *	frac16_t
	The error input is a 16-bit fractional value within the range <-1 ; 1). The parameters are pointed to by an input pointer. The function returns a 16-bit fractional value in the range <f16LowerLimit ; f16UpperLimit>.		

2.3.2 PCLIB_CTRL_PI_T_F16

Variable name	Input type	Description
f16Kp	frac16_t	Proportional gain. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16Ki	frac16_t	Integral gain. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16PreviousIntegralOutput	frac16_t	Internal integral accumulator. Controlled by the algorithm.
f16IntegralUpperLimit	frac16_t	Upper limit of the the integral accumulator. These parameters must be greater than f16IntegralLowerLimit. Set by the user.
f16IntegralLowerLimit	frac16_t	Lower limit of the the integral accumulator. These parameters must be lower than f16IntegralUpperLimit. Set by the user.
f16UpperLimit	frac16_t	Upper limit of the the controller's output. These parameters must be greater than f16LowerLimit. Set by the user.

Table continues on the next page...

Variable name	Input type	Description
f16LowerLimit	frac16_t	Lower limit of the the controller's output. These parameters must be lower than f16UpperLimit. Set by the user.

2.3.3 Declaration

The available PCLIB_CtrlPI functions have the following declarations:

```
void PCLIB_CtrlPIInit_F16(PCLIB_CTRL_PI_T_F16 *psParam)
frac16_t PCLIB_CtrlPI_F16(frac16_t f16InErr, PCLIB_CTRL_PI_T_F16 *psParam)
```

2.3.4 Function use

The use of the PCLIB_CtrlPIInit_F16 and PCLIB_CtrlPI functions is shown in the following example:

```
#include "pclib.h"

static frac16_t f16Result, f16InErr;
static PCLIB_CTRL_PI_T_F16 sParam;

void Isr(void);

void main(void)
{
    f16InErr = FRAC16(-0.4);
    sParam.f16Kp = FRAC16(0.1);
    sParam.f16Ki = FRAC16(0.2);
    sParam.f16IntegralUpperLimit = FRAC16(0.9);
    sParam.f16IntegralLowerLimit = FRAC16(-0.9);
    sParam.f16UpperLimit = FRAC16(0.9);
    sParam.f16LowerLimit = FRAC16(-0.9);

    PCLIB_CtrlPIInit_F16(&psParam);
}

/* Periodical function or interrupt */
void Isr()
{
   f16Result = PCLIB_CtrlPI_F16(f16InErr, &sParam);
}
```

2.4 PCLIB_CtrlPIandLPFilter

The PCLIB_CtrlPIandLPFilter function calculates the Proportional-Integral (PI) compensation block, along with the low-pass filter. The low-pass filter's pole and zero are placed at much higher frequency to compensate for the output capacitor ESR. It can be represented as follows:

$$Output = (Kp + \frac{Ki}{s}) \cdot [\frac{(s+a)}{(s+b)}]$$

Equation 14.

It increases the system performance even at the high frequency (in bode plot frequency domain) of system operations. This is equivalent to:

$$\frac{y[s]}{x[s]} = \frac{(s-Z1)(s-Z2)}{(s-P1)(s-P2)}$$

Equation 15.

where y[s] is the output, and x[s] is the input to the system. This control law has two poles (P1 and P2) and two zeroes (Z1 and Z2). The value or the placement of these poles and zeroes in the bode plot influence the stability and performance of the control loop and the system. The z-domain controller Gc(z) at sampling time Ts is expressed using the Tustin method as follows:

$$\frac{y[t]}{x[t]} = \frac{(b2 \cdot z^{-2} + b1 \cdot z^{-1} + b0)}{(1 - a2 \cdot z^{-2} - a1 \cdot z^{-1})}$$

Equation 16.

$$y[t] - a1 \cdot y[t] \cdot z^{-1} - a2 \cdot y[t] \cdot z^{-2} = b0 \cdot x[t] + b1 \cdot x[t] \cdot z^{-1} + b2 \cdot x[t] \cdot z^{-2}$$

Equation 17.

where:

