AMCLIB User's Guide

ARM® Cortex® M4F

Document Number: CM4FAMCLIBUG
Rev. 1, 06/2016

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 Advanced Motor Control Library (AMCLIB) for the family of ARM Cortex M4F core-based microcontrollers. This library contains optimized functions.

1.1.2 Data types

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

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

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

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

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

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

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

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

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

1.1.3 API definition

AMCLIB 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
float_t	FLT	f

1.1.4 Supported compilers

AMCLIB for the ARM Cortex M4F core is written in . 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 *amclib.a* file.

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

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

The interfaces to the algorithms included in this library are combined into a single public interface include file, *amclib.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 AMCLIB 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\CM4F_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 Library path variable. If not, continue with the next section.

1.2.1 New project (without SDK)

This example uses the NXP MKV46F256xxx15 MCU, and the default installation path (C:\NXP\RTCESL\CM4F_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.

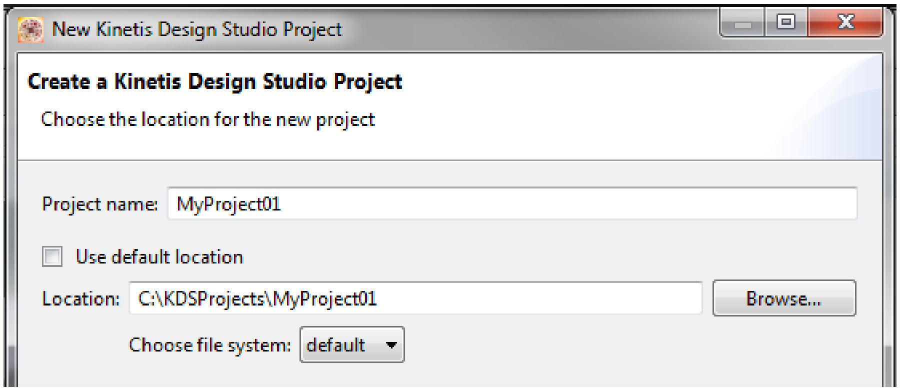

Figure 1-1. Project name and location

5. Expand the tree by clicking Processors, then . 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.

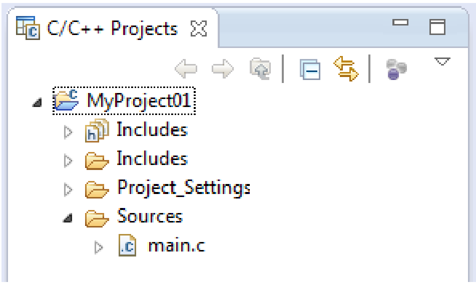

Figure 1-3. Project folder

7. Right-click the MyProject01 node in the Project Explorer, click Properties, or select Project > Properties from the menu. A project properties dialog appears.
8. Expand the C/C++ Build node and select Settings.
9. In the right-hand part, set the Float ABI as FP instructions (hard) and FPU type as fpv4_sp_d16. See Figure 1-4.

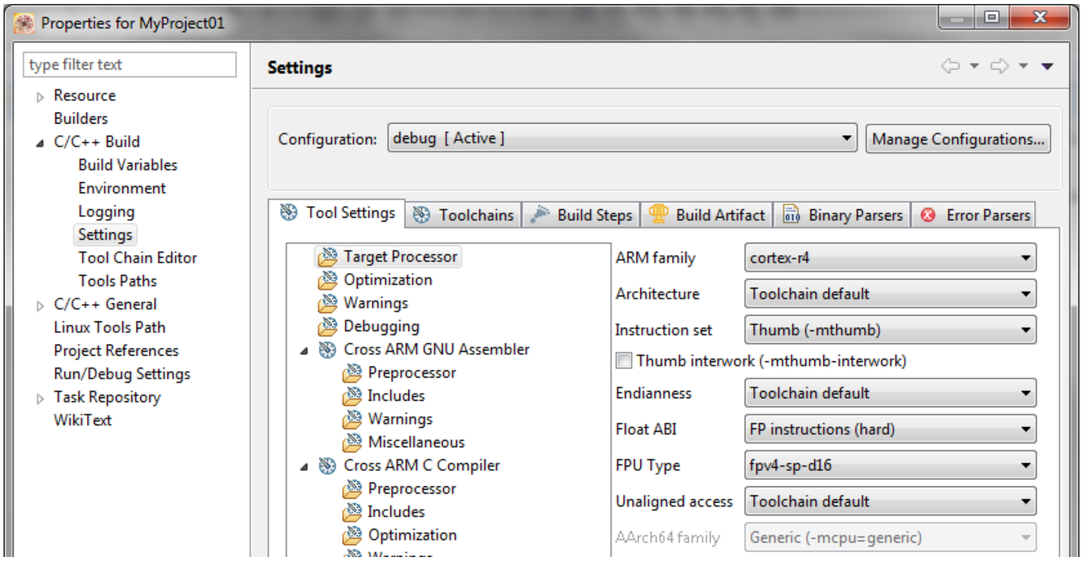

Figure 1-4. FPU setting

1.2.2 Library path variable

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

1. Right-click the MyProject01 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 1-5.

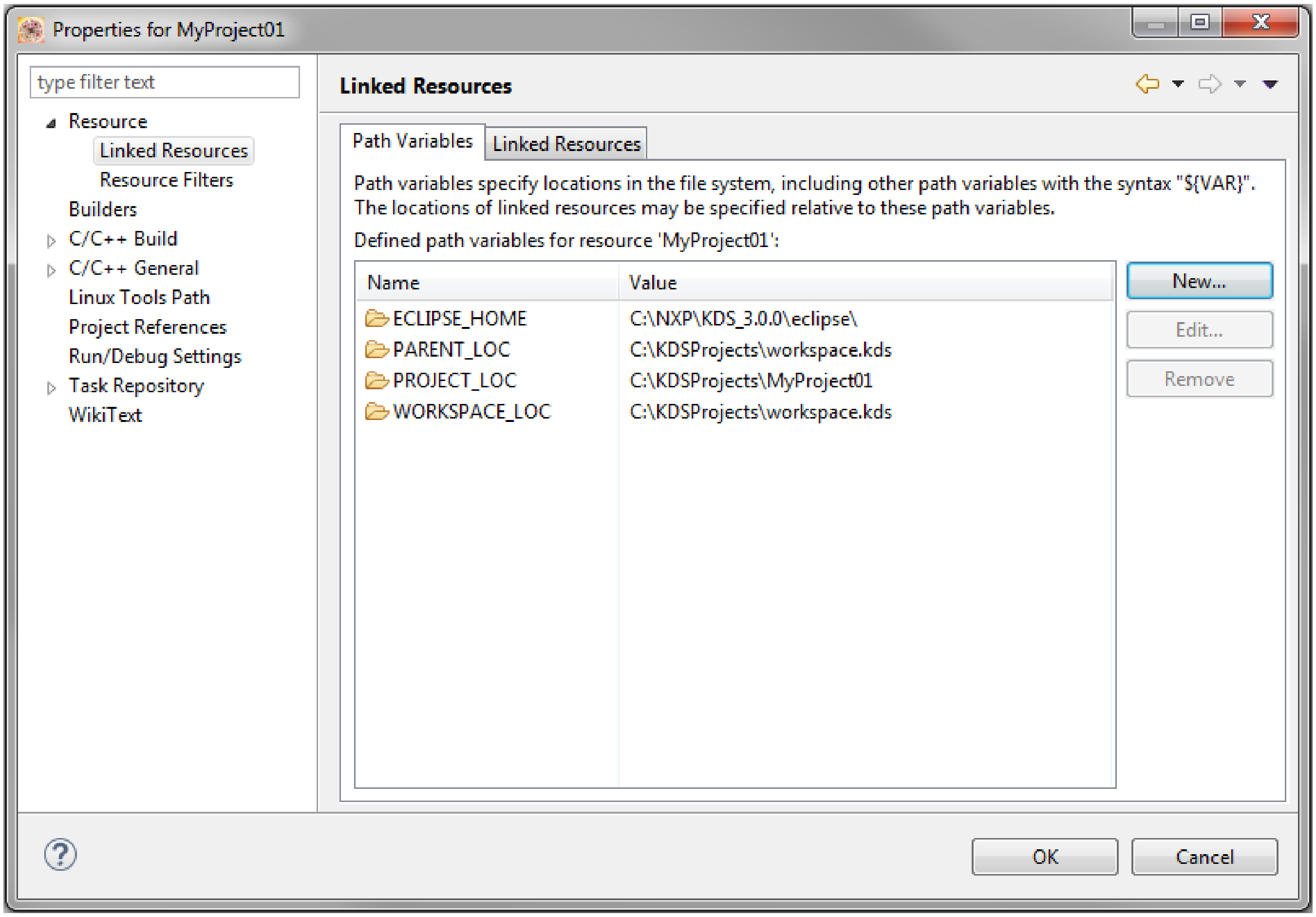

Figure 1-5. Project properties

3. Click the New… button in the right-hand side.
4. In the dialog that appears (see Figure 1-6), 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\CM4F_RTCESL_4.3_KDS. Click OK.

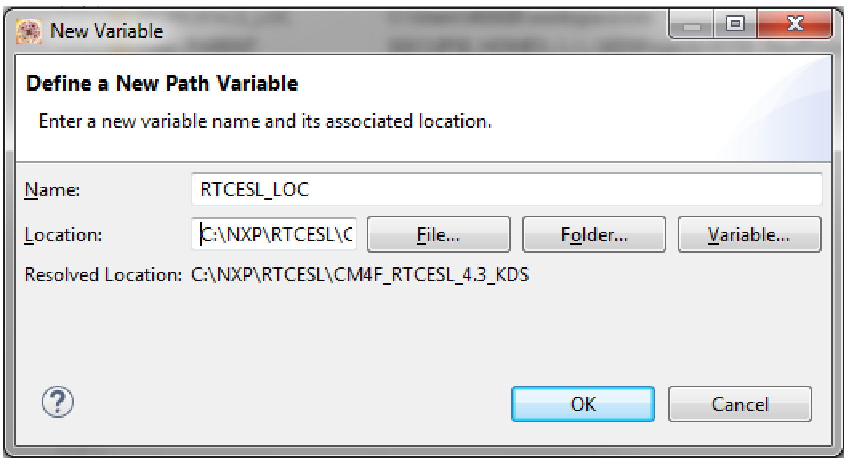

Figure 1-6. New variable

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 1-7), type this variable name into the Name box: RTCESL_LOC.
9. Type the library parent folder path into the Value box: C:\NXP\RTCESL\CM4F_RTCESL_4.3_KDS.
10. Tick the Add to all configurations box to use this variable in all configurations. See Figure 1-7.
11. Click OK.
12. In the previous dialog, click OK.

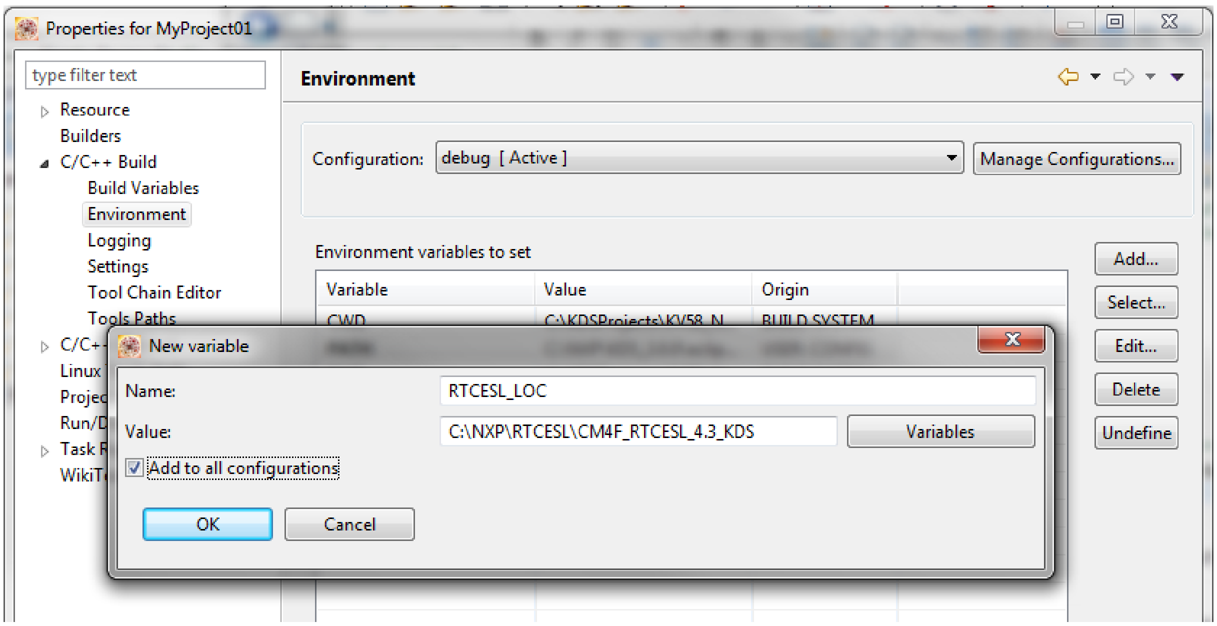

Figure 1-7. Environment variable

1.2.3 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 1-8.
5. Click Finish, and you will see the library folder linked in the project. See Figure 1-9.

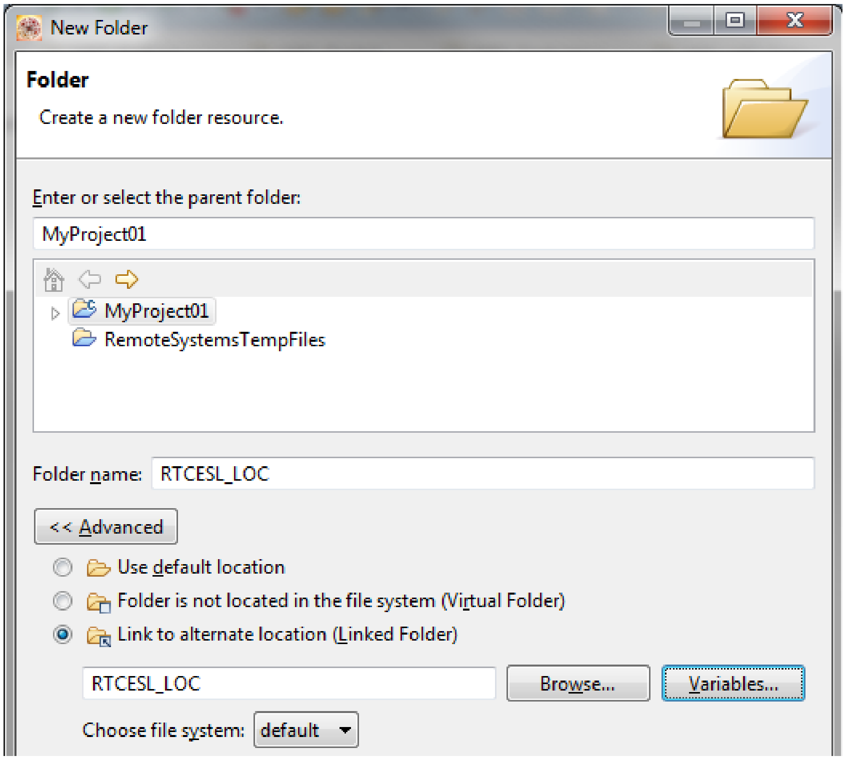

Figure 1-8. Folder link

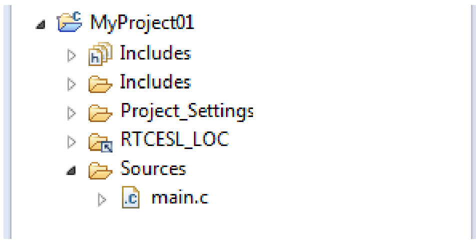

Figure 1-9. Projects libraries paths

1.2.4 Library path setup

AMCLIB requiresMLIB and GDFLIB and GFLIB and GMCLIB to be included too. These steps show how to include all dependent modules:

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 1-11.
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 1-10): ${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}\GFLIB.
8. Click OK, and then click the Add… button.
9. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box by adding the following: ${RTCESL_LOC}\GDFLIB.
10. Click OK, and then click the Add… button.
11. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box by adding the following: ${RTCESL_LOC}\GMCLIB.
12. Click OK, and then click the Add… button.
13. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box by adding the following: ${RTCESL_LOC}\AMCLIB.
14. Click OK, and the paths will be visible in the list. See Figure 1-11.

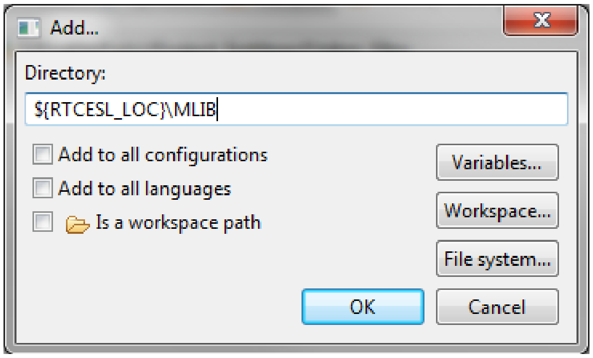

Figure 1-10. Library path inclusion

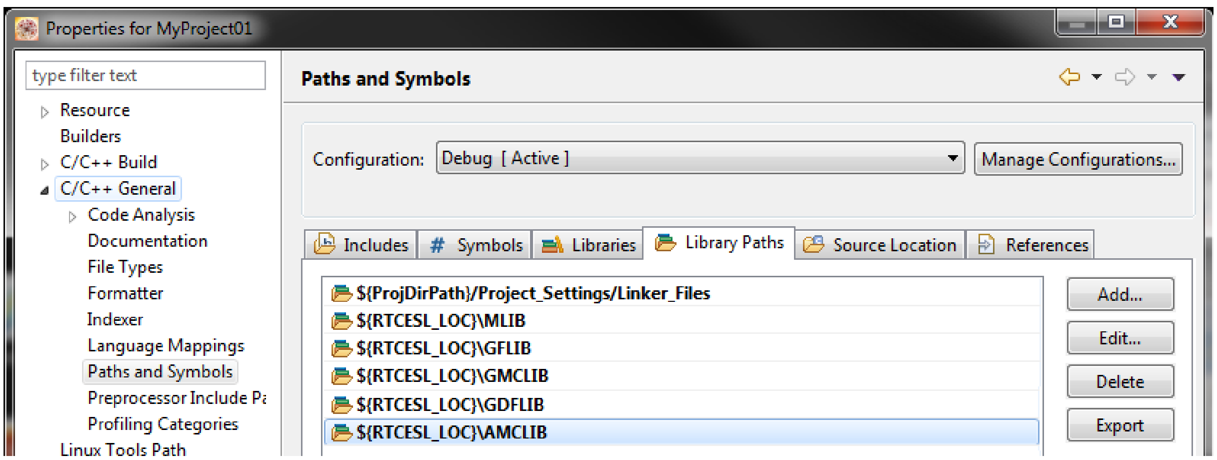

Figure 1-11. Library paths

15. After adding the library paths, add the library files. Click the Libraries tab. See Figure 1-13.
16. Click the Add… button on the right, and a dialog appears.
17. Type the following into the File text box (see Figure 1-12): :mlib.a
18. Click OK, and then click the Add… button.

19. Type the following into the File text box: :gflib.a
20. Click OK, and then click the Add… button.
21. Type the following into the File text box: :gdflib.a
22. Click OK, and then click the Add… button.
23. Type the following into the File text box: :gmclib.a
24. Click OK, and then click the Add… button.
25. Type the following into the File text box: :amclib.a
26. Click OK, and you will see the libraries added in the list. See Figure 1-13.