- y[t] = y[n] is the present output
- $y[t] \cdot z^{-1}$ = y[n-1] is the previous output
- $y[t] \cdot z^{-2}$ = y[n-2] is the previous to previous output
- x[t] = x[n] is the present error
- $x[t] \cdot z^{-1}$ = x[n-1] is the previous error
- $x[t] \cdot z^{-2}$ = x[n-2] is the previous to previous error
- b0, b1, b2, a1, and a2 are the control coefficients and functions of Z1, Z2, P1, P2, and sampling time Ts.

$$y[n] = a1 \cdot y[n-1] + a2 \cdot y[n-2] + b0 \cdot x[n] + b1 \cdot x[n-1] + b2 \cdot x[n-2]$$

Equation 18.

For a proper use of this function, it is recommended to initialize the function's data by the PCLIB_CtrlPIandLPInit functions, before using the function. This function clears the internal buffers of the PIandLP controller. You must call this function when you want the PIandLP controller to be initialized. The init function must not be called together with PCLIB_CtrlPIandLPFilter, unless a periodic clearing of buffers is required.

2.4.1 Available versions

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

Table 2-7. Init function versions

Function name	Input type	Parameters	Result type
PCLIB_CtrlPIandLPInit_F16	frac16_t	PCLIB_CTRL_PI_LP_T_F16 *	void
	The inputs are a 16-bit fractional initial value and a pointer to the controller's parameters structure. It clears the internal delay parameter buffers of the controller.		

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

Table 2-8. Function versions

Function name	Input type	Parameters	Result type
PCLIB_CtrlPIandLP_F16	frac16_t	PCLIB_CTRL_PI_LP_T_F16 *	frac16_t
	The error input is a 16-bit fractional value within the range <-1 ; 1). The parameters are pointed to by an input pointer. The function returns a 16-bit fractional value in the range <-1 ; 1).		

2.4.2 PCLIB_CTRL_PI_LP_T_F16

Variable name	Input type	Description
f16CoeffB0	frac16_t	Control coefficient for the present error. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16CoeffB1	frac16_t	Control coefficient for the past error. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16CoeffB2	frac16_t	Control coefficient for the past to past error. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16CoeffA1	frac16_t	Control coefficient for the past result. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16CoeffA2	frac16_t	Control coefficient for the past to past result. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.

Table continues on the next page...

Variable name	Input type	Description
f16DelayX1	frac16_t	Delay parameter for the past error. Controlled by the algorithm.
f16DelayX2	frac16_t	Delay parameter for the past to past error. Controlled by the algorithm.
f16DelayY1	frac16_t	Delay parameter for the past result. Controlled by the algorithm.
f16DelayY2	frac16_t	Delay parameter for the past to past result. Controlled by the algorithm.

2.4.3 Declaration

The available PCLIB_CtrlPIandLPFilter functions have the following declarations:

```
void PCLIB_CtrlPIandLPInit_F16(PCLIB_CTRL_PI_LP_T_F16 *psParam)
frac16_t PCLIB_CtrlPIandLP_F16(frac16_t f16InErr, PCLIB_CTRL_PI_LP_T_F16 *psParam)
```

2.4.4 Function use

The use of the PCLIB_CtrlPIandLPInit_F16 and PCLIB_CtrlPIandLPFilter functions is shown in the following example:

```
#include "pclib.h"

static frac16_t f16Result, f16InErr;
static PCLIB_CTRL_PI_LP_T_F16 sParam;

void Isr(void);

void main(void)
{
    f16InErr = FRAC16(-0.4);
    sParam.f16CoeffB0 = FRAC16(0.1);
    sParam.f16CoeffB1 = FRAC16(0.2);
    sParam.f16CoeffB2 = FRAC16(0.15);
    sParam.f16CoeffA1 = FRAC16(0.1);
    sParam.f16CoeffA2 = FRAC16(0.25);

    PCLIB_CtrlPIandLPInit_F16(&sParam);
}

/* Periodical function or interrupt */
void Isr()
{
   f16Result = PCLIB_CtrlPIandLP_F16(f16InErr, &sParam);
}
```

2.5 PCLIB_CtrlPID

The PCLIB_CtrlPID function calculates the Proportional-Integral-Derivative (PID) algorithm, according to the proportional (Kp), integral (Ki), and differential (Kd) coefficients. The controller output is limited, and you can define the limit values.