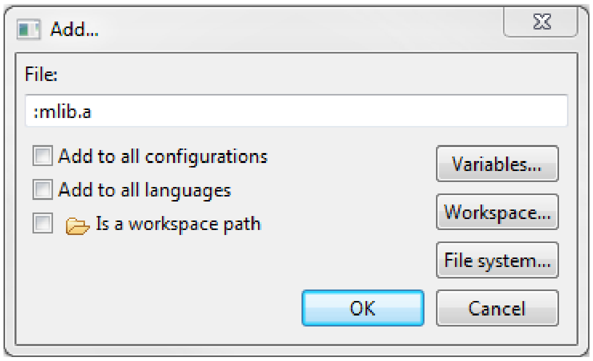

Figure 1-12. Library file inclusion

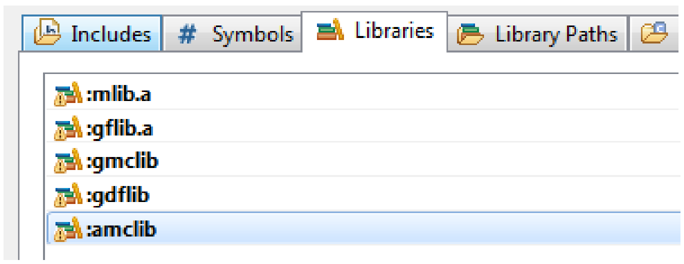

Figure 1-13. Libraries

27. In the right-hand dialog, select the Includes tab, and click GNU C in the Languages list. See Figure 1-15.
28. Click the Add… button on the right, and a dialog appears. See Figure 1-14.
29. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box to be: ${RTCESL_LOC}\MLIB\Include
30. Click OK, and then click the Add… button.
31. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box to be: ${RTCESL_LOC}\GFLIB\Include
32. Click OK, and then click the Add… button.
33. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box to be: ${RTCESL_LOC}\GDFLIB\Include
34. Click OK, and then click the Add… button.
35. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box to be: ${RTCESL_LOC}\GMCLIB\Include
36. Click OK, and then click the Add… button.

37. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box to be: ${RTCESL_LOC}\AMCLIB\Include
38. Click OK, and you will see the paths added in the list. See Figure 1-15. Click OK.

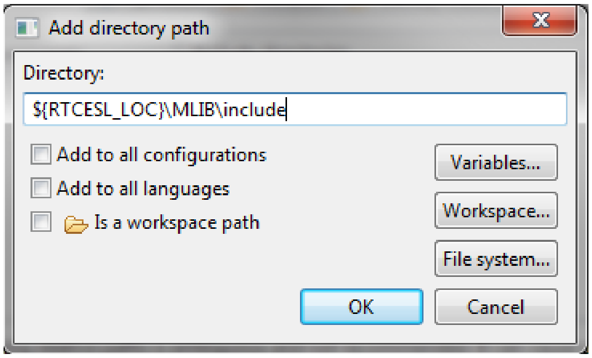

Figure 1-14. Library include path addition

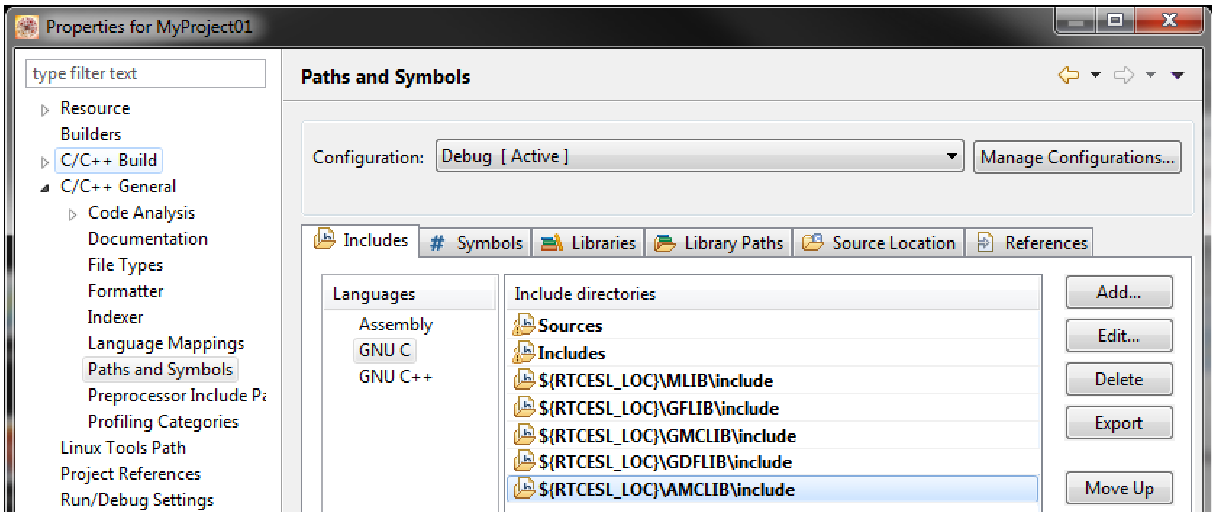

Figure 1-15. Compiler setting

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_fp.h"
#include "gflib_fp.h"
#include "gdflib_fp.h"
#include "gmclib_fp.h"
#include "amclib_fp.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 AMCLIB 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 \CM4F_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 Linking the files into the project chapter otherwise read next chapter.

1.3.1 NXP pack installation for new project (without SDK)

This example uses the NXP part, and the default installation path (C:\NXP\RTCESL \CM4F_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 1-16.
7. When installed, the button has the "Up to date" title. Now close the Pack Installer.

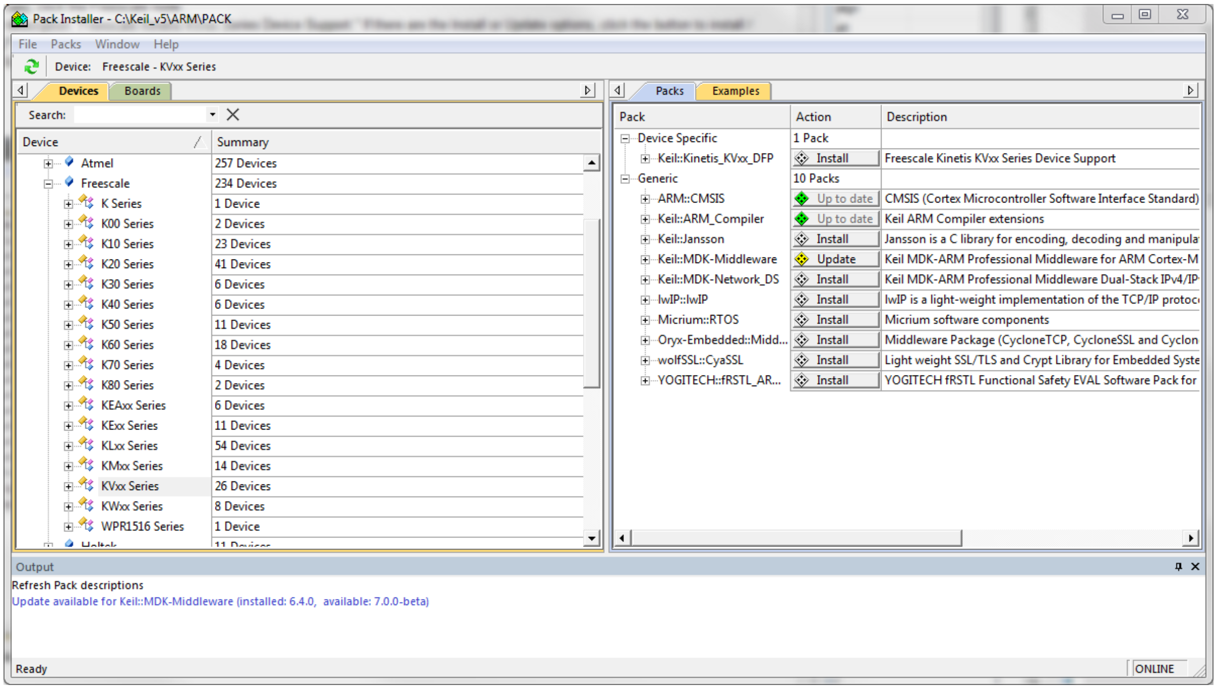

Figure 1-16. Pack Installer

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 1-17.

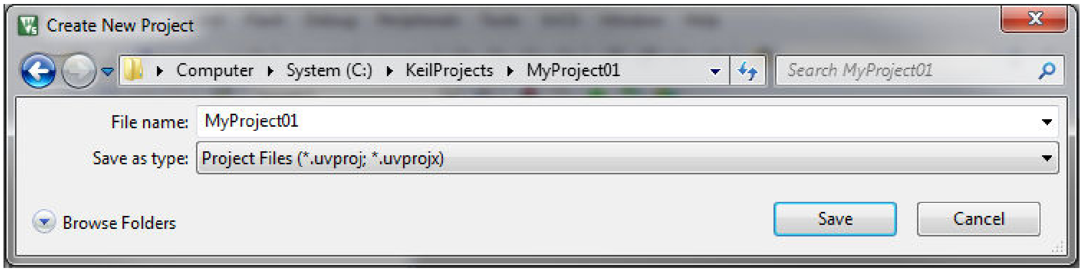

Figure 1-17. Create New Project dialog

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 MKV46F256xxx15 node, and then click OK. See Figure 1-18.

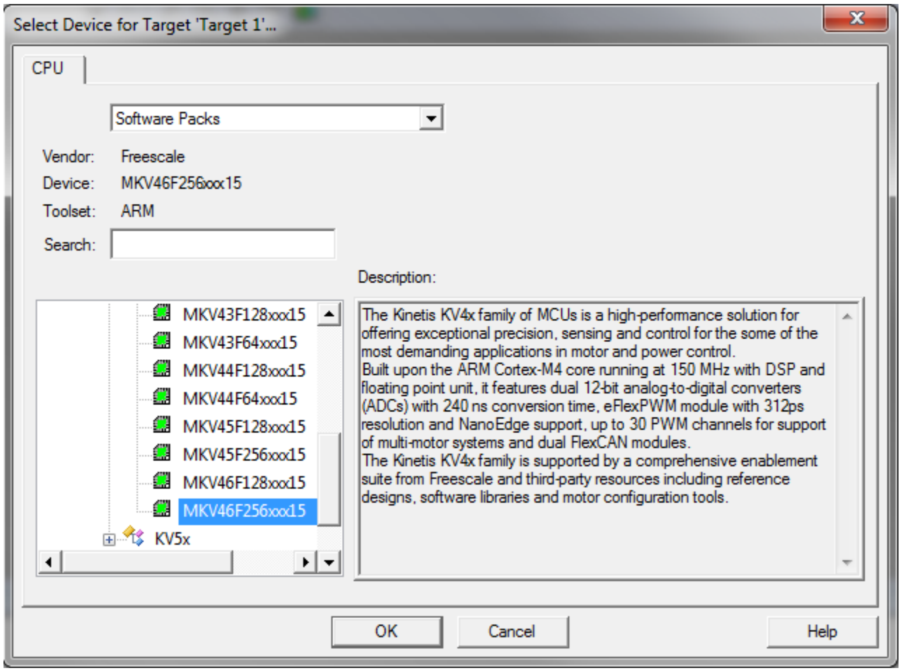

Figure 1-18. Select Device dialog

8. In the next dialog, expand the Device node, and tick the box next to the Startup node. See Figure 1-19.
9. Expand the CMSIS node, and tick the box next to the CORE node.

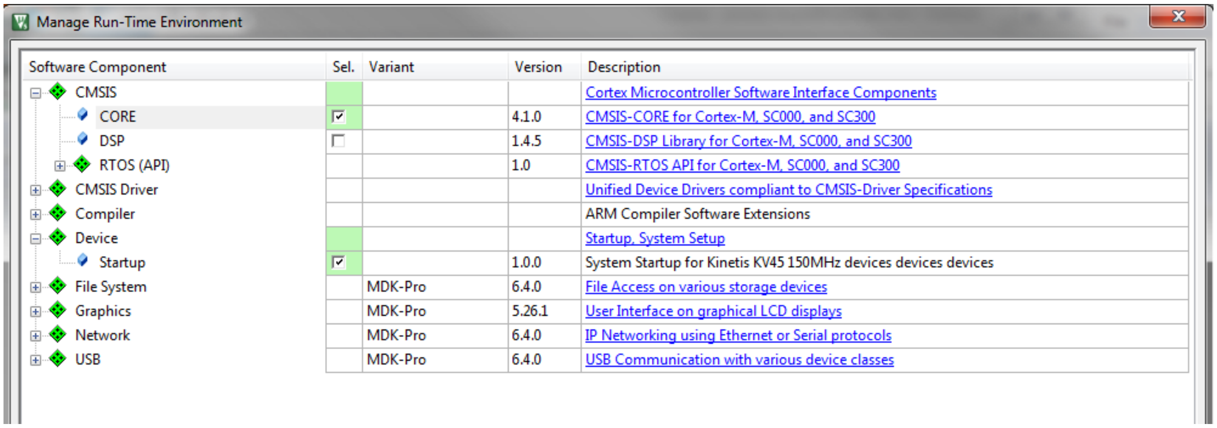

Figure 1-19. Manage Run-Time Environment dialog

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 1-20.

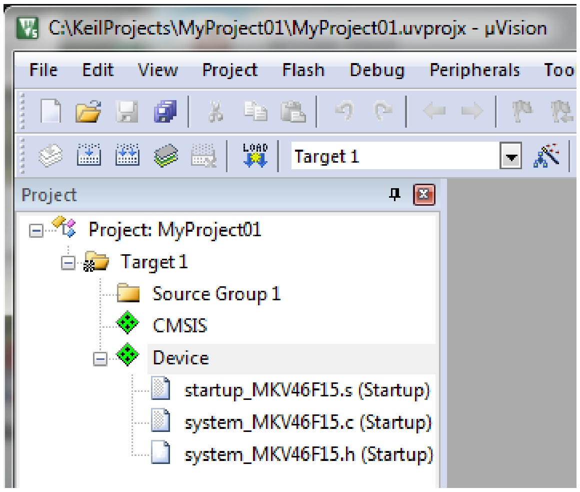

Figure 1-20. Project

11. In the main menu, go to Project > Options for Target 'Target1'…, and a dialog appears.
12. Select the Target tab.
13. Select Use Single Precision in the Floating Point Hardware option. See Figure 1-20.

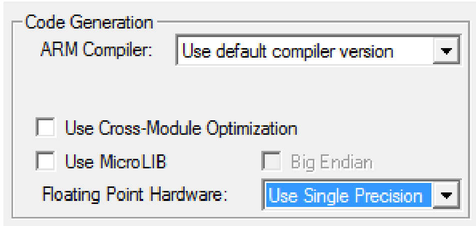

Figure 1-21. FPU

1.3.3 Linking the files into the project

AMCLIB requires MLIB and GDFLIB and GFLIB and GMCLIB 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 \CM4F_RTCESL_4.3_KEIL\MLIB\Include, and select the *mlib_fp.h* file. If the file does not appear, set the Files of type filter to Text file. Click Add. See Figure 1-22.

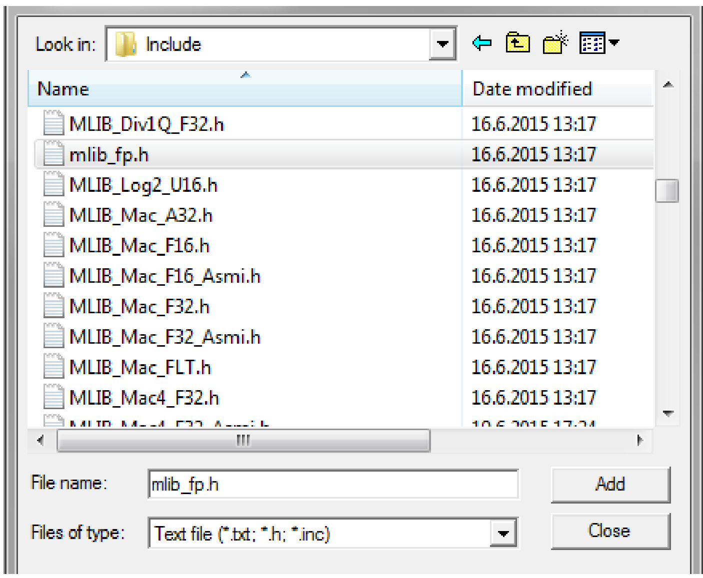

Figure 1-22. Adding .h files dialog

5. Navigate to the parent folder C:\NXP\RTCESL\CM4F_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 1-23.

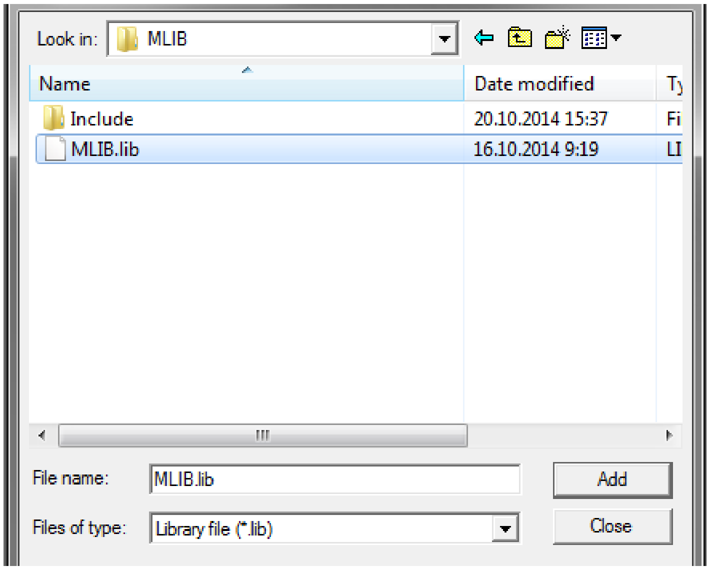

Figure 1-23. Adding .lib files dialog

6. Navigate into the library installation folder C:\NXP\RTCESL \CM4F_RTCESL_4.3_KEIL\GFLIB\Include, and select the *gflib_fp.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\CM4F_RTCESL_4.3_KEIL\GFLIB, and select the *gflib.lib* file. If the file does not appear, set the Files of type filter to Library file. Click Add.
8. Navigate into the library installation folder C:\NXP\RTCESL \CM4F_RTCESL_4.3_KEIL\GDFLIB\Include, and select the *gdflib_fp.h* file. If the file does not appear, set the Files of type filter to Text file. Click Add.
9. Navigate to the parent folder C:\NXP\RTCESL\CM4F_RTCESL_4.3_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.
10. Navigate into the library installation folder C:\NXP\RTCESL \CM4F_RTCESL_4.3_KEIL\GMCLIB\Include, and select the *gmclib_fp.h* file. If the file does not appear, set the Files of type filter to Text file. Click Add.
11. Navigate to the parent folder C:\NXP\RTCESL\CM4F_RTCESL_4.3_KEIL \GMCLIB, and select the *gmclib.lib* file. If the file does not appear, set the Files of type filter to Library file. Click Add.
12. Navigate into the library installation folder C:\NXP\RTCESL \CM4F_RTCESL_4.3_KEIL\AMCLIB\Include, and select the *amclib_fp.h* file. If the file does not appear, set the Files of type filter to Text file. Click Add.
13. Navigate to the parent folder C:\NXP\RTCESL\CM4F_RTCESL_4.3_KEIL \AMCLIB, and select the *amclib.lib* file. If the file does not appear, set the Files of type filter to Library file. Click Add.
14. Now, all necessary files are in the project tree; see Figure 1-24. Click Close.

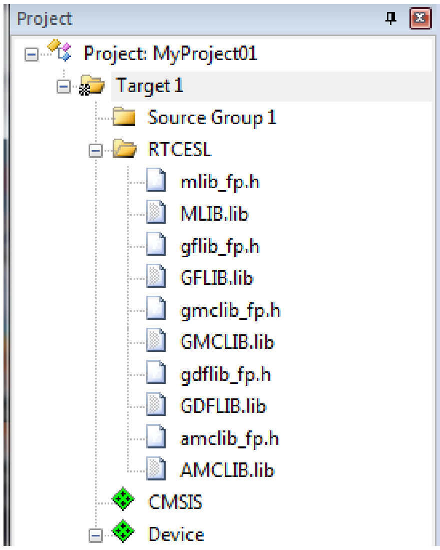

Figure 1-24. Project workspace

1.3.4 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 1-25.
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\CM4F_RTCESL_4.3_KEIL\MLIB\Include"
   - "C:\NXP\RTCESL\CM4F_RTCESL_4.3_KEIL\GFLIB\Include"
   - "C:\NXP\RTCESL\CM4F_RTCESL_4.3_KEIL\GDFLIB\Include"
   - "C:\NXP\RTCESL\CM4F_RTCESL_4.3_KEIL\GMCLIB\Include"
   - "C:\NXP\RTCESL\CM4F_RTCESL_4.3_KEIL\AMCLIB\Include"
4. Click OK.
5. Click OK in the main dialog.

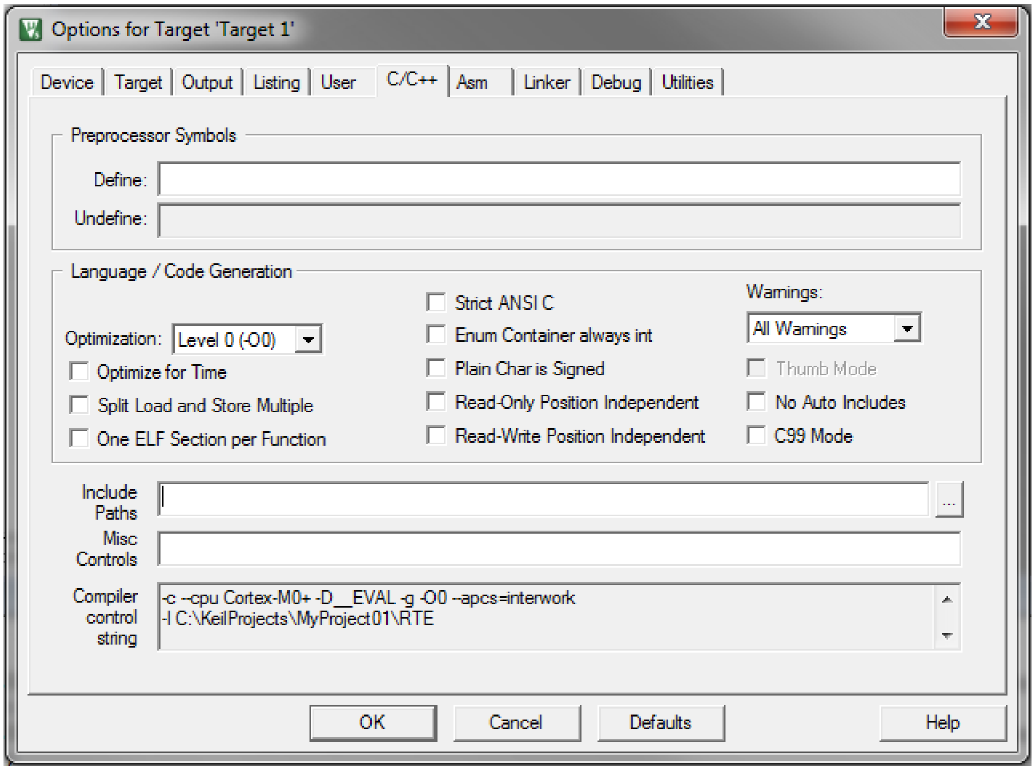

Figure 1-25. Library path addition

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 1-26.

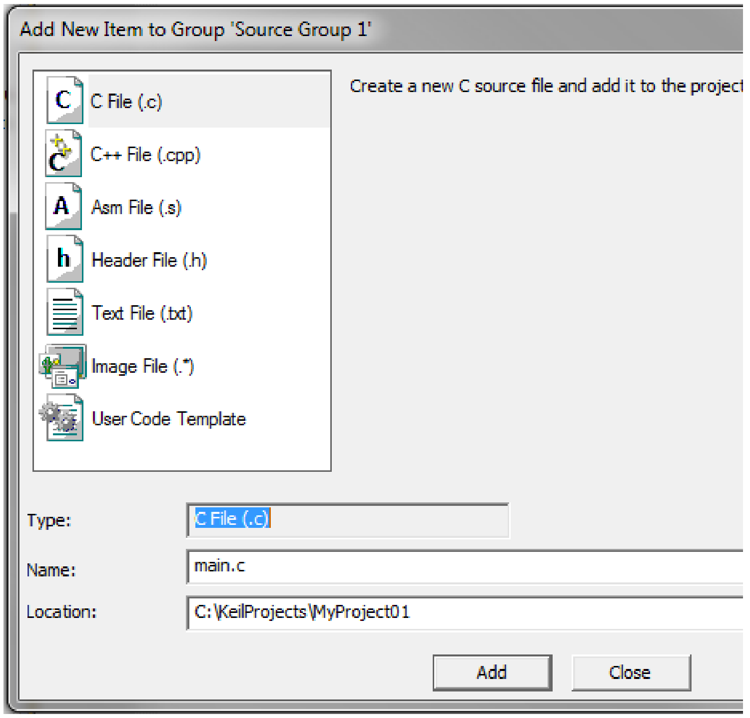

Figure 1-26. Adding new source file dialog

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_fp.h"
#include "gflib_fp.h"
#include "gdflib_fp.h"
#include "gmclib_fp.h"
#include "amclib_fp.h"

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

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

1.4 Library integration into project (IAR Embedded Workbench)

This section provides a step-by-step guide on how to quickly and easily include the AMCLIB 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\CM4F_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 Linking the files into the project chapter otherwise read next chapter.

1.4.1 New project (without SDK)

This example uses the NXP MKV46F256xxx15 part, and the default installation path (C:\NXP\RTCESL\CM4F_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 1-27.

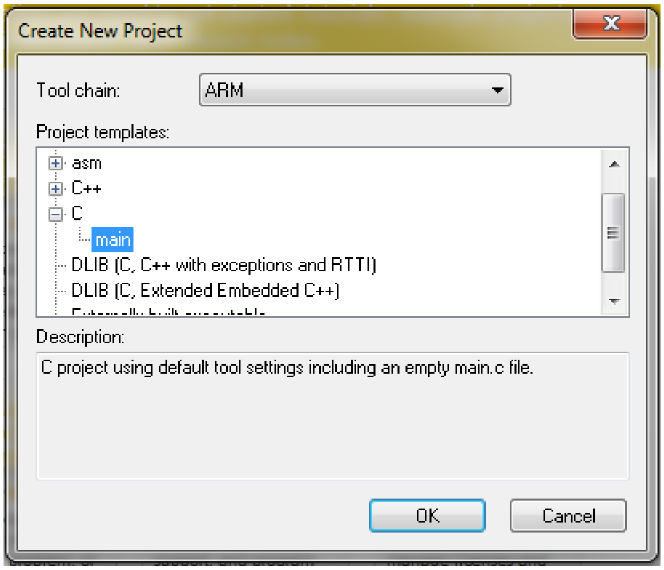

Figure 1-27. Create New Project dialog

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 1-28.

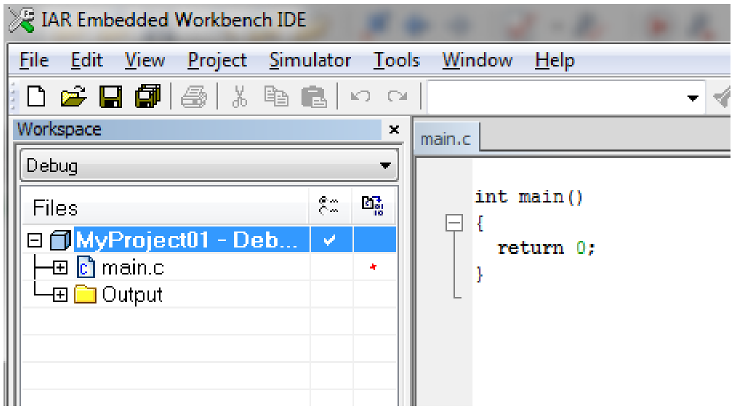

Figure 1-28. New project

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 > KV4x > NXP MKV46F256xxx15. Select VFPv4 single precision in the FPU option. Click OK. See Figure 1-29.

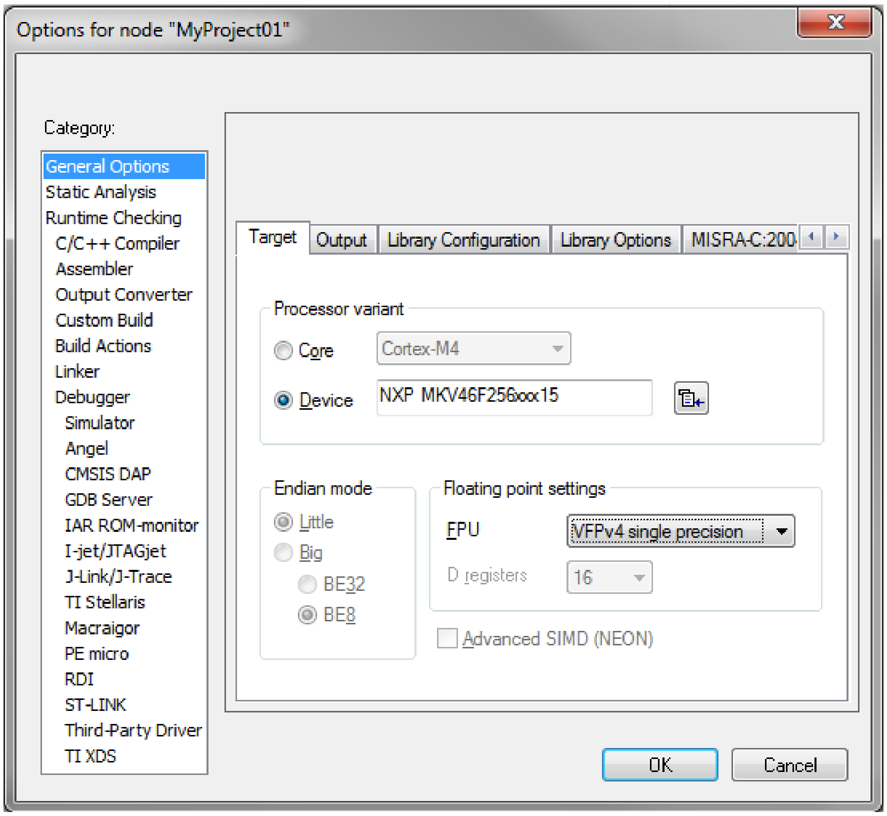

Figure 1-29. Options dialog

1.4.2 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 1-30.

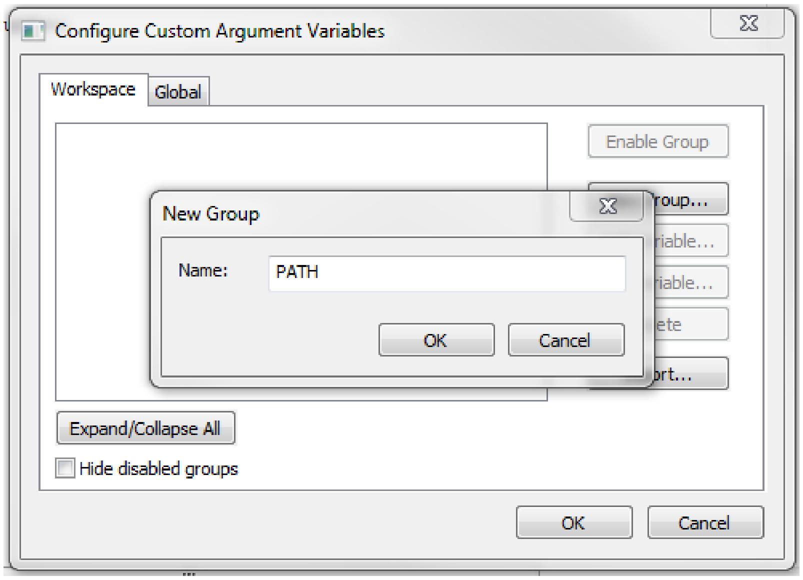

Figure 1-30. New Group

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\CM4F_RTCESL_4.3_IAR. Click OK.
6. In the main dialog, click OK. See Figure 1-31.

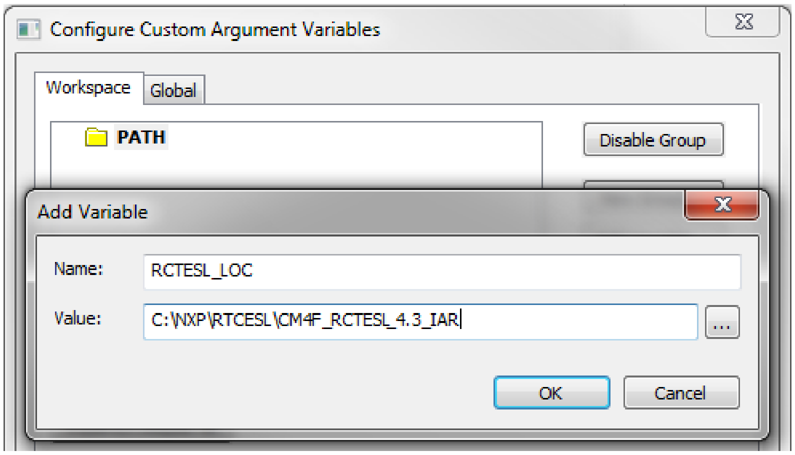

Figure 1-31. New variable

1.4.3 Linking the files into the project

AMCLIB requires MLIB and GDFLIB and GFLIB and GMCLIB 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 1-33.
5. Navigate into the library installation folder C:\NXP\RTCESL \CM4F_RTCESL_4.3_IAR\MLIB\Include, and select the *mlib_fp.h* file. (If the file does not appear, set the file-type filter to Source Files.) Click Open. See Figure 1-32.
6. Navigate into the library installation folder C:\NXP\RTCESL \CM4F_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.

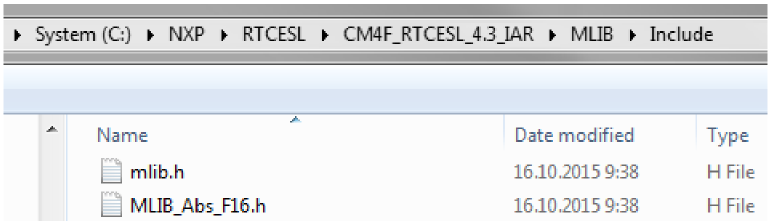

Figure 1-32. Add Files dialog

7. Click on the RTCESL node, go to Project > Add Group…, and create a GFLIB subgroup.

8. Click on the newly created node GFLIB, and go to the main menu Project > Add Files….
9. Navigate into the library installation folder C:\NXP\RTCESL \CM4F_RTCESL_4.3_IAR\GFLIB\Include, and select the *gflib_fp.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 \CM4F_RTCESL_4.3_IAR\GFLIB, and select the *gflib.a* file. If the file does not appear, set the file-type filter to Library / Object files. Click Open.
11. Click on the RTCESL node, go to Project > Add Group…, and create a GDFLIB subgroup.
12. Click on the newly created node GDFLIB, and go to the main menu Project > Add Files….
13. Navigate into the library installation folder C:\NXP\RTCESL \CM4F_RTCESL_4.3_IAR\GDFLIB\Include, and select the *gdflib_fp.h* file. (If the file does not appear, set the file-type filter to Source Files.) Click Open.
14. Navigate into the library installation folder C:\NXP\RTCESL \CM4F_RTCESL_4.3_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.
15. Click on the RTCESL node, go to Project > Add Group…, and create a GMCLIB subgroup.
16. Click on the newly created node GMCLIB, and go to the main menu Project > Add Files….
17. Navigate into the library installation folder C:\NXP\RTCESL \CM4F_RTCESL_4.3_IAR\GMCLIB\Include, and select the *gmclib_fp.h* file. If the file does not appear, set the file-type filter to Source Files. Click Open.
18. Navigate into the library installation folder C:\NXP\RTCESL \CM4F_RTCESL_4.3_IAR\GMCLIB, and select the *gmclib.a* file. If the file does not appear, set the file-type filter to Library / Object files. Click Open.
19. Click on the RTCESL node, go to Project > Add Group…, and create an AMCLIB subgroup.
20. Click on the newly created node AMCLIB, and go to the main menu Project > Add Files….
21. Navigate into the library installation folder C:\NXP\RTCESL \CM4F_RTCESL_4.3_IAR\AMCLIB\Include, and select the *amclib_fp.h* file. If the file does not appear, set the file-type filter to Source Files. Click Open.
22. Navigate into the library installation folder C:\NXP\RTCESL \CM4F_RTCESL_4.3_IAR\AMCLIB, and select the *amclib.a* file. If the file does not appear, set the file-type filter to Library / Object files. Click Open.
23. Now you will see the files added in the workspace. See Figure 1-33.

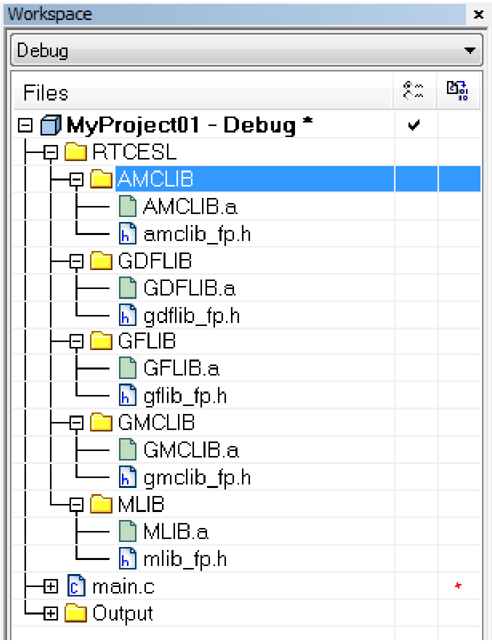

Figure 1-33. Project workspace

1.4.4 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
   - $RTCESL_LOC$\GFLIB\Include
   - $RTCESL_LOC$\GDFLIB\Include
   - $RTCESL_LOC$\GMCLIB\Include
   - $RTCESL_LOC$\AMCLIB\Include
5. Click OK in the main dialog. See Figure 1-34.