The PID algorithm in the continuous time domain is expressed as follows:

$$y(t) = Kp \cdot e(t) + \int_0^t (Ki \cdot e(t) \cdot dt) + Kd \cdot \frac{de(t)}{dt}$$

Equation 19.

where:

- e(t) is the input error in the continuous time domain
- y(t) is the controller output in the continuous time domain
- Kp is the proportional coefficient
- Ki is the integral coefficient
- Kd is the differential coefficient

It can be rewritten as:

$$Kp \cdot e(t) = Kp \cdot x(t)$$

Equation 20.

$$Ki \int_0^t e(t) \cdot dt = \frac{Ki}{1 - z^{-1}} \cdot x(t)$$

Equation 21.

$$Kd \cdot \frac{de(t)}{dt} = Kd \cdot (1 - z^{-1}) \cdot x(t)$$

Equation 22.

$$y(t) = \frac{(Kp + Ki + Kd) \cdot x(t) + (-Kp - 2 \cdot Kd) \cdot x(t) \cdot z^{-1} + Kd \cdot x(t) \cdot z^{-2}}{1 - z^{-1}}$$

Equation 23.

It can be further simplified as:

$K_p + K_i + K_d = KA$

$-K_p - 2K_d = KB$

$K_d = KC$

therefore:

$$y(t) = \frac{Ka \cdot x(t) + Kb \cdot x(t) \cdot z^{-1} + Kc \cdot x(t) \cdot z^{-2}}{1 - z^{-1}}$$

Equation 24.

$$y(t) = y(t) \cdot z^{-1} + Ka \cdot x(t) + Kb \cdot x(t) \cdot z^{-1} + Kc \cdot x(t) \cdot z^{-2}$$

Equation 25.

$$y[n] = y[n-1] + Ka \cdot x[n] + Kb \cdot x[n-1] + Kc \cdot x[n-2]$$

Equation 26.

where:

- y(t) = y[n] is the present output
- $y(t) \cdot z^{-1}$ = y[n-1] is the previous output
- x(t) = x[n] is the present error
- $x(t) \cdot z^{-1}$ = x[n-1] is the previous error
- $x(t) \cdot z^{-2}$ = x[n-2] is the previous to previous error

For a proper use of this function, it is recommended to initialize the function's data by the PCLIB_CtrlPIDInit functions, before using this function. This function clears the internal buffers of the PID controller. You must call this function when you want the PID controller to be initialized. The init function must not be called together with PCLIB_CtrlPID, unless a periodic clearing of buffers is required.

2.5.1 Available versions

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

Table 2-9. Init function versions

Function name	Input type	Parameters	Result type
PCLIB_CtrlPIDInit_F16	frac16_t	PCLIB_CTRL_PID_T_F16 *	void
	The inputs are a 16-bit fractional initial value and a pointer to the controller parameters' structure. It clears the internal delay parameter buffers of the controller.		

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

Table 2-10. Function versions

Function name	Input type	Parameters	Result type
PCLIB_CtrlPID_F16	frac16_t	PCLIB_CTRL_PID_T_F16 *	frac16_t
	The error input is a 16-bit fractional value within the range <-1 ; 1). The parameters are pointed to by an input pointer. The function returns a 16-bit fractional value in the range <f16LowerLimit ; f16UpperLimit>.		

2.5.2 PCLIB_CTRL_PID_T_F16

Variable name	Input type	Description
f16Ka	frac16_t	Control coefficient for the present error. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16Kb	frac16_t	Control coefficient for the past error. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16Kc	frac16_t	Control coefficient for the past to past error. The parameter is a 16-bit fractional value within the range <-1 ; 1). Set by the user.
f16DelayX1	frac16_t	Delay parameter for the past error. Controlled by the algorithm.
f16DelayX2	frac16_t	Delay parameter for the past to past error. Controlled by the algorithm.
f16DelayY1	frac16_t	Delay parameter for the past result. Controlled by the algorithm.
f16UpperLimit	frac16_t	Upper limit of the controller's output. This parameter must be greater than f16LowerLimit. Set by the user.
f16LowerLimit	frac16_t	Lower limit of the controller's output. This parameter must be lower than f16UpperLimit. Set by the user.