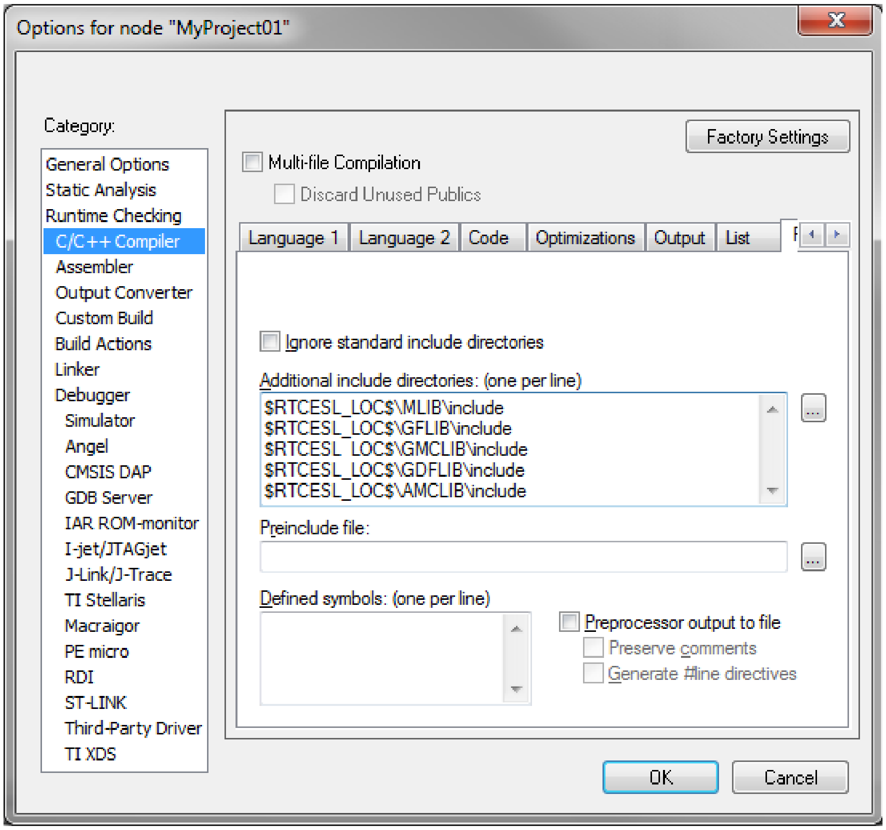

Figure 1-34. Library path adition

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_fp.h"
#include "gflib_fp.h"
#include "gdflib_fp.h"
#include "gmclib_fp.h"
#include "amclib_fp.h"
```

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

Chapter 2
Algorithms in detail

2.1 AMCLIB_ACIMRotFluxObsrv

The AMCLIB_ACIMRotFluxObsrv function calculates the ACIM flux estimate and its position (angle) from the available measured signals (currents and voltages). In the case of ACIM FOC, the rotor flux position (angle) is needed to perform the Park transformation.

The closed-loop flux observer is formed from the two most desirable open-loop estimators, which are referred to as the voltage model and the current model (as shown in Figure 2-1). The current model is used for low-speed operation and the voltage model is used for high-speed operation. A smooth transition between these two models is ensured by the PI controller.

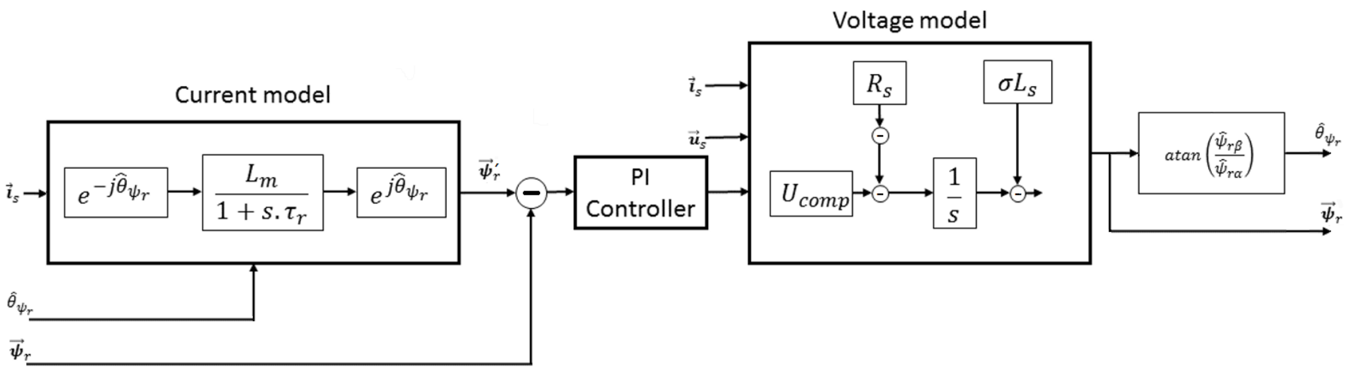

Figure 2-1. ACIM rotor flux observer block diagram

The voltage model (stator model) is used to estimate the stator flux-linkage vector or the rotor flux-linkage vector without a speed signal. The voltage model is derived by integrating the stator voltage equation in the stator stationary coordinates as:

$$\vec{u}_s = R_s \cdot \vec{i}_s + \frac{d\vec{\psi}_s}{dt}$$
$$\vec{\psi}_s = \int \left(\vec{u}_s - R_s \cdot \vec{i}_s\right) dt$$
$$\vec{\psi}_r = \frac{L_r}{L_m}\left(\vec{\psi}_s - L_s \cdot \sigma \cdot \vec{i}_s\right)$$

Equation 1

Expressed in discrete form as:

$$\psi_{s\alpha}(k) = \frac{\tau_1}{\tau_1 + T_s}\left[\psi_{s\alpha}(k-1) + T_s \cdot \left(u_{s\alpha}(k) - R_s \cdot i_{s\alpha}(k)\right)\right]$$
$$\psi_{s\beta}(k) = \frac{\tau_1}{\tau_1 + T_s}\left[\psi_{s\beta}(k-1) + T_s \cdot \left(u_{s\beta}(k) - R_s \cdot i_{s\beta}(k)\right)\right]$$
$$\psi_{r\alpha}(k) = \frac{L_r}{L_m}\left(\psi_{s\alpha}(k) - L_s \cdot \sigma \cdot i_{s\alpha}(k)\right)$$
$$\psi_{r\beta}(k) = \frac{L_r}{L_m}\left(\psi_{s\beta}(k) - L_s \cdot \sigma \cdot i_{s\beta}(k)\right)$$

Equation 2

where:

- u_s is the stator voltage vector
- i_s is the stator current vector
- Ψ_s is the stator flux-linkage vector
- Ψ_r is the rotor flux-linkage vector
- ω_r is the rotor electrical angular speed
- ω_s is the electrical angular slip speed
- R_s is the stator resistance
- R_r is the rotor equivalent resistance
- L_s is the stator equivalent inductance
- L_r is the rotor equivalent inductance
- L_m is the mutual equivalent inductance
- τ_r is the motor electrical time constant
- T_s is the sample time
- σ is the motor leakage coefficient

These equations show that the rotor flux linkage is basically the difference between the stator flux-linkage and the leakage flux. The rotor flux equation is used to estimate the respective flux-linkage vector, corresponding angle. The argument Ψ_r of the rotor flux-linkage vector is the rotor field angle $\theta_{\Psi r}$ calculated as:

$$\theta_{\psi_r} = \operatorname{atan}\left(\frac{\psi_{r\beta}}{\psi_{r\alpha}}\right)$$

Equation 3

The voltage model (stator model) is sufficiently robust and accurate at higher stator frequencies. Two basic deficiencies can degrade this model as the speed reduces: the integration problem, and model's sensitivity to stator resistance mismatch.

The current model (rotor model) is derived from the differential equation of the rotor winding. The stator coordinate implementation is:

$$\frac{d\overrightarrow{\psi_r}}{dt} = \frac{L_m}{\tau_r}\overrightarrow{i_s} - \frac{1}{\tau_r}\overrightarrow{\psi_r} - j\omega_{slip} \cdot \overrightarrow{\psi_r}$$

Equation 4

When applying field-oriented control assumptions (such as $\Psi_{rq} = 0$), then the rotor flux estimated by the current model in the synchronous rotating frame is:

$$\frac{d\overrightarrow{\psi_{rd}}}{dt} = -\frac{1}{\tau_r}\overrightarrow{\psi_{rd}} + \frac{L_m}{\tau_r}\overrightarrow{i_{sd}}$$

Equation 5

In discrete form:

$$\psi_{rd}(k) = \frac{\tau_r}{\tau_r + T_s}\left[\psi_{rd}(k-1) + T_s\frac{L_m}{\tau_r}i_{sd}(k)\right]$$

Equation 6

The accuracy of the rotor model depends on correct model parameters. It is the rotor time constant in particular that determines the accuracy of the estimated field angle (the most critical variable in a vector-controlled drive).

2.1.1 Available versions

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

Table 2-1. Init version

Function name	Parameters	Result type
AMCLIB_ACIMRotFluxObsrvInit_FLT	AMCLIB_ACIM_ROT_FLUX_OBSRV_T_FLT *	void
	The initialization does not have any input.	

Table 2-2. Function version

Function name	Input/output type		Result type
AMCLIB_ACIMRotFluxObsrv_FLT	Input	GMCLIB_2COOR_ALBE_T_FLT *	void
		GMCLIB_2COOR_ALBE_T_FLT *	

Table continues on the next page...

Table 2-2. Function version (continued)

Function name	Input/output type		Result type
	Parameters	AMCLIB_ACIM_ROT_FLUX_OBSRV_T_FLT *	
	Rotor flux observer with a 32-bit single precision floating-point inputs: stator current and voltage in alpha-beta coordinates. All are within the full range. The function does not return anything. All calculated variables are stored in the AMCLIB_ACIM_ROT_FLUX_OBSRV_T_FLT structure.		

2.1.2 AMCLIB_ACIM_ROT_FLUX_OBSRV_T_FLT type description

Variable name		Data type	Description
sPsiRotRDQ		GMCLIB_2COOR_DQ_T_FLT	The output rotor flux estimated structure calculated from the current model. The structure consists of the D and Q rotor flux components stored for the next steps. The quadrature component is forced to zero value - required by FOC.
sPsiRotSAlBe		GMCLIB_2COOR_ALBE_T_FLT	The output rotor flux estimated structure calculated from the voltage model. The structure consists of the alpha and beta rotor flux components stored for the next steps.
sPsiStatSAlBe		GMCLIB_2COOR_ALBE_T_FLT	The output stator flux estimated structure calculated from the voltage model. The structure consists of the alpha and beta stator flux components stored for the next steps.
fltTorque		float_t	The output estimated motor torque calculated as: $T = \frac{3}{2} \cdot P_P \cdot \frac{L_m}{L_r} (\Psi_{r\alpha} \cdot i_{s\beta} - \Psi_{r\beta} \cdot i_{s\alpha})$ The variable is a 32-bit single precision floating-point type value.
a32RotFluxPos		acc32_t	The output rotor flux estimated electric position (angle) - a 32-bit accumulator is normalized to the range <-1 ; 1) that represents an angle (in radians) within the range <-π ; π).
sCtrl	fltCompAlphaInteg_1	float_t	The state variable in the alpha part of the controller; integral part at step k-1.
	fltCompBetaInteg_1	float_t	The state variable in the beta part of the controller; integral part at step k-1.
	fltCompAlphaErr_1	float_t	The state variable in the alpha part of the controller; error part at step k-1.
	fltCompBetaErr_1	float_t	The state variable in the beta part of the controller; error part at step k-1.
	fltPGain	float_t	The proportional gain Kp for the stator model PI correction. Set by the user.
	fltIGain	float_t	The integration gain Ki for the stator model PI correction. Set by the user.
fltPsiRA1Gain		float_t	The gain is defined as: $\frac{\tau_r}{\tau_r + T_s}$ where: $\tau_r = \frac{L_r}{R_r}$

Table continues on the next page...

Variable name	Data type	Description
		The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
fltPsiRB1Gain	float_t	The coefficient gain is defined as: $\frac{L_m \cdot T_s}{\tau_r + T_s}$ where: $\tau_r = \frac{L_r}{R_r}$ The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
fltPsiSA1Gain	float_t	The gain is defined as: $\frac{1}{1 + T_s \cdot 2\pi \cdot f_{\text{integ}}}$ The f_{integ} is a cut-off frequency of a low-pass filter approximation of a pure integrator. The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
fltPsiSA2Gain	float_t	The coefficient gain is defined as: $\frac{L_s}{1 + T_s \cdot 2\pi \cdot f_{\text{integ}}} \cdot T_s$ The f_{integ} is a cut-off frequency of a low-pass filter approximation of a pure integrator. The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
fltKrInvGain	float_t	The gain is defined as: $\frac{L_r}{L_m}$ The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
fltKrLsTotLeakGain	float_t	The coefficient gain is defined as: $\frac{L_s \cdot L_r - L_m^2}{L_m}$ The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
fltRsEst	float_t	The stator resistance parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
fltTorqueGain	float_t	The torque constant coefficient gain is defined as: $\frac{3 \cdot P_P \cdot L_m}{2 \cdot L_r}$ The P_P is a number of motor pole-pairs. The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.

2.1.3 Declaration

The available AMCLIB_ACIMRotFluxObsrvInit function has the following declaration:

```
void AMCLIB_ACIMRotFluxObsrvInit_FLT(AMCLIB_ACIM_ROT_FLUX_OBSRV_T_FLT *psCtrl)
```

The available AMCLIB_ACIMRotFluxObsrv_FLT function has the following declaration:

```
void AMCLIB_ACIMRotFluxObsrv_FLT(const GMCLIB_2COOR_ALBE_T_FLT *psISAlBe, const
GMCLIB_2COOR_ALBE_T_FLT *psUSAlBe, AMCLIB_ACIM_ROT_FLUX_OBSRV_T_FLT *psCtrl)
```

2.1.4 Function use

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

```
#include "amclib.h"

static GMCLIB_2COOR_ALBE_T_FLT sIsAlBe, sUsAlBe, sPsiRAlBe;
static AMCLIB_ACIM_ROT_FLUX_OBSRV_T_FLT sRfoParam;

void Isr(void);

void main (void)
{
    sRfoParam.sCtrl.fltPGain     = 32750.0F;
    sRfoParam.sCtrl.fltIGain     = 12500.0F;
    sRfoParam.fltKrInvGain       = 1.0851063829787235F;
    sRfoParam.fltKrLsTotLeakGain = 0.08340425531914897F;
    sRfoParam.fltPsiRA1Gain      = 0.995151077592515F;
    sRfoParam.fltPsiRB1Gain      = 0.002278993531517996F;
    sRfoParam.fltPsiSA1Gain      = 0.9981185907806752F;
    sRfoParam.fltPsiSA2Gain      = 0.0000998118590780675 2F;
    sRfoParam.fltRsEst           = 26.06F;

    /* Initialization of the RFO's structure */
    AMCLIB_ACIMRotFluxObsrvInit_FLT (&sRfoParam);

    sIsAlBe.fltAlpha = 0.05F;
    sIsAlBe.fltBeta  = 0.1F;
    sUsAlBe.fltAlpha = 0.2F;
    sUsAlBe.fltBeta  = -0.1F;
}

/* Periodical function or interrupt */
void Isr(void)
{
    /* Rotor flux observer calculation */
    AMCLIB_ACIMRotFluxObsrv_FLT(&sIsAlBe, &sUsAlBe, &sRfoParam);
}
```

2.2 AMCLIB_ACIMSpeedMRAS

The AMCLIB_ACIMSpeedMRAS function is based on the model reference approach (MRAS), and it uses the redundancy of two machine models of different structures that estimate the same state variable based on different sets of input variables. It means that the rotor speed can obtained using an estimator with MRAS principle, in which the error vector is formed from the outputs of two models (both dependent on different motor parameters) - as shown in Figure 2-2.

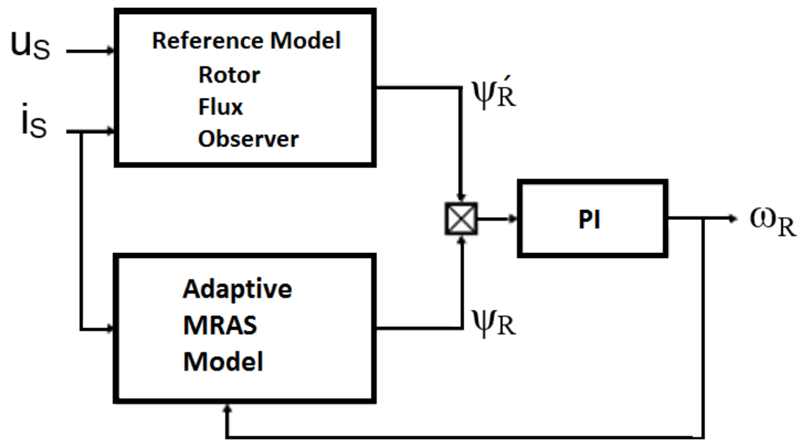

Figure 2-2. The estimated and real rotor *dq* synchronous reference frames

The closed-loop flux observer provides a stationary-axis-based rotor flux Ψ_R from RFO as a reference for the MRAS model, whereas the adaptive model of MRAS is the current-mode flux observer, which provides adjustable stationary-axis-based rotor flux:

$$\frac{d\overrightarrow{\psi_r^{MRAS}}}{dt} = -\frac{1}{\tau_r}\cdot\overrightarrow{\psi_r^{MRAS}} + j\omega_r\cdot\overrightarrow{\psi_r^{MRAS}} + \frac{L_m}{\tau_r}\vec{i_s}$$

Equation 7

where:

- i_s is the stator current vector
- Ψ_r is the rotor flux-linkage vector
- ω_r is the rotor electrical angular speed
- τ_r is the rotor electrical time constant
- L_m is the mutual equivalent inductance

The phase angle between the two estimated rotor flux vectors is used to correct the adaptive model, according to:

$$e_{MRAS} = \overrightarrow{\psi_{r\alpha}^{RFO}}\cdot\overrightarrow{\psi_{r\beta}^{MRAS}} - \overrightarrow{\psi_{r\beta}^{RFO}}\cdot\overrightarrow{\psi_{r\alpha}^{MRAS}}$$

Equation 8

The estimated speed ω_R is adjusted by a PI regulator.

2.2.1 Available versions

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

Table 2-3. Init version

<table>
<tr><th>Function name</th><th>Parameters</th><th>Result type</th></tr>
<tr><td rowspan="2">AMCLIB_ACIMSpeedMRASInit_FLT</td><td>AMCLIB_ACIMSpeedMRAS_T_FLT *</td><td>void</td></tr>
<tr><td colspan="2">The initialization does not have an input.</td></tr>
</table>

Table 2-4. Function version

<table>
<tr><th>Function name</th><th colspan="2">Input/output type</th><th>Result type</th></tr>
<tr><td rowspan="5">AMCLIB_ACIMSpeedMRAS_FLT</td><td rowspan="3">Input</td><td>GMCLIB_2COOR_ALBE_T_FLT *</td><td rowspan="4">void</td></tr>
<tr><td>GMCLIB_2COOR_ALBE_T_FLT *</td></tr>
<tr><td>acc32_t</td></tr>
<tr><td>Parameters</td><td>AMCLIB_ACIMSpeedMRAS_T_FLT *</td></tr>
<tr><td colspan="3">The AMCLIB_ACIMSpeedMRAS_FLT function with a 32-bit single precision floating-point inputs: stator current and voltage in alpha-beta coordinates, and a 32-bit accumulator input rotor position (lower fraction 15 bits = position, higher integer bits are not used). All are within the full range.</td></tr>
</table>

2.2.2 AMCLIB_ACIMSpeedMRAS_T_FLT type description

<table>
<tr><th colspan="2">Variable name</th><th>Data type</th><th>Description</th></tr>
<tr><td colspan="2">fltSpeedIIR1Param</td><td>GDFLIB_FILTER_IIR1_T_FLT</td><td>The IIR1 filter structure for estimated speed filtration. Set by the user.</td></tr>
<tr><td colspan="2">sPsiRotRDQ</td><td>GMCLIB_2COOR_DQ_T_FLT</td><td>The output rotor flux estimated structure from the current model. The structure consists of the D and Q rotor flux components stored for the next step.</td></tr>
<tr><td colspan="2">fltSpeed</td><td>float_t</td><td>The output rotor estimated electrical speed.</td></tr>
<tr><td colspan="2">fltSpeedElIIR1</td><td>float_t</td><td>The output rotor estimated electrical speed filtered.</td></tr>
<tr><td colspan="2">fltSpeedMeIIR1</td><td>float_t</td><td>The output rotor estimated mechanical speed filtered.</td></tr>
<tr><td colspan="2">a32RotPos</td><td>acc32_t</td><td>The output rotor estimated electric position (angle) - a 32-bit accumulator is normalized to the range <-1 ; 1) that represents an angle (in radians) within the range <-π ; π).</td></tr>
<tr><td rowspan="4">sCtrl</td><td>fltSpeedInteg_1</td><td>float_t</td><td>The speed integral part - state variable at step k-1 of the electrical speed controller.</td></tr>
<tr><td>fltSpeedErr_1</td><td>float_t</td><td>The speed error - state variable at step k-1 of the electrical speed controller.</td></tr>
<tr><td>fltPGain</td><td>float_t</td><td>The MRAS proportional gain coefficient. Set by the user.</td></tr>
<tr><td>fltIGain</td><td>float_t</td><td>The MRAS integral gain coefficient. Set by the user.</td></tr>
</table>

Table continues on the next page...

Variable name	Data type	Description
fltPsiRA1Gain	float_t	The coefficient gain is defined as: $\frac{\tau_r}{\tau_r + T_s}$ where: $\tau_r = \frac{L_r}{R_r}$ The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
fltPsiRB1Gain	float_t	The coefficient gain is defined as: $\frac{L_m \cdot T_s}{\tau_r + T_s}$ where: $\tau_r = \frac{L_r}{R_r}$ The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
fltTs	float_t	The sample time constant - the time between the steps. Set by the user.
fltSpeedMeGain	float_t	The speed gain coefficient, defined as: $\frac{60}{2\pi \cdot P_P}$ Where P_P is the number of motor pole-pairs. The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.

2.2.3 Declaration

The available AMCLIB_ACIMSpeedMRASInit functions have the following declarations:

```
void AMCLIB_ACIMSpeedMRASInit_FLT(AMCLIB_ACIM_SPEED_MRAS_T *psCtrl)
```

The available AMCLIB_ACIMSpeedMRAS functions have the following declarations:

```
void AMCLIB_ACIMSpeedMRAS_FLT(const GMCLIB_2COOR_ALBE_T_FLT *psISAlBe, const
GMCLIB_2COOR_ALBE_T_FLT *psPsiRAlBe, acc32_t a32RotPos, AMCLIB_ACIM_SPEED_MRAS_T *psCtrl)
```

2.2.4 Function use

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

```
#include "amclib.h"

static GMCLIB_2COOR_ALBE_T_FLT sIsAlBe, sPsiRAlBe;
static AMCLIB_ACIM_SPEED_MRAS_T sMrasParam;
static acc32_t a32RotPosIn;

void Isr(void);
```

```
void main (void)
{
    sMrasParam.sCtrl.fltIGain  = 12500.0F;
    sMrasParam.sCtrl.fltPGain  = 32750.0F;
    sMrasParam.fltSpeedIIR1Param.sFltCoeff.fltB0 = 0.00313F;
    sMrasParam.fltSpeedIIR1Param.sFltCoeff.fltB1 = 0.00313F;
    sMrasParam.fltSpeedIIR1Param.sFltCoeff.fltA1 = 0.99373F;
    sMrasParam.fltPsiRA1Gain   = 0.995151077592515F;
    sMrasParam.fltPsiRB1Gain   = 0.002278993531517996F;
    sMrasParam.fltTs           = 0.0001F;

    /* Initialization of the MRAS's structure */
    AMCLIB_ACIMSpeedMRASInit_FLT (&sMrasParam);

    sIsAlBe.fltAlpha   = 0.05F;
    sIsAlBe.fltBeta    = 0.1F;
    sPsiRAlBe.fltAlpha = 0.2F;
    sPsiRAlBe.fltBeta  = -0.1F;
}

/* Periodical function or interrupt */
void Isr(void)
{
    /* Speed estimation calculation based on MRAS */
    AMCLIB_ACIMSpeedMRAS_FLT(&sIsAlBe, &sPsiRAlBe, a32RotPosIn, &sMrasParam);
}
```

2.3 AMCLIB_AngleTrackObsrv

The AMCLIB_TrackObsrv function calculates an angle-tracking observer for determination of angular speed and position of the input signal. It requires two input arguments as sine and cosine samples. The practical implementation of the angle-tracking observer algorithm is described below.

The angle-tracking observer compares values of the input signals - sin(θ), cos(θ) with their corresponding estimations. As in any common closed-loop systems, the intent is to minimize the observer error towards zero value. The observer error is given here by subtracting the estimated resolver rotor angle from the actual rotor angle.

The tracking-observer algorithm uses the phase-locked loop mechanism. It is recommended to call this function at every sampling period. It requires a single input argument as phase error. A phase-tracking observer with standard PI controller used as the loop compensator is shown in Figure 2-3.

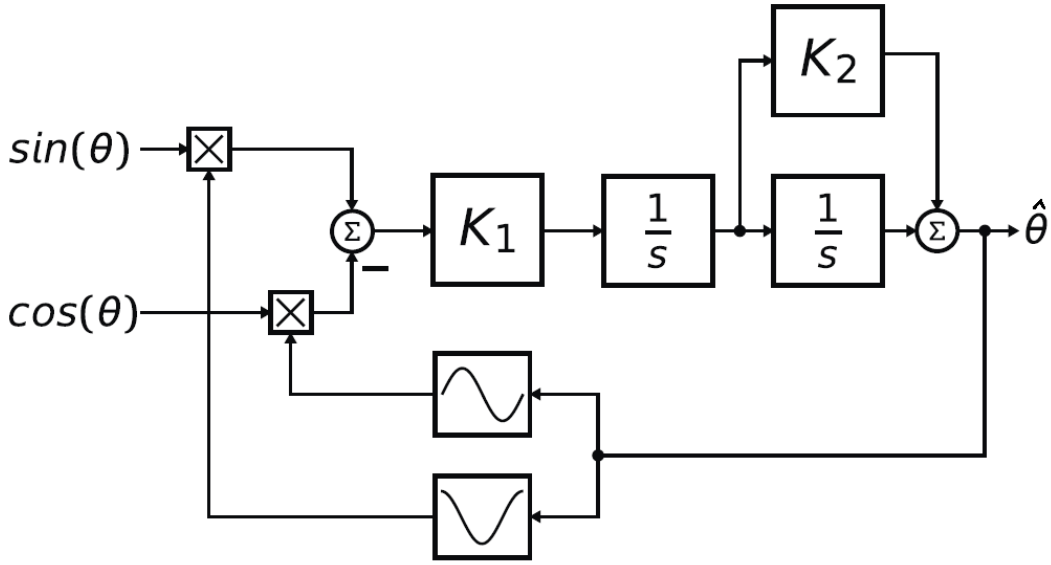

Figure 2-3. Block diagram of proposed PLL scheme for position estimation

Note that the mathematical expression of the observer error is known as the formula of the difference between two angles:

$$\sin(\theta - \hat{\theta}) = \sin(\theta) \cdot \cos(\hat{\theta}) - \cos(\theta) \cdot \sin(\hat{\theta})$$

Equation 9

If the deviation between the estimated and the actual angle is very small, then the observer error may be expressed using the following equation:

$$\sin(\theta - \hat{\theta}) \approx \theta - \hat{\theta}$$

Equation 10

The primary benefit of the angle-tracking observer utilization, in comparison with the trigonometric method, is its smoothing capability. This filtering is achieved by the integrator and the proportional and integral controllers, which are connected in series and closed by a unit feedback loop. This block diagram tracks the actual rotor angle and speed, and continuously updates their estimations. The angle-tracking observer transfer function is expressed as follows:

$$\frac{\hat{\theta}(s)}{\theta(s)} = \frac{K_1(1 + sK_2)}{s^2 + sK_1K_2 + K_1}$$

Equation 11

The characteristic polynomial of the angle-tracking observer corresponds to the denominator of the following transfer function:

$s^2 + sK_1K_2 + K_1$

Appropriate dynamic behavior of the angle-tracking observer is achieved by the placement of the poles of characteristic polynomial. This general method is based on matching the coefficients of characteristic polynomial with the coefficients of a general second-order system.

The analog integrators in the previous figure (marked as 1 / s) are replaced by an equivalent of the discrete-time integrator using the backward Euler integration method. The discrete-time block diagram of the angle-tracking observer is shown in the following figure:

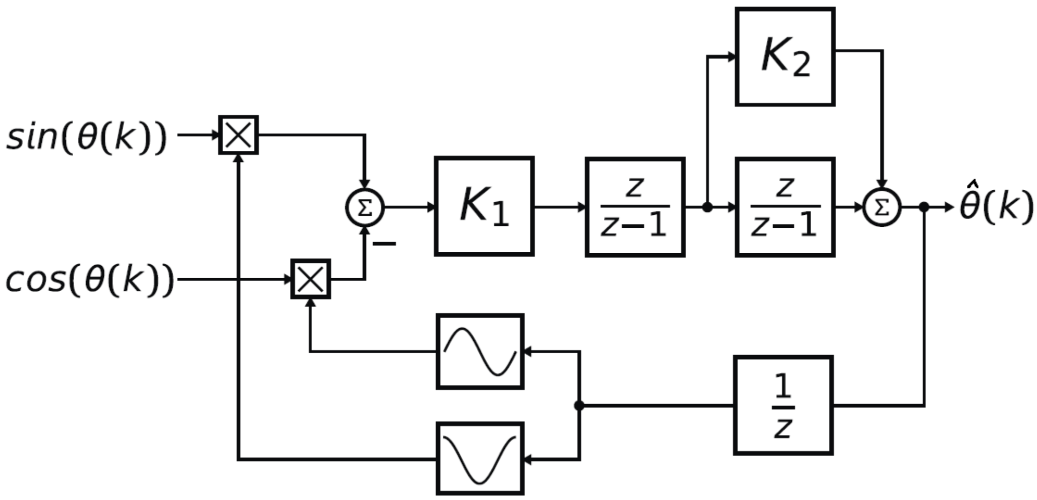

Figure 2-4. Block scheme of discrete-time tracking observer

The essential equations for implementating the angle-tracking observer (according to this block scheme) are as follows:

$$e(k) = \sin(\theta(k)) \cdot \cos(\hat{\theta}(k-1)) - \cos(\theta(k)) \cdot \sin(\hat{\theta}(k-1))$$

Equation 12

$$\omega(k) = T_s \bullet K_1 \bullet e(k) + \omega(k-1)$$

Equation 13

$$a_2(k) = T_s \bullet \omega(k) + a_2(k-1)$$

Equation 14

$$\hat{\theta}(k) = K_2 \cdot \omega(k) + a_2(k)$$

Equation 15

where:

- K_1 is the integral gain of the I controller
- K_2 is the proportional gain of the PI controller

- T_s is the sampling period [s]
- e(k) is the position error in step k
- ω(k) is the rotor speed [rad / s] in step k
- ω(k - 1) is the rotor speed [rad / s] in step k - 1
- a(k) is the integral output of the PI controler [rad / s] in step k
- a(k - 1) is the integral output of the PI controler [rad / s] in step k - 1
- θ(k) is the rotor angle [rad] in step k
- θ(k - 1) is the rotor angle [rad] in step k - 1
- θ(k) is the estimated rotor angle [rad] in step k
- θ(k - 1) is the estimated rotor angle [rad] in step k - 1

In the fractional arithmetic, Equation 12 on page 42 to Equation 15 on page 42 are as follows:

$$\omega_{sc}(k) \bullet \omega_{max} = T_s \bullet K_1 \bullet e(k) + \omega_{sc}(k-1) \bullet \omega_{max}$$

Equation 16

$$a_{2sc}(k) \bullet \theta_{max} = T_s \bullet \omega_{sc}(k) \bullet \omega_{max} + a_{2sc}(k-1) \bullet \theta_{max}$$

Equation 17

$$\hat{\theta}_{sc}(k) \bullet \theta_{max} = K_2 \bullet \omega_{sc}(k) \bullet \omega_{max} + a_{2sc}(k) \bullet \theta_{max}$$

Equation 18

where:

- $e_{sc}(k)$ is the scaled position error in step k
- $\omega_{sc}(k)$ is the scaled rotor speed [rad / s] in step k
- $\omega_{sc}(k - 1)$ is the scaled rotor speed [rad / s] in step k - 1
- $a_{sc}(k)$ is the integral output of the PI controler [rad / s] in step k
- $a_{sc}(k - 1)$ is the integral output of the PI controler [rad / s] in step k - 1
- $\theta_{sc}(k)$ is the scaled rotor angle [rad] in step k
- $\theta_{sc}(k - 1)$ is the scaled rotor angle [rad] in step k - 1
- $\theta_{sc}(k)$ is the scaled rotor angle [rad] in step k
- $\theta_{sc}(k - 1)$ is the scaled rotor angle [rad] in step k - 1
- ω_{max} is the maximum speed
- θ_{max} is the maximum rotor angle (typicaly π)

2.3.1 Available versions

The function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1).
- Accumulator output with floating point inputs - the output is the accumulator type, where the inputs for the calculation are the floating-point types within the range <-1.0 ; 1.0>.

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

Table 2-5. Init versions

Function name	Init angle	Parameters	Result type
AMCLIB_AngleTrackObsrvInit_F16	frac16_t	AMCLIB_ANGLE_TRACK_OBSRV_T_F32 *	void
	The input is a 16-bit fractional value of the angle normalized to the range <-1 ; 1) that represents an angle in (radians) within the range <-π ; π).		
AMCLIB_AngleTrackObsrvInit_A32af	acc32_t	AMCLIB_ANGLE_TRACK_OBSRV_T_FLT *	void
	The input is a 32-bit accumulator value of the angle divided by π.		

Table 2-6. Function versions

Function name	Input type	Parameters	Result type
AMCLIB_AngleTrackObsrv_F16	GMCLIB_2COOR_SINCOS_T_F16 *	AMCLIB_ANGLE_TRACK_OBSRV_T_F32 *	frac16_t
	Angle-tracking observer with a two-componenent (sin/cos) 16-bit fractional position input within the range <-1 ; 1). The output from the obsever is a 16-bit fractional position normalized to the range <-1 ; 1) that represents an angle (in radians) within the range <-π ; π).		
AMCLIB_AngleTrackObsrv_A32ff	GMCLIB_2COOR_SINCOS_T_FLT *	AMCLIB_ANGLE_TRACK_OBSRV_T_FLT *	acc32_t
	Tracking observer with a a two-componenent (sin/cos) 32-bit accumulator position input within the range <-1.0 ; 1.0>. The output from the obsever is a 32-bit accumulator position normalized to the range <-1 ; 1) that represents an angle (in radians) within the range <-π ; π).		

2.3.2 AMCLIB_ANGLE_TRACK_OBSRV_T_F32

Variable name	Input type	Description
f32Speed	frac32_t	Estimated speed as the output of the first numerical integrator. The parameter is within the range <-1 ; 1). Controlled by the AMCLIB_AngleTrackObsrv_F16 algorithm; cleared by the AMCLIB_AngleTrackObsrvInit_F16 function.

Table continues on the next page...

Variable name	Input type	Description
f32A2	frac32_t	Output of the second numerical integrator. The parameter is within the range <-1 ; 1). Controlled by the AMCLIB_AngleTrackObsrv_F16 and AMCLIB_AngleTrackObsrvInit_F16 algorithms.
f16Theta	frac16_t	Estimated position as the output of the observer. The parameter is normalized to the range <-1 ; 1) that represents an angle (in radians) within the range <-π ; π). Controlled by the AMCLIB_AngleTrackObsrv_F16 and AMCLIB_AngleTrackObsrvInit_F16 algorithms.
f16SinEstim	frac16_t	Sine of the estimated position as the output of the actual step. Keeps the sine of the position for the next step. The parameter is within the range <-1 ; 1). Controlled by the AMCLIB_AngleTrackObsrv_F16 and AMCLIB_AngleTrackObsrvInit_F16 algorithms.
f16CosEstim	frac16_t	Cosine of the estimated position as the output of the actual step. Keeps the cosine of the position for the next step. The parameter is within the range <-1 ; 1). Controlled by the AMCLIB_AngleTrackObsrv_F16 and AMCLIB_AngleTrackObsrvInit_F16 algorithms.
f16K1Gain	frac16_t	Observer K1 gain is set up according to Equation 16 on page 43 as: $T_s \cdot K_1 \cdot \frac{1}{\omega_{max}} \cdot 2^{-K1sh}$ The parameter is a 16-bit fractional type within the range <0 ; 1). Set by the user.
i16K1GainSh	int16_t	Observer K2 gain shift takes care of keeping the f16K1Gain variable within the fractional range <-1 ; 1). The shift is determined as: $\log_2(T_s \cdot K_1 \cdot \frac{1}{\omega_{max}}) - \log_2 1 < K1sh \leq \log_2(T_s \cdot K_1 \cdot \frac{1}{\omega_{max}}) - \log_2 0.5$ The parameter is a 16-bit integer type within the range <-15 ; 15>. Set by the user.
f16K2Gain	frac16_t	Observer K2 gain is set up according to Equation 18 on page 43 as: $K_2 \cdot \frac{\omega_{max}}{\theta_{max}} \cdot 2^{-K2sh}$ The parameter is a 16-bit fractional type within the range <0 ; 1). Set by the user.
i16K2GainSh	int16_t	Observer K2 gain shift takes care of keeping the f16K2Gain variable within the fractional range <-1 ; 1). The shift is determined as: $\log_2(K_2 \cdot \frac{\omega_{max}}{\theta_{max}}) - \log_2 1 < K2sh \leq \log_2(K_2 \cdot \frac{\omega_{max}}{\theta_{max}}) - \log_2 0.5$ The parameter is a 16-bit integer type within the range <-15 ; 15>. Set by the user.
f16A2Gain	frac16_t	Observer A2 gain for the output position is set up according to Equation 17 on page 43 as: $T_s \cdot \frac{\omega_{max}}{\theta_{max}} \cdot 2^{-A2sh}$ The parameter is a 16-bit fractional type within the range <0 ; 1). Set by the user.
i16A2GainSh	int16_t	Observer A2 gain shift for the position integrator takes care of keeping the f16A2Gain variable within the fractional range <-1 ; 1). The shift is determined as: $\log_2(T_s \cdot \frac{\omega_{max}}{\theta_{max}}) - \log_2 1 < A2sh \leq \log_2(T_s \cdot \frac{\omega_{max}}{\theta_{max}}) - \log_2 0.5$ The parameter is a 16-bit integer type within the range <-15 ; 15>. Set by the user.

2.3.3 AMCLIB_ANGLE_TRACK_OBSRV_T_FLT

Variable name	Input type	Description
fltSpeed	float_t	Estimated speed as the output of the first numerical integrator. The parameter is within the full range. Controlled by the AMCLIB_AngleTrackObsrv_A32ff algorithm; cleared by AMCLIB_AngleTrackObsrvInit_A32af function.
f32A2	frac32_t	Output of the second numerical integrator. The parameter is within the range <-1 ; 1). Controlled by the AMCLIB_AngleTrackObsrv_A32ff and AMCLIB_AngleTrackObsrvInit_A32af algorithms.
a32Theta	acc32_t	Estimated position as the output of the observer. The parameter is normalized to the range <-1 ; 1) that represents an angle (in radians) within the range <-π ; π). Controlled by the AMCLIB_AngleTrackObsrv_A32ff and AMCLIB_AngleTrackObsrvInit_A32af algorithms.
fltSinEstim	float_t	Sine of the estimated position as the output of the actual step. Keeps the sine of the position for the next step. The parameter is within the range <-1 ; 1>. Controlled by the AMCLIB_AngleTrackObsrv_A32ff and AMCLIB_AngleTrackObsrvInit_A32af algorithms.
fltCosEstim	float_t	Cosine of the estimated position as the output of the actual step. Keeps the cosine of the position for the next step. The parameter is within the range <-1 ; 1>. Controlled by the AMCLIB_AngleTrackObsrv_A32ff and AMCLIB_AngleTrackObsrvInit_A32af algorithms.
fltK1Gain	float_t	Observer K1 gain is set up according to Equation 13 on page 42 as: K_1T_s. The parameter is a 32-bit single precision floating-point non-negative value. Set by the user.
fltK2Gain	float_t	Observer K2 gain is set up according to Equation 15 on page 42 as: K_2. The parameter is a 32-bit single precision floating-point non-negative value. Set by the user.
fltA2Gain	float_t	Observer A2 gain for the output position is set up according to Equation 14 on page 42 as: T_s. The parameter is a 32-bit single precision floating-point non-negative value. Set by the user.

2.3.4 Declaration

The available AMCLIB_AngleTrackObsrvInit functions have the following declarations:

```
void AMCLIB_AngleTrackObsrvInit_F16(frac16_t f16ThetaInit, AMCLIB_ANGLE_TRACK_OBSRV_T_F32
*psCtrl)
void AMCLIB_AngleTrackObsrvInit_A32ff(acc32_t a32ThetaInit, AMCLIB_ANGLE_TRACK_OBSRV_T_FLT
*psCtrl)
```

The available AMCLIB_AngleTrackObsrv functions have the following declarations:

```
frac16_t AMCLIB_AngleTrackObsrv_F16(const GMCLIB_2COOR_SINCOS_T_F16 *psAnglePos,
AMCLIB_ANGLE_TRACK_OBSRV_T_F32 *psCtrl)
acc32_t AMCLIB_AngleTrackObsrv_A32ff(const GMCLIB_2COOR_SINCOS_T_FLT *psAnglePos,
AMCLIB_ANGLE_TRACK_OBSRV_T_FLT *psCtrl)
```

2.3.5 Function use

The use of the AMCLIB_AngleTrackObsrvInit and AMCLIB_AngleTrackObsrv functions is shown in the following example:

```
#include "amclib.h"

static AMCLIB_ANGLE_TRACK_OBSRV_T_F32  sAto;
static GMCLIB_2COOR_SINCOS_T_F16 sAnglePos;
static frac16_t       f16PositionEstim, f16PositionInit;

void Isr(void);

void main(void)
{
  sAto.f16K1Gain     = FRAC16(0.6434);
  sAto.i16K1GainSh   = -9;
  sAto.f16K2Gain     = FRAC16(0.6801);
  sAto.i16K2GainSh   = -2;
  sAto.f16A2Gain     = FRAC16(0.6400);
  sAto.i16A2GainSh   = -4;

  f16PositionInit = FRAC16(0.0);

  AMCLIB_AngleTrackObsrvInit_F16(f16PositionInit, &sAto);

  sAnglePos.f16Sin   = FRAC16(0.0);
  sAnglePos.f16Cos   = FRAC16(1.0);
}

/* Periodical function or interrupt */
void Isr(void)
{
  /* Angle tracking observer calculation */
  f16PositionEstim = AMCLIB_AngleTrackObsrv_F16(&sAnglePos, &sAto);
}
```

2.4 AMCLIB_PMSMBemfObsrvAB

The AMCLIB_PMSMBemfObsrvAB function calculates the algorithm of the back-electro-motive force (back-EMF) observer in a stationary reference frame. The estimation method for the rotor position and angular speed is based on the mathematical model of an interior PMSM motor with extended electro-motive force function, which is realized in the alpha/beta stationary reference frame.

The back-EMF observer detects the generated motor voltages, induced by the permanent magnets. The angle-tracking observer uses the back-EMF signals to calculate the position and speed of the rotor. The transformed model is then derived as:

$$\begin{bmatrix} u_\alpha \\ u_\beta \end{bmatrix} = \begin{bmatrix} R_S + sL_D & \omega_r \Delta L \\ -\omega_r \Delta L & R_S + sL_D \end{bmatrix} \cdot \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} + \left[\Delta L \cdot \left(\omega_r i_D - s i_Q \right) + \Psi_m \omega_r \right] \cdot \begin{bmatrix} -\sin(\theta_r) \\ \cos(\theta_r) \end{bmatrix}$$

Equation 19

Where:

- R_S is the stator resistance
- L_D and L_Q are the D-axis and Q-axis inductances
- $\Delta L = L_D - L_Q$ is the motor saliency
- Ψ_m is the back-EMF constant
- ω_r is the angular electrical rotor speed
- u_α and u_β are the estimated stator voltages
- i_α and i_β are the estimated stator currents
- θ_r is the estimated rotor electrical position
- s is the operator of the derivative

This extended back-EMF model includes both the position information from the conventionally defined back-EMF and the stator inductance as well. This enables extracting the rotor position and velocity information by estimating the extended back-EMF only.

Both the alpha and beta axes consist of the stator current observer based on the RL motor circuit which requires motor parameters.

The current observer input is the sum of the actual applied motor voltage and the cross-coupled rotational term, which corresponds to the motor saliency (L_D - L_Q) and compensator corrective output. The observer provides the back-EMF signals as a disturbance because the back-EMF is not included in the observer model.

The block diagram of the observer in the estimated reference frame is shown in Figure 2-5. The observer compensator is substituted by a standard PI controller. As shown in Figure 2-5, the observer model and hence also the PI controller gains in both axes are identical to each other.

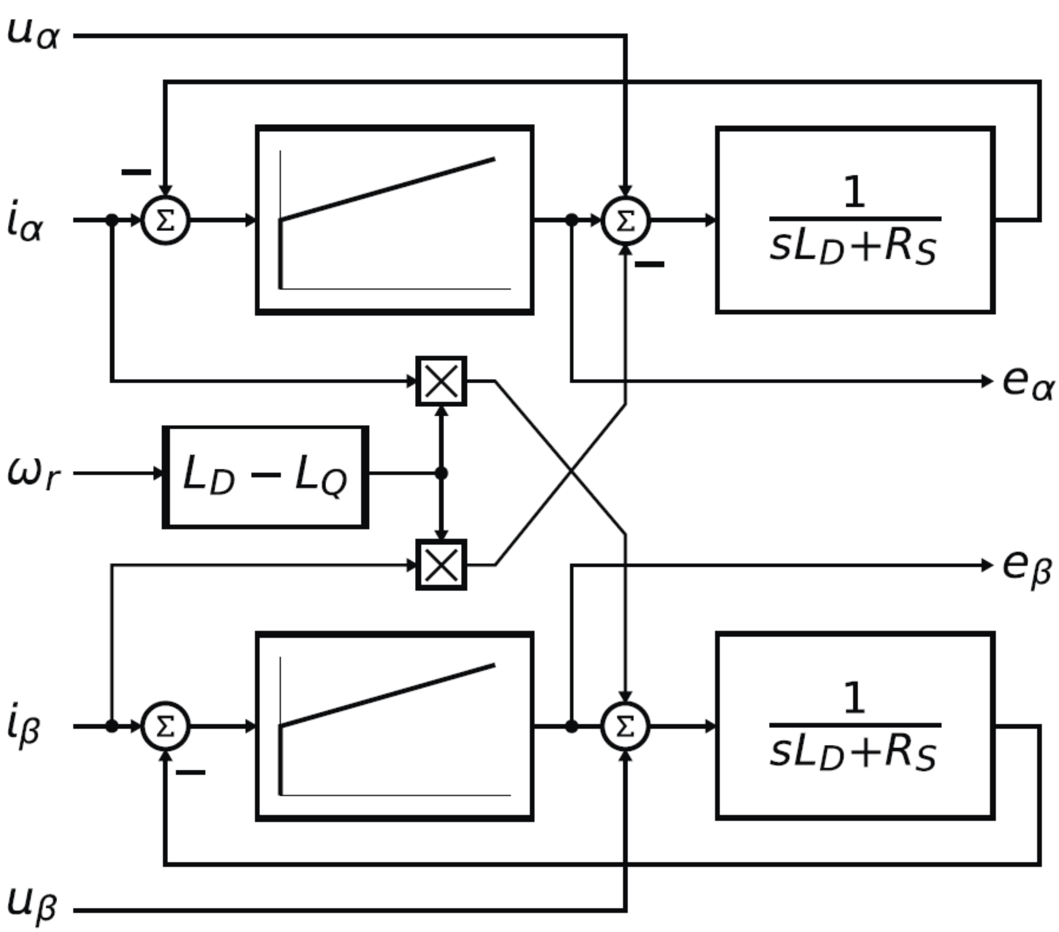

Figure 2-5. Block diagram of back-EMF observer

It is obvious that the accuracy of the back-EMF estimates is determined by the correctness of used motor parameters (R, L), by the fidelity of the reference stator voltage, and by the quality of compensator such as bandwidth, phase lag, and so on.

The appropriate dynamic behavior of the back-EMF observer is achieved by the placement of the poles of the stator current observer characteristic polynomial. This general method is based on matching the coefficients of the characteristic polynomial to the coefficients of the general second-order system.

$$\hat{E}_{\alpha\beta}(s) = -E_{\alpha\beta}(s) \cdot \frac{F_C(s)}{sL_D + R_S + F_C(s)}$$

Equation 20

The back-EMF observer is a Luenberger-type observer with a motor model, which is implemented using the backward Euler transformation as:

$$i(k) = \frac{T_s}{L_D + T_s R_S} \cdot u(k) + \frac{T_s}{L_D + T_s R_S} \cdot e(k) - \frac{\Delta L T_s}{L_D + T_s R_S} \cdot \omega_e(k) \cdot i'(k) + \frac{L_D}{L_D + T_s R_S} \cdot i(k-1)$$

Equation 21

Where:

- $i(k) = [i_\gamma, i_\delta]$ is the stator current vector in the actual step
- $i(k - 1) = [i_\gamma, i_\delta]$ is the stator current vector in the previous step
- $u(k) = [u_\gamma, u_\delta]$ is the stator voltage vector in the actual step
- $e(k) = [e_\gamma, e_\delta]$ is the stator back-EMF voltage vector in the actual step
- $i'(k) = [i_\gamma, -i_\delta]$ is the complementary stator current vector in the actual step
- $\omega_e(k)$ is the electrical angular speed in the actual step
- T_S is the sampling time [s]

This equation is transformed into the fractional arithmetic as:

$$i_{sc}(k)\bullet i_{max} = \frac{T_s}{L_D + T_s R_S}\bullet u_{sc}(k)\bullet u_{max} + \frac{T_s}{L_D + T_s R_S}\bullet e_{sc}(k)\bullet e_{max} - \frac{\Delta L T_s}{L_D + T_s R_S}\bullet \omega_{esc}(k)\bullet \omega_{max}\bullet i'_{sc}(k)\bullet i_{max} + \frac{L_D}{L_D + T_s R_S}\bullet i_{sc}(k-1)\bullet i_{max}$$

Equation 22

Where:

- $i_{sc}(k) = [i_\gamma, i_\delta]$ is the scaled stator current vector in the actual step
- $i_{sc}(k - 1) = [i_\gamma, i_\delta]$ is the scaled stator current vector in the previous step
- $u_{sc}(k) = [u_\gamma, u_\delta]$ is the scaled stator voltage vector in the actual step
- $e_{sc}(k) = [e_\gamma, e_\delta]$ is the scaled stator back-EMF voltage vector in the actual step
- $i'_{sc}(k) = [i_\gamma, -i_\delta]$ is the scaled complementary stator current vector in the actual step
- $\omega_{esc}(k)$ is the scaled electrical angular speed in the actual step
- i_{max} is the maximum current [A]
- e_{max} is the maximum back-EMF voltage [V]
- u_{max} is the maximum stator voltage [V]
- ω_{max} is the maximum electrical angular speed in [rad / s]

If the Luenberger-type stator current observer is properly designed in the stationary reference frame, the back-EMF can be estimated as a disturbance produced by the observer controller. However, this is only valid when the back-EMF term is not included in the observer model. The observer is a closed-loop current observer, therefore, it acts as a state filter for the back-EMF term.

The estimate of the extended EMF term can be derived from Equation 20 on page 49 as:

$$-\frac{\hat{E}_{\gamma\delta}(s)}{E_{\gamma\delta}(s)} = \frac{sK_P + K_I}{s^2 L_D + sR_S + sK_P + K_I}$$

Equation 23

The observer controller can be designed by comparing the closed-loop characteristic polynomial to that of a standard second-order system as:

$$s^2 + \frac{K_P + R_S}{L_D} \bullet s + \frac{K_I}{L_D} = s^2 + 2\xi\omega_0 s + \omega_0^2$$

Equation 24

where:

- ω_0 is the natural frequency of the closed-loop system (loop bandwith)
- ξ is the loop attenuation
- K_P is the proporional gain
- K_I is the integral gain

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 parameters use the accumulator types.
- Floating-point output - the output is the floating-point result within the type's full range.

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

Table 2-7. Init versions

Function name	Parameters	Result type
AMCLIB_PMSMBemfObsrvABInit_F16	AMCLIB_BEMF_OBSRV_AB_T_A32 *	void
	Initialization does not have any input.	
AMCLIB_PMSMBemfObsrvABInit_A32fff	AMCLIB_BEMF_OBSRV_AB_T_FLT *	void
	Initialization does not have any input.	

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

Table 2-8. Function versions

Function name	Input/output type		Result type
AMCLIB_PMSMBemfObsrvAB_F16	**Input**	GMCLIB_2COOR_ALBE_T_F16 *	void
		GMCLIB_2COOR_ALBE_T_F16 *	
		frac16_t	
	Parameters	AMCLIB_BEMF_OBSRV_AB_T_A32 *	
	The back-EMF observer with a 16-bit fractional input Alpha/Beta current and voltage, and a 16-bit electrical speed. All are within the range <-1 ; 1).		

Table continues on the next page...

Table 2-8. Function versions (continued)

Function name	Input/output type		Result type
AMCLIB_PMSMBemfObsrvAB_FLT	**Input**	GMCLIB_2COOR_ALBE_T_FLT *	void
		GMCLIB_2COOR_ALBE_T_FLT *	
		float_t	
	Parameters	AMCLIB_BEMF_OBSRV_AB_T_FLT *	
	The back-EMF observer with a 32-bit single precision floating-point input Alpha/Beta current and voltage, and a 32-bit single precision floating-point electrical speed. All are within the full range.		

2.4.2 AMCLIB_BEMF_OBSRV_AB_T_A32 type description

Variable name		Data type	Description
sEObsrv		GMCLIB_2COOR_ALBE_T_F32	The estimated back-EMF voltage structure.
sIObsrv		GMCLIB_2COOR_ALBE_T_F32	The estimated current structure.
sCtrl	f32IAlpha_1	frac32_t	The state variable in the alpha part of the observer, integral part at step k-1. The variable is within the range <-1 ; 1).
	f32IBeta_1	frac32_t	The state variable in the beta part of the observer, integral part at step k-1. The variable is within the range <-1 ; 1).
	a32PGain	acc32_t	The observer proportional gain is set up according to Equation 24 on page 51 as: $2\xi\omega_0 L_D - R_S$ The parameter is within the range <0 ; 65536.0). Set by the user.
	a32IGain	acc32_t	The observer integral gain is set up according to Equation 24 on page 51 as: $\omega_0^2 L_D$ The parameter is within the range <0 ; 65536.0). Set by the user.
a32IGain		acc32_t	The current coefficient gain is set up according to Equation 5 as: $\frac{L_D}{L_D + T_s R_S}$ The parameter is within the range <0 ; 65536.0). Set by the user.
a32UGain		acc32_t	The voltage coefficient gain is set up according to Equation 5 as: $\frac{T_s}{L_D + T_s R_S} \cdot \frac{u_{max}}{i_{max}}$

Table continues on the next page...

Variable name	Data type	Description
		The parameter is within the range <0 ; 65536.0). Set by the user.
a32WIGain	acc32_t	The angular speed coefficient gain is set up according to Equation 5 as: $\frac{\Delta L T_s}{L_D + T_s R_S} \cdot \omega_{max}$ The parameter is within the range <0 ; 65536.0).Set by the user.
a32EGain	acc32_t	The back-EMF coefficient gain is set up according to Equation 5 as: $\frac{T_s}{L_D + T_s R_S} \cdot \frac{e_{max}}{i_{max}}$ The parameter is within the range <0 ; 65536.0). Set by the user.
sUnityVctr	GMCLIB_2COOR_SINCOS_T_F16	The output - estimated angle as the sin/cos vector.

2.4.3 AMCLIB_BEMF_OBSRV_AB_T_FLT type description

Variable name		Data type	Description
sEObsrv		GMCLIB_2COOR_ALBE_T_FLT	The estimated back-EMF voltage structure.
sIObsrv		GMCLIB_2COOR_ALBE_T_FLT	The estimated current structure.
sCtrl	fltIAlpha_1	float_t	The state variable in the alpha part of the observer, integral part at step k-1. The variable is within the range <-1 ; 1).
	fltIBeta_1	float_t	The state variable in the beta part of the observer, integral part at step k-1. The variable is within the range <-1 ; 1).
	fltPGain	float_t	The observer proportional gain is set up according to Equation 24 on page 51 as: $2\xi\omega_0 L_D - R_S$ The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
	fltIGain	float_t	The observer integral gain is set up according to Equation 24 on page 51 as: $\omega_0^2 L_D$ The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
fltIGain		float_t	The current coefficient gain is set up according to Equation 4 as: $\frac{L_D}{L_D + T_s R_S}$

Table continues on the next page...

Variable name	Data type	Description
		The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
fltUGain	float_t	The voltage coefficient gain is set up according to Equation 4 as: $\frac{T_s}{L_D+T_sR_S}$ The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
fltWIGain	float_t	The angular speed coefficient gain is set up according to Equation 4 as: $\frac{\Delta L T_s}{L_D+T_sR_S}$ The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
fltEGain	float_t	The back-EMF coefficient gain is set up according to Equation 4 as: $\frac{T_s}{L_D+T_sR_S}$ The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
sUnityVctr	GMCLIB_2COOR_SINCOS_T_FLT	The output - estimated angle as the sin/cos vector.

2.4.4 Declaration

The available AMCLIB_PMSMBemfObsrvABInit functions have the following declarations:

```
void AMCLIB_PMSMBemfObsrvABInit_F16(AMCLIB_BEMF_OBSRV_AB_T_A32 *psCtrl)
void AMCLIB_PMSMBemfObsrvABInit_FLT(AMCLIB_BEMF_OBSRV_AB_T_FLT *psCtrl)
```

The available AMCLIB_PMSMBemfObsrvAB functions have the following declarations:

```
void AMCLIB_PMSMBemfObsrvAB_F16(const GMCLIB_2COOR_ALBE_T_F16 *psIAlBe, const
GMCLIB_2COOR_ALBE_T_F16 *psUAlBe, frac16_t f16Speed, AMCLIB_BEMF_OBSRV_AB_T_A32 *psCtrl)
void AMCLIB_PMSMBemfObsrvAB_FLT(const GMCLIB_2COOR_ALBE_T_FLT *psIAlBe, const
GMCLIB_2COOR_ALBE_T_FLT *psUAlBe, float_t fltSpeed, AMCLIB_BEMF_OBSRV_AB_T_FLT *psCtrl)
```

2.4.5 Function use

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

```
#include "amclib.h"

static GMCLIB_2COOR_ALBE_T_F16 sIAlBe, sUAlBe;
static AMCLIB_BEMF_OBSRV_AB_T_A32 sBemfObsrv;
static frac16_t f16Speed;

void Isr(void);

void main (void)
{
  sBemfObsrv.sCtrl.a32PGain= ACC32(1.697);
  sBemfObsrv.sCtrl.a32IGain= ACC32(0.134);
  sBemfObsrv.a32IGain = ACC32(0.986);
  sBemfObsrv.a32UGain = ACC32(0.170);
  sBemfObsrv.a32WIGain= ACC32(0.110);
  sBemfObsrv.a32EGain = ACC32(0.116);

  /* Initialization of the observer's structure */
  AMCLIB_PMSMBemfObsrvABInit_F16(&sBemfObsrv);

  sIAlBe.f16Alpha = FRAC16(0.05);
  sIAlBe.f16Beta  = FRAC16(0.1);
  sUAlBe.f16Alpha = FRAC16(0.2);
  sUAlBe.f16Beta  = FRAC16(-0.1);
}