2.5.3 Declaration

The available PCLIB_CtrlPID functions have the following declarations:

```
void PCLIB_CtrlPIDInit_F16(PCLIB_CTRL_PID_T_F16 *psParam)
frac16_t PCLIB_CtrlPID_F16(frac16_t f16InErr, PCLIB_CTRL_PID_T_F16 *psParam)
```

2.5.4 Function use

The use of the PCLIB_CtrlPIDInit_F16 and PCLIB_CtrlPID functions is shown in the following example:

```
#include "pclib.h"

static frac16_t f16Result, f16InErr;
static PCLIB_CTRL_PID_T_F16 sParam;

void Isr(void);

void main(void)
{
    f16InErr = FRAC16(-0.4);
    sParam.f16Ka = FRAC16(0.1);
    sParam.f16Kb = FRAC16(0.2);
    sParam.f16Kc = FRAC16(0.15);
    sParam.f16UpperLimit = FRAC16(0.9);
    sParam.f16LowerLimit = FRAC16(-0.9);

    PCLIB_CtrlPIDInit_F16(&sParam);
}
```

```
/* Periodical function or interrupt */
void Isr()
{
   f16Result = PCLIB_CtrlPID_F16(f16InErr, &sParam);
}
```

Appendix A
Library types

A.1 bool_t

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

```
typedef unsigned short bool_t;
```

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

Table A-1. Data storage

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

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

A.2 uint8_t

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

```
typedef unsigned char uint8_t;
```

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

Table A-2. Data storage

Value	7	6	5	4	3	2	1	0
	Integer							
255	1	1	1	1	1	1	1	1
	F				F			
11	0	0	0	0	1	0	1	1
	0				B			
124	0	1	1	1	1	1	0	0
	7				C			
159	1	0	0	1	1	1	1	1
	9				F			

A.3 uint16_t

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

```
typedef unsigned short uint16_t;
```

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

Table A-3. Data storage

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

A.4 uint32_t

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

```
typedef unsigned long uint32_t;
```

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

Table A-4. Data storage

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

A.5 int8_t

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

```
typedef char int8_t;
```

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

Table A-5. Data storage

	7	6	5	4	3	2	1	0
Value	Sign	Integer						
127	0	1	1	1	1	1	1	1
	7				F			
-128	1	0	0	0	0	0	0	0
	8				0			
60	0	0	1	1	1	1	0	0
	3				C			
-97	1	0	0	1	1	1	1	1
	9				F			

A.6 int16_t

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

```
typedef short int16_t;
```

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

Table A-6. Data storage

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

A.7 int32_t

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

```
typedef long int32_t;
```

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

Table A-7. Data storage

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

A.8 frac8_t

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

```
typedef char frac8_t;
```

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

Table A-8. Data storage

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

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

A.9 frac16_t

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

```
typedef short frac16_t;
```

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

Table A-9. Data storage

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

Table continues on the next page...

Table A-9. Data storage (continued)

	8				0				0				0			
0.47357	0	0	1	1	1	1	0	0	1	0	0	1	1	1	1	0
	3				C				9				E			
-0.75586	1	0	0	1	1	1	1	1	0	1	0	0	0	0	0	0
	9				F				4				0			

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

A.10 frac32_t

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

```
typedef long frac32_t;
```

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

Table A-10. Data storage

	31		24 23		16 15		8 7	0
Value	S	Fractional						
0.9999999995	7	F	F	F	F	F	F	F
-1.0	8	0	0	0	0	0	0	0
0.02606645970	0	3	5	6	2	5	5	0
-0.3929787632	C	D	B	2	D	F	3	4

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

A.11 acc16_t

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

```
typedef short acc16_t;
```

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

Table A-11. Data storage

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

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

A.12 acc32_t

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

```
typedef long acc32_t;
```

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

Table A-12. Data storage

Value	31	24	23	16	15	8	7	0
	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 */
}
```

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number CM0PCLIBUG
Revision 3, 04/2016