/* Periodical function or interrupt */
void Isr(void)
{
  /* BEMF Observer calculation */
  AMCLIB_PMSMBemfObsrvAB_F16(&sIAlBe, &sUAlBe, f16Speed, &sBemfObsrv);
}
```

2.5 AMCLIB_PMSMBemfObsrvDQ

The AMCLIB_PMSMBemfObsrvDQ function calculates the algorithm of back-electro-motive force observer in a rotating reference frame. The method for estimating the rotor position and angular speed is based on the mathematical model of an interior PMSM motor with an extended electro-motive force function, which is realized in an estimated quasi-synchronous reference frame γ-δ as shown in Figure 2-6.

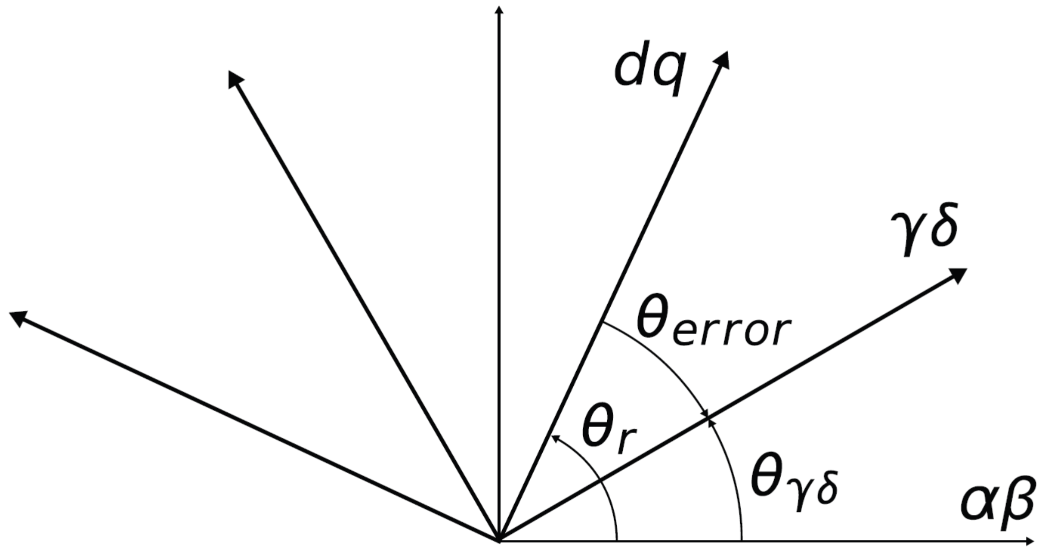

Figure 2-6. The estimated and real rotor *dq* synchronous reference frames

The back-EMF observer detects the generated motor voltages induced by the permanent magnets. A tracking observer uses the back-EMF signals to calculate the position and speed of the rotor. The transformed model is then derived as follows:

$$\begin{bmatrix} u_\gamma \\ u_\delta \end{bmatrix} = \begin{bmatrix} R_S + sL_D & -\omega_r L_Q \\ \omega_r L_Q & R_S + sL_D \end{bmatrix} \bullet \begin{bmatrix} i_\gamma \\ i_\delta \end{bmatrix} + \left(\Delta L \bullet \left(\omega_r i_D - s i_Q \right) + \Psi_m \omega_r \right) \bullet \begin{bmatrix} -\sin(\theta_{error}) \\ \cos(\theta_{error}) \end{bmatrix}$$

Equation 25

where:

- R_S is the stator resistance
- L_D and L_Q are the D-axis and Q-axis inductances
- Ψ_m is the back-EMF constant
- ω_r is the angular electrical rotor speed
- u_γ and u_δ are the estimated stator voltages
- i_γ and i_δ are the estimated stator currents
- θ_{error} is the error between the actual D-Q frame and the estimated frame position
- s is the operator of the derivative

The block diagram of the observer in the estimated reference frame is shown in Figure 2-7. The observer compensator is substituted by a standard PI controller. As shown in Figure 2-7, the observer model and hence also the PI controller gains in both axes are identical to each other.

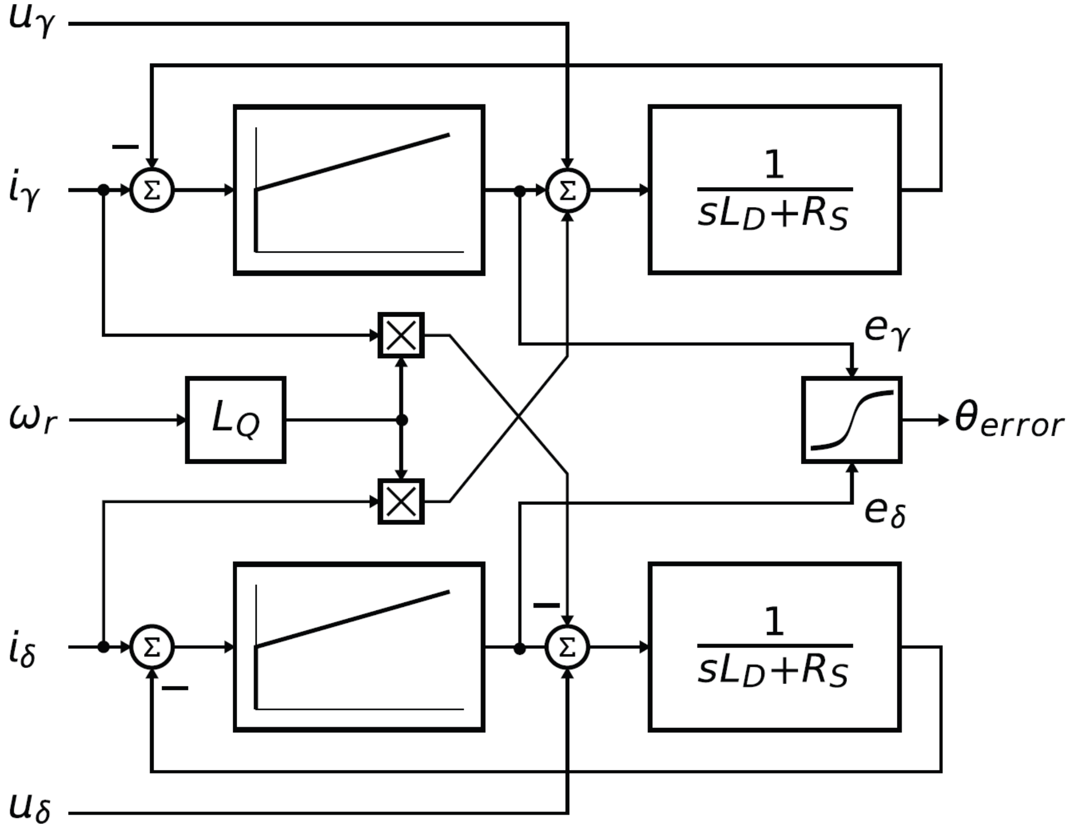

Figure 2-7. Block diagram of proposed Luenberger-type stator current observer acting as state filter for back-EMF

The position estimation can now be performed by extracting the θ_{error} term from the model, and adjusting the position of the estimated reference frame to achieve $\theta_{error} = 0$. Because the θ_{error} term is only included in the saliency-based EMF component of both u_γ and u_δ axis voltage equations, the Luenberger-based disturbance observer is designed to observe the u_γ and u_δ voltage components. The position displacement information θ_{error} is then obtained from the estimated back-EMFs as follows:

$$\theta_{error} = \operatorname{atan}\left(\frac{-e_\gamma}{e_\delta}\right)$$

Equation 26

The estimated position $\hat{\theta}_e$ can be obtained by driving the position of the estimated reference frame to achieve zero displacement $\theta_{error} = 0$. The phase-locked-loop mechanism can be adopted, where the loop compensator ensures correct tracking of the actual rotor flux position by keeping the error signal θ_{error} zeroed, $\theta_{error} = 0$.

A perfect match between the actual and estimated motor model parameters is assumed, and then the back-EMF transfer function can be simplified as follows:

$$\hat{E}_{\alpha\beta}(s) = -E_{\alpha\beta}(s) \cdot \frac{F_C(s)}{sL_D + R_S + F_C(s)}$$

Equation 27

The appropriate dynamic behavior of the back-EMF observer is achieved by the placement of the poles of the stator current observer characteristic polynomial. This general method is based on matching the coefficients of the characteristic polynomial with the coefficients of the general second-order system.

The back-EMF observer is a Luenberger-type observer with a motor model, which is implemented using the backward Euler transformation as follows:

$$i(k) = \frac{T_s}{L_D + T_s R_S} \cdot u(k) + \frac{T_s}{L_D + T_s R_S} \cdot e(k) + \frac{L_Q T_s}{L_D + T_s R_S} \cdot \omega_e(k) \cdot i'(k) + \frac{L_D}{L_D + T_s R_S} \cdot i(k-1)$$

Equation 28

where:

- $i(k) = [i_\gamma, i_\delta]$ is the stator current vector in the actual step
- $i(k - 1) = [i_\gamma, i_\delta]$ is the stator current vector in the previous step
- $u(k) = [u_\gamma, u_\delta]$ is the stator voltage vector in the actual step
- $e(k) = [e_\gamma, e_\delta]$ is the stator back-EMF voltage vector in the actual step
- $i'(k) = [i_\gamma, -i_\delta]$ is the complementary stator current vector in the actual step
- $\omega_e(k)$ is the electrical angular speed in the actual step
- T_S is the sampling time [s]

This equation is transformed into the fractional arithmetic as follows:

$$i_{sc}(k) \cdot i_{max} = \frac{T_s}{L_D + T_s R_S} \cdot u_{sc}(k) \cdot u_{max} + \frac{T_s}{L_D + T_s R_S} \cdot e_{sc}(k) \cdot e_{max} + \frac{L_Q T_s}{L_D + T_s R_S} \cdot \omega_{esc}(k) \cdot \omega_{max} \cdot i'_{sc}(k) \cdot i_{max} + \frac{L_D}{L_D + T_s R_S} \cdot i_{sc}(k-1) \cdot i_{max}$$

Equation 29

where:

- $i_{sc}(k) = [i_\gamma, i_\delta]$ is the scaled stator current vector in the actual step
- $i_{sc}(k - 1) = [i_\gamma, i_\delta]$ is the scaled stator current vector in the previous step
- $u_{sc}(k) = [u_\gamma, u_\delta]$ is the scaled stator voltage vector in the actual step
- $e_{sc}(k) = [e_\gamma, e_\delta]$ is the scaled stator back-EMF voltage vector in the actual step
- $i'_{sc}(k) = [i_\gamma, -i_\delta]$ is the scaled complementary stator current vector in the actual step
- $\omega_{esc}(k)$ is the scaled electrical angular speed in the actual step
- i_{max} is the maximum current [A]
- e_{max} is the maximum back-EMF voltage [V]
- u_{max} is the maximum stator voltage [V]
- ω_{max} is the maximum electrical angular speed in [rad / s]

If the Luenberger-type stator current observer is properly designed in the stationary reference frame, the back-EMF can be estimated as a disturbance produced by the observer controller. However, this is only valid when the back-EMF term is not included in the observer model. The observer is a closed-loop current observer, therefore it acts as a state filter for the back-EMF term.

The estimate of the extended EMF term can be derived from Equation 27 on page 58 as follows:

$$-\frac{\hat{E}_{\gamma\delta}(s)}{E_{\gamma\delta}(s)} = \frac{sK_P + K_I}{s^2 L_D + sR_S + sK_P + K_I}$$

Equation 30

The observer controller can be designed by comparing the closed-loop characteristic polynomial with that of a standard second-order system as follows:

$$s^2 + \frac{K_P + R_S}{L_D} \cdot s + \frac{K_I}{L_D} = s^2 + 2\xi\omega_0 s + \omega_0^2$$

Equation 31

where:

- ω_0 is the natural frequency of the closed-loop system (loop bandwith)
- ξ is the loop attenuation
- K_P is the proporional gain
- k_I is the integral gain

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 parameters use the accumulator types.
- Accumulator output with floating-point inputs - the output is the accumulator result; the result is within the range <-1 ; 1). The inputs are 32-bit single precision floating-point values.

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

Table 2-9. Init versions

Function name	Parameters	Result type
AMCLIB_PMSMBemfObsrvDQInit_F16	AMCLIB_BEMF_OBSRV_DQ_T_A32 *	void

Table continues on the next page...

Table 2-9. Init versions (continued)

Function name	Parameters	Result type
	Initialization does not have any input.	
AMCLIB_PMSMBemfObsrvDQInit_A32fff	AMCLIB_BEMF_OBSRV_DQ_T_FLT *	void
	Initialization does not have any input.	

Table 2-10. Function versions

Function name	Input/output type		Result type
AMCLIB_PMSMBemfObsrvDQ_F16	**Input**	GMCLIB_2COOR_DQ_T_F16 *	frac16_t
		GMCLIB_2COOR_DQ_T_F16 *	
		frac16_t	
	Parameters	AMCLIB_BEMF_OBSRV_DQ_T_A32 *	
	Back-EMF observer with a 16-bit fractional input D-Q current and voltage, and a 16-bit electrical speed. All are within the range <-1 ; 1).		
AMCLIB_PMSMBemfObsrvDQ_A32fff	**Input**	GMCLIB_2COOR_DQ_T_FLT *	acc32_t
		GMCLIB_2COOR_DQ_T_FLT *	
		float_t	
	Parameters	AMCLIB_BEMF_OBSRV_DQ_T_FLT *	
	Back-EMF observer with a 32-bit single precision floating-point input D-Q current and voltage, and a 32-bit single precision floating-point electrical speed. All are within the full range. The output is a 32-bit accumulator angle error normalized to the range <-1 ; 1) that represents an angle (in radians) within the range <-π ; π).		

2.5.2 AMCLIB_BEMF_OBSRV_DQ_T_A32 type description

Variable name		Data type	Description
sEObsrv		GMCLIB_2COOR_DQ_T_F32	Estimated back-EMF voltage structure.
sIObsrv		GMCLIB_2COOR_DQ_T_F32	Estimated current structure.
sCtrl	f32ID_1	frac32_t	State variable in the alpha part of the observer, integral part at step k - 1. The variable is within the range <-1 ; 1).
	f32IQ_1	frac32_t	State variable in the beta part of the observer, integral part at step k - 1. The variable is within the range <-1 ; 1).
	a32PGain	acc32_t	The observer proportional gain is set up according to Equation 31 on page 59 as: $2\xi\omega_0 L_D - R_S$ The parameter is within the range <0 ; 65536.0). Set by the user.

Table continues on the next page...

Variable name		Data type	Description
	a32IGain	acc32_t	The observer integral gain is set up according to Equation 31 on page 59 as: $\omega_0^2 L_D$ The parameter is within the range <0 ; 65536.0). Set by the user.
a32IGain		acc32_t	The current coefficient gain is set up according to Equation 29 on page 58 as: $\frac{L_D}{L_D + T_s R_S}$ The parameter is within the range <0 ; 65536.0). Set by the user.
a32UGain		acc32_t	The voltage coefficient gain is set up according to Equation 29 on page 58 as: $\frac{T_s}{L_D + T_s R_S} \cdot \frac{u_{max}}{i_{max}}$ The parameter is within the range <0 ; 65536.0). Set by the user.
a32WIGain		acc32_t	The angular speed coefficient gain is set up according to Equation 29 on page 58 as: $\frac{L_Q T_s}{L_D + T_s R_S} \cdot \omega_{max}$ The parameter is within the range <0 ; 65536.0). Set by the user.
a32EGain		acc32_t	The back-EMF coefficient gain is set up according to Equation 29 on page 58 as: $\frac{T_s}{L_D + T_s R_S} \cdot \frac{e_{max}}{i_{max}}$ The parameter is within the range <0 ; 65536.0). Set by the user.
f16Error		frac16_t	Output - estimated phase error between a real D / Q frame system and an estimated D / Q reference system. The error is within the range <-1 ; 1).

2.5.3 AMCLIB_BEMF_OBSRV_DQ_T_FLT type description

Variable name		Data type	Description
sEObsrv		GMCLIB_2COOR_DQ_T_FLT	Estimated back-EMF voltage structure.
sIObsrv		GMCLIB_2COOR_DQ_T_FLT	Estimated current structure.
sCtrl	fltID_1	float_t	State variable in the alpha part of the observer; integral part at step k - 1. The variable is within the range <-1 ; 1).

Table continues on the next page...

Variable name		Data type	Description
	fltIQ_1	float_t	State variable in the beta part of the observer; integral part at step k - 1. The variable is within the range <-1 ; 1).
	fltPGain	float_t	Observer proportional gain is set up according to Equation 31 on page 59 as: $2\xi\omega_0 L_D - R_S$ The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
	fltIGain	float_t	The observer integral gain is set up according to Equation 31 on page 59 as: $\omega_0^2 L_D$ The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
fltIGain		float_t	The current coefficient gain is set up according to Equation 28 on page 58 as: $\frac{L_D}{L_D + T_s R_S}$ The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
fltUGain		float_t	The voltage coefficient gain is set up according to Equation 28 on page 58 as: $\frac{T_s}{L_D + T_s R_S}$ The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
fltWIGain		float_t	The angular speed coefficient gain is set up according to Equation 28 on page 58 as: $\frac{L_Q T_s}{L_D + T_s R_S}$ The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
fltEGain		float_t	The back-EMF coefficient gain is set up according to Equation 28 on page 58 as: $\frac{T_s}{L_D + T_s R_S}$ The parameter is a 32-bit single precision floating-point type non-negative value. Set by the user.
a32Error		acc32_t	Output - estimated phase error between a real D / Q frame system and an estimated D / Q reference system. The error is within the range <-1 ; 1).

2.5.4 Declaration

The available AMCLIB_PMSMBemfObsrvDQInit functions have the following declarations:

```
void AMCLIB_PMSMBemfObsrvDQInit_F16(AMCLIB_BEMF_OBSRV_DQ_T_A32 *psCtrl)
void AMCLIB_PMSMBemfObsrvDQInit_A32fff(AMCLIB_BEMF_OBSRV_DQ_T_FLT *psCtrl)
```

The available AMCLIB_PMSMBemfObsrvDQ functions have the following declarations:

```
frac16_t AMCLIB_PMSMBemfObsrvDQ_F16(const GMCLIB_2COOR_DQ_T_F16 *psIDQ, const
GMCLIB_2COOR_DQ_T_F16 *psUDQ, frac16_t f16Speed, AMCLIB_BEMF_OBSRV_DQ_T_A32 *psCtrl)
acc32_t AMCLIB_PMSMBemfObsrvDQ_A32fff(const GMCLIB_2COOR_DQ_T_FLT *psIDQ, const
GMCLIB_2COOR_DQ_T_FLT *psUDQ, float_t fltSpeed, AMCLIB_BEMF_OBSRV_DQ_T_FLT *psCtrl)
```

2.5.5 Function use

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

```
#include "amclib.h"

static GMCLIB_2COOR_DQ_T_F16       sIdq, sUdq;
static AMCLIB_BEMF_OBSRV_DQ_T_A32  sBemfObsrv;
static frac16_t f16Speed, f16Error;

void Isr(void);

void main (void)
{
  sBemfObsrv.sCtrl.a32PGain= ACC32(1.697);
  sBemfObsrv.sCtrl.a32IGain= ACC32(0.134);
  sBemfObsrv.a32IGain = ACC32(0.986);
  sBemfObsrv.a32UGain = ACC32(0.170);
  sBemfObsrv.a32WIGain= ACC32(0.110);
  sBemfObsrv.a32EGain = ACC32(0.116);

  /* Initialization of the observer's structure */
  AMCLIB_PMSMBemfObsrvDQInit_F16(&sBemfObsrv);

  sIdq.f16D = FRAC16(0.05);
  sIdq.f16Q = FRAC16(0.1);
  sUdq.f16D = FRAC16(0.2);
  sUdq.f16Q = FRAC16(-0.1);
}

/* Periodical function or interrupt */
void Isr(void)
{
  /* BEMF Observer calculation */
  f16Error = AMCLIB_PMSMBemfObsrvDQ_F16(&sIdq, &sUdq, f16Speed, &sBemfObsrv);
}
```

2.6 AMCLIB_TrackObsrv

The AMCLIB_TrackObsrv function calculates a tracking observer for the determination of angular speed and position of the input error functional signal. The tracking-observer algorithm uses the phase-locked-loop mechanism. It is recommended to call this function at every sampling period. It requires a single input argument as a phase error. A phase-tracking observer with a standard PI controller used as the loop compensator is shown in Figure 2-8.

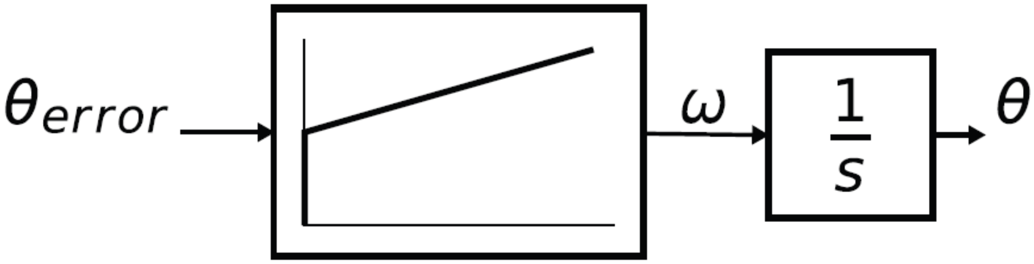

Figure 2-8. Block diagram of proposed PLL scheme for position estimation

The depicted tracking observer structure has the following transfer function:

$$\frac{\hat{\theta}(s)}{\theta(s)} = \frac{sK_P + K_I}{s^2 + sK_P + K_I}$$

Equation 32

The controller gains K_p and K_i are calculated by comparing the characteristic polynomial of the resulting transfer function to a standard second-order system polynomial.

The essential equations for implementation of the tracking observer according to the block scheme in Figure 2-8 are as follows:

$$\omega(k) = K_P \bullet e(k) + T_s \bullet K_I \bullet e(k) + \omega(k-1)$$

Equation 33

$$\theta(k) = T_s \cdot \omega(k) + \theta(k-1)$$

Equation 34

where:

- K_P is the proportional gain
- K_I is the integral gain
- T_s is the sampling period [s]
- e(k) is the position error in step k

- $\omega(k)$ is the rotor speed [rad / s] in step k
- $\omega(k - 1)$ is the rotor speed [rad / s] in step k - 1
- $\theta(k)$ is the rotor angle [rad] in step k
- $\theta(k - 1)$ is the rotor angle [rad] in step k - 1

In the fractional arithmetic, Equation 32 on page 64 and Equation 33 on page 64 are as follows:

$$\omega_{sc}(k) \cdot \omega_{max} = K_P \cdot e_{sc}(k) \cdot \theta_{max} + T_s \cdot K_I \cdot e_{sc}(k) \cdot \theta_{max} + \omega_{sc}(k-1) \cdot \omega_{max}$$

Equation 35

$$\theta_{sc}(k) \cdot \theta_{max} = T_s \cdot \omega_{sc}(k) \cdot \omega_{max} + \theta_{sc}(k-1) \cdot \theta_{max}$$

Equation 36

where:

- $e_{sc}(k)$ is the scaled position error in step k
- $\omega_{sc}(k)$ is the scaled rotor speed [rad / s] in step k
- $\omega_{sc}(k - 1)$ is the scaled rotor speed [rad / s] in step k - 1
- $\theta_{sc}(k)$ is the scaled rotor angle [rad] in step k
- $\theta_{sc}(k - 1)$ is the scaled rotor angle [rad] in step k - 1
- ω_{max} is the maximum speed
- θ_{max} is the maximum rotor angle (typically)

2.6.1 Available versions

The function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1).
- Accumulator output with floating point structure - the output is the accumulator result; the result is within the range <-1 ; 1). The structure of the parameters contains the 32-bit single precision floating-point values.

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

Table 2-11. Init versions

Function name	Init angle	Parameters	Result type
AMCLIB_TrackObsrvInit_F16	frac16_t	AMCLIB_TRACK_OBSRV_T_F32 *	void
	The input is a 16-bit fractional value of the angle normalized to the range <-1 ; 1) that represents an angle (in radians) within the range <-π ; π).		

Table continues on the next page...

Table 2-11. Init versions (continued)

Function name	Init angle	Parameters	Result type
AMCLIB_TrackObsrvInit_A32af	acc32_t	AMCLIB_TRACK_OBSRV_T_FLT *	void
	Input is the 32-bit accumulator value of the angle normalized to the range <-1 ; 1) that represents an angle in radians within the range <-π ; π). The parameters are 32-bit single precision values.		

Table 2-12. Function versions

Function name	Input type	Parameters	Result type
AMCLIB_TrackObsrv_F16	frac16_t	AMCLIB_TRACK_OBSRV_T_F32 *	frac16_t
	Tracking observer with a 16-bit fractional position error input divided by π. The output from the obsever is a 16-bit fractional position normalized to the range <-1 ; 1) that represents an angle (in radians) within the range <-π ; π).		
AMCLIB_TrackObsrv_A32af	acc32_t	AMCLIB_TRACK_OBSRV_T_FLT *	acc32_t
	Tracking observer with a 32-bit accumulator position divided by π. The output from the obsever is a 32-bit accumulator position normalized to the range <-1 ; 1) that represents an angle (in radians) within the range <-π ; π). The parameters are 32-bit single precision values.		

2.6.2 AMCLIB_TRACK_OBSRV_T_F32

Variable name	Input type	Description
f32Theta	frac32_t	Estimated position as the output of the second numerical integrator. The parameter is within the range <-1 ; 1). Controlled by the algorithm.
f32Speed	frac32_t	Estimated speed as the output of the first numerical integrator. The parameter is within the range <-1 ; 1). Controlled by the algorithm.
f32I_1	frac32_t	State variable in the controller part of the observer; integral part at step k - 1. The parameter is within the range <-1 ; 1). Controlled by the algorithm.
f16IGain	frac16_t	The observer integral gain is set up according to Equation 35 on page 65 as: $T_s \cdot K_I \cdot \frac{\theta_{max}}{\omega_{max}} \cdot 2^{-Ish}$ The parameter is a 16-bit fractional type within the range <0 ; 1). Set by the user.
i16IGainSh	int16_t	The observer integral gain shift takes care of keeping the f16IGain variable within the fractional range <-1 ; 1). The shift is determined as: $\log_2(T_s \cdot K_I \cdot \frac{\theta_{max}}{\omega_{max}}) - \log_2 1 < Ish \leq \log_2(T_s \cdot K_I \cdot \frac{\theta_{max}}{\omega_{max}}) - \log_2 0.5$ The parameter is a 16-bit integer type within the range <-15 ; 15>. Set by the user.
f16PGain	frac16_t	The observer proportional gain is set up according to Equation 35 on page 65 as: $K_P \cdot \frac{\theta_{max}}{\omega_{max}} \cdot 2^{-Psh}$ The parameter is a 16-bit fractional type within the range <0 ; 1). Set by the user.

Table continues on the next page...

Variable name	Input type	Description
i16PGainSh	int16_t	The observer proportional gain shift takes care of keeping the f16PGain variable within the fractional range <-1 ; 1). The shift is determined as: $\log_2(K_P \cdot \frac{\theta_{max}}{\omega_{max}}) - \log_2 1 < Psh \leq \log_2(K_P \cdot \frac{\theta_{max}}{\omega_{max}}) - \log_2 0.5$ The parameter is a 16-bit integer type within the range <-15 ; 15>. Set by the user.
f16ThGain	frac16_t	The observer gain for the output position integrator is set up according to Equation 36 on page 65 as: $T_s \cdot \frac{\omega_{max}}{\theta_{max}} \cdot 2^{-Thsh}$ The parameter is a 16-bit fractional type within the range <0 ; 1). Set by the user.
i16ThGainSh	int16_t	The observer gain shift for the position integrator takes care of keeping the f16ThGain variable within the fractional range <-1 ; 1). The shift is determined as: $\log_2(T_s \cdot \frac{\omega_{max}}{\theta_{max}}) - \log_2 1 < THsh \leq \log_2(T_s \cdot \frac{\omega_{max}}{\theta_{max}}) - \log_2 0.5$ The parameter is a 16-bit integer type within the range <-15 ; 15>. Set by the user.

2.6.3 AMCLIB_TRACK_OBSRV_T_FLT

Variable name	Input type	Description
f32Theta	frac32_t	Estimated position as the output of the second numerical integrator. The parameter is within the range <-1 ; 1). Controlled by the algorithm.
fltSpeed	float_t	Estimated speed as the output of the first numerical integrator. The parameter is within the full range. Controlled by the algorithm.
fltI_1	float_t	State variable in the controller part of the observer; integral part at the step k - 1. The parameter is within the full range. Controlled by the algorithm.
fltIGain	float_t	The observer integral gain is set up according to Equation 33 on page 64 as: $K_I T_s$ The parameter is a 32-bit single precision floating-point non-negative value. Set by the user.
fltPGain	float_t	The observer proportional gain is set up according to Equation 33 on page 64 as: K_P The parameter is a 32-bit single precision floating-point non-negative value. Set by the user.
fltThGain	float_t	The observer gain for the output position integrator is set up according to Equation 34 on page 64 as: T_s The parameter is a 32-bit single precision floating-point non-negative value. Set by the user.

2.6.4 Declaration

The available AMCLIB_TrackObsrvInit functions have the following declarations:

```
void AMCLIB_TrackObsrvInit_F16(frac16_t f16ThetaInit, AMCLIB_TRACK_OBSRV_T_F32 *psCtrl)
void AMCLIB_TrackObsrvInit_A32af(acc32_t a32ThetaInit, AMCLIB_TRACK_OBSRV_T_FLT *psCtrl)
```

The available AMCLIB_TrackObsrv functions have the following declarations:

```
frac16_t AMCLIB_TrackObsrv_F16(frac16_t f16Error, AMCLIB_TRACK_OBSRV_T_F32 *psCtrl)
acc32_t AMCLIB_TrackObsrv_A32af(acc32_t a32Error, AMCLIB_TRACK_OBSRV_T_FLT *psCtrl)
```

2.6.5 Function use

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

```
#include "amclib.h"

static AMCLIB_TRACK_OBSRV_T_F32  sTo;
static frac16_t      f16ThetaError;
static frac16_t      f16PositionEstim;

void Isr(void);

void main(void)
{
  sTo.f16IGain     = FRAC16(0.6434);
  sTo.i16IGainSh   = -9;
  sTo.f16PGain     = FRAC16(0.6801);
  sTo.i16PGainSh   = -2;
  sTo.f16ThGain    = FRAC16(0.6400);
  sTo.i16ThGainSh  = -4;

  AMCLIB_TrackObsrvInit_F16(FRAC16(0.0), &sTo);

  f16ThetaError    = FRAC16(0.5);
}

/* Periodical function or interrupt */
void Isr(void)
{
  /* Tracking observer calculation */
  f16PositionEstim = AMCLIB_TrackObsrv_F16(f16ThetaError, &sTo);
}
```

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

	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 A-12. Data storage

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

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

A.13 float_t

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

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

The 32-bit type is composed of:

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

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

The float_t type definition is as follows:

```
typedef float float_t;
```

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

Table A-13. Data storage - normalized values

	31							24	23							16	15							8	7							0
Value	S	Exponent								Mantissa																						
$(2.0 - 2^{-23}) \cdot 2^{127}$	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$\approx 3.40282 \cdot 10^{38}$	7				F				7				F				F				F				F				F			
$-(2.0 - 2^{-23}) \cdot 2^{127}$	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$\approx -3.40282 \cdot 10^{38}$	F				F				7				F				F				F				F				F			

Table continues on the next page...

Table A-13. Data storage - normalized values (continued)

2^{-126}	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\approx 1.17549 \cdot 10^{-38}$	0				0				8				0				0				0				0				0			
-2^{-126}	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\approx -1.17549 \cdot 10^{-38}$	8				0				8				0				0				0				0				0			
1.0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	3				F				8				0				0				0				0				0			
-1.0	1	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	B				F				8				0				0				0				0				0			
π	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	1	1	1	1	1	1	0	1	1	0	1	1
≈ 3.1415927	4				0				4				9				0				F				D				B			
-20810.086	1	1	0	0	0	1	1	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	0	0	0	0	1	0	1	1	0	0
	C				6				A				2				9				4				2				C			

Table A-14. Data storage - denormalized values

	31							24	23							16	15							8	7							0
Value	S	Exponent								Mantissa																						
0.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0				0				0				0				0				0				0				0			
-0.0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	8				0				0				0				0				0				0				0			
$(1.0 - 2^{-23}) \cdot 2^{-126}$	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$\approx 1.17549 \cdot 10^{-38}$	0				0				7				F				F				F				F				F			
$-(1.0 - 2^{-23}) \cdot 2^{-126}$	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$\approx -1.17549 \cdot 10^{-38}$	8				0				7				F				F				F				F				F			
$2^{-1} \cdot 2^{-126}$	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\approx 5.87747 \cdot 10^{-39}$	0				0				4				0				0				0				0				0			
$-2^{-1} \cdot 2^{-126}$	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\approx -5.87747 \cdot 10^{-39}$	8				0				4				0				0				0				0				0			
$2^{-23} \cdot 2^{-126}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
$\approx 1.40130 \cdot 10^{-45}$	0				0				0				0				0				0				0				1			
$-2^{-23} \cdot 2^{-126}$	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
$\approx -1.40130 \cdot 10^{-45}$	8				0				0				0				0				0				0				1			

Table A-15. Data storage - special values

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

A.14 GMCLIB_3COOR_T_F16

The GMCLIB_3COOR_T_F16 structure type corresponds to the three-phase stationary coordinate system, based on the A, B, and C components. Each member is of the frac16_t data type. The structure definition is as follows:

```
typedef struct
{
    frac16_t f16A;
    frac16_t f16B;
    frac16_t f16C;
} GMCLIB_3COOR_T_F16;
```

The structure description is as follows:

Table A-16. GMCLIB_3COOR_T_F16 members description

Type	Name	Description
frac16_t	f16A	A component; 16-bit fractional type
frac16_t	f16B	B component; 16-bit fractional type
frac16_t	f16C	C component; 16-bit fractional type

A.15 GMCLIB_3COOR_T_FLT

The GMCLIB_3COOR_T_FLT structure type corresponds to the three-phase stationary coordinate system, based on the A, B, and C components. Each member is of the float_t data type. The structure definition is as follows:

```
typedef struct
{
```

```
    float_t fltA;
    float_t fltB;
    float_t fltC;
} GMCLIB_3COOR_T_FLT;
```

The structure description is as follows:

Table A-17. GMCLIB_3COOR_T_FLT members description

Type	Name	Description
float_t	fltA	A component; 32-bit single precision floating-point type
float_t	fltB	B component; 32-bit single precision floating-point type
float_t	fltC	C component; 32-bit single precision floating-point type

A.16 GMCLIB_2COOR_ALBE_T_F16

The GMCLIB_2COOR_ALBE_T_F16 structure type corresponds to the two-phase stationary coordinate system, based on the Alpha and Beta orthogonal components. Each member is of the frac16_t data type. The structure definition is as follows:

```
typedef struct
{
    frac16_t f16Alpha;
    frac16_t f16Beta;
} GMCLIB_2COOR_ALBE_T_F16;
```

The structure description is as follows:

Table A-18. GMCLIB_2COOR_ALBE_T_F16 members description

Type	Name	Description
frac16_t	f16Apha	α-component; 16-bit fractional type
frac16_t	f16Beta	β-component; 16-bit fractional type

A.17 GMCLIB_2COOR_ALBE_T_FLT

The GMCLIB_2COOR_ALBE_T_FLT structure type corresponds to the two-phase stationary coordinate system based on the Alpha and Beta orthogonal components. Each member is of the float_t data type. The structure definition is as follows:

```
typedef struct
{
    float_t fltAlpha;
    float_t fltBeta;
} GMCLIB_2COOR_ALBE_T_FLT;
```

The structure description is as follows:

Table A-19. GMCLIB_2COOR_ALBE_T_FLT members description

Type	Name	Description
float_t	fltApha	α-component; 32-bit single precision floating-point type
float_t	fltBeta	β-component; 32-bit single precision floating-point type

A.18 GMCLIB_2COOR_DQ_T_F16

The GMCLIB_2COOR_DQ_T_F16 structure type corresponds to the two-phase rotating coordinate system, based on the D and Q orthogonal components. Each member is of the frac16_t data type. The structure definition is as follows:

```
typedef struct
{
    frac16_t f16D;
    frac16_t f16Q;
} GMCLIB_2COOR_DQ_T_F16;
```

The structure description is as follows:

Table A-20. GMCLIB_2COOR_DQ_T_F16 members description

Type	Name	Description
frac16_t	f16D	D-component; 16-bit fractional type
frac16_t	f16Q	Q-component; 16-bit fractional type

A.19 GMCLIB_2COOR_DQ_T_F32

The GMCLIB_2COOR_DQ_T_F32 structure type corresponds to the two-phase rotating coordinate system, based on the D and Q orthogonal components. Each member is of the frac32_t data type. The structure definition is as follows:

```
typedef struct
{
    frac32_t f32D;
    frac32_t f32Q;
} GMCLIB_2COOR_DQ_T_F32;
```

The structure description is as follows:

Table A-21. GMCLIB_2COOR_DQ_T_F32 members description

Type	Name	Description
frac32_t	f32D	D-component; 32-bit fractional type
frac32_t	f32Q	Q-component; 32-bit fractional type

A.20 GMCLIB_2COOR_DQ_T_FLT

The GMCLIB_2COOR_DQ_T_FLT structure type corresponds to the two-phase rotating coordinate system, based on the D and Q orthogonal components. Each member is of the float_t data type. The structure definition is as follows:

```
typedef struct
{
    float_t fltD;
    float_t fltQ;
} GMCLIB_2COOR_DQ_T_FLT;
```

The structure description is as follows:

Table A-22. GMCLIB_2COOR_DQ_T_FLT members description

Type	Name	Description
float_t	fltD	D-component; 32-bit single precision floating-point type
float_t	fltQ	Q-component; 32-bit single precision floating-point type

A.21 GMCLIB_2COOR_SINCOS_T_F16

The GMCLIB_2COOR_SINCOS_T_F16 structure type corresponds to the two-phase coordinate system, based on the Sin and Cos components of a certain angle. Each member is of the frac16_t data type. The structure definition is as follows:

```
typedef struct
{
    frac16_t f16Sin;
    frac16_t f16Cos;
} GMCLIB_2COOR_SINCOS_T_F16;
```

The structure description is as follows:

Table A-23. GMCLIB_2COOR_SINCOS_T_F16 members description

Type	Name	Description
frac16_t	f16Sin	Sin component; 16-bit fractional type
frac16_t	f16Cos	Cos component; 16-bit fractional type

A.22 GMCLIB_2COOR_SINCOS_T_FLT

The GMCLIB_2COOR_SINCOS_T_FLT structure type corresponds to the two-phase coordinate system, based on the Sin and Cos components of a certain angle. Each member is of the float_t data type. The structure definition is as follows:

```
typedef struct
{
    float_t fltSin;
    float_t fltCos;
} GMCLIB_2COOR_SINCOS_T_FLT;
```

The structure description is as follows:

Table A-24. GMCLIB_2COOR_SINCOS_T_FLT members description

Type	Name	Description
float_t	fltSin	Sin component; 32-bit single precision floating-point type
float_t	fltCos	Cos component; 32-bit single precision floating-point type

A.23 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.24 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.25 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.26 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.27 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.28 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.29 ACC32

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

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

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

```
#include "mlib.h"

static acc32_t a32Val;

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

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number CM4FAMCLIBUG
Revision 1, 06/2016