eclipseina

EC-LIB®

DEMO VERSION FOR PC

User Manual

2020

EC-LIB® - User Manual for the Demo Version

Contents

1 Introduction

This manual "EC-LIB® Function Library – Demo Version Manual" provides you with a short overview of required information to use the EC-LIB® Function Library Demo Version for fixed point 32-bit mathematics within a sample project.

Technical background

This demo version is intended to evaluating the functionality and usability of the EC-LIB® on a PC instead of the target microcontroller which is going to be used later.
This demo version of the EC-LIB® has the following technical basis:

- The implemented macros and functions comply to the C99 standard.
- It was compiled using the GCC compiler.

Motivation

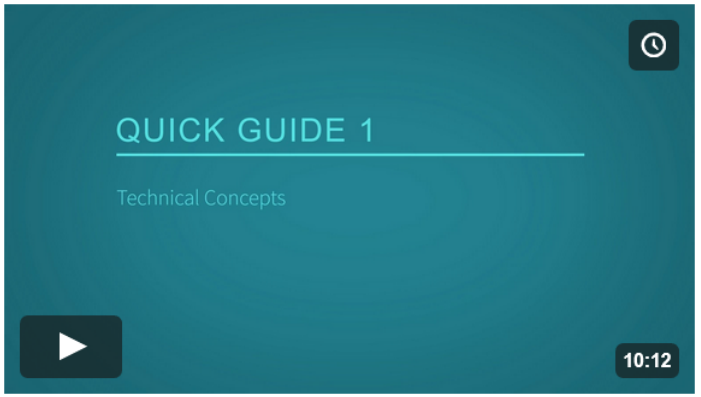
EC-LIB® was created to help developers avoid interruptions to their development workflow when implementing fixed point mathematics in applications where it is preferred to floating point arithmetic.

2 Getting Started

For a quick and easy start with the EC-LIB® watch the video content EC-LIB® Quick Guide on Vimeo.

2.1 Quick Guide

In this Quick Guide you will be introduced to the conceptual background as well as the steps of integration, definition of variables and their initialization. Moreover, you will learn how to implement a sample project for the comparison of the volumes of two different geometric bodies.



The following Quick Guide videos provide you with all necessary information and guide you in implementing your first test project:

1. Technical concept
   https://vimeo.com/298124324
2. Download
   https://vimeo.com/298129849
3. Integrating the EC-LIB® into your Eclipse project
   https://vimeo.com/298129863
4. Eclipse Configuration and File Setup
   https://vimeo.com/298129874
5. Web Tool and Choice of Shift Factor
   https://vimeo.com/298129883
6. Coding
   https://vimeo.com/298129902

2.2 Accompanying Text to the Quick Guide

Complementary to the videos 3, 4 and 6 a short written guide explains how to set up your Eclipse project and integrate the EC-LIB®, as well as the full code for the sample project.

For C code examples the Eclipse environment syntax highlighting is used. To distinguish macros, functions and variables easily, `code font` has been used.

violet: C control statements
green: C code comments
bold: Function or macro call

Project Setup

The following instructions explain in detail how to set up your Eclipse project and integrate the EC-LIB®.

1. Download the EC-LIB® Demo Version and unzip it. You will find the folder `EC-LIB`.
2. Create your own C project in Eclipse, for example in Eclipse IDE for C/C++ Developers.
   To do so open Eclipse IDE for C/C++ Developers and select: `File` > `New` > `C/C++ Project`
   Now select `C Managed Build` and name your project `EclibExample`. Finish by clicking on `Next` > `Next` > `Finish`
   For more information please visit the Eclipse Help.
3. Integrate the EC-LIB® into your project
   a) Copy the folders `EC-LIB_Functions` and `EC-LIB_Headers` from `EC-LIB` and past them in to your project folder `eclipse-workspace` / `EclibExample`.
   b) Open your project in Eclipse and right click on the folder `EclibExample` and select `Refresh`. Now you will be able to fold out the folder `EclibExample` and see the folders `EC-LIB_Functions` and `EC-LIB_Headers`.
4. Configurate your project by right click on the folder `EclibExample` in Eclipse and select `Properties`.
   A new window will pop up. Fold out `C/C++ Build` and select `Settings`. On the right site you will see the settings menu.
   a) Select the dialect by clicking on `Dialect` and select the `Language standard ISO C99(-std=c99)` form the drop down menu.
   b) Include the header by clicking on `Includes`. On the left you will see `Include paths(-I)` click on the symbol `Add` with the green plus next to it.
   A new window will pop up where you insert `ec32lib_demo` as the `Directory`.
   Click on `Workspace` and a new window will pop up.
   Fold out `EclibExample` and select `EC-LIB_Headers`
   Close the two windows by clicking `OK` twice.
   c) Include the library by clicking on `Libraries`.
      i. On the left you will see `Libraries search path(-L)` click on the symbol `Add` with the green plus next to it.
      A new window will pop up where you insert `ec32lib_demo` as the `Directory`.
      Click on `Workspace` and a new window will pop up.
      Fold out `EclibExample` and select `EC-LIB_Functions`.
      Close the two windows by clicking `OK` twice.
      ii. On the left you will see `Libraries (-l)` click on the symbol `Add` with the green plus next to it.
      A new window will pop up where you insert `ec32lib_demo` and click `OK`.
   d) Finish by clicking `Apply and close`.
   e) If you are asked, if you wish to rebuild click `Yes`.

5. Now you can set up the File.
   a) Create a new source file by right click on the folder `EclibExample` in Eclipse and select `New` > `Source File`. A new window will pop up. Name your file `VolumeComparator.c` and click `Finish`.
   b) Write in your source file

```
#include"ec32demo.h"
#include<stdio.h>
int main(){
return 0;
}
```

   c) Build your project by clicking on the hammer symbol `Build 'Debug' for project 'EclibExample'`

No error messages should occur. Congratulations! Now you can start with the sample project.

Sample Project

Here you find the full code for the sample project, which is explained in detail in the videos 5 and 6.

```
//Define
//Pyramid
ECLIB_FIX32_19SR(volumePyramid);
ECLIB_FIX32_25SR(widthPyramid);
ECLIB_FIX32_24SR(heightPyramid);
//Cuboid
ECLIB_FIX32_21SR(volumeCuboid);
ECLIB_FIX32_26SR(widthCuboid);
ECLIB_FIX32_26SR(heightCuboid);
ECLIB_FIX32_26SR(lengthCuboid);
int main(){
      ECLIB_Bool resComparison = EC_FALSE;
       //Initialize
       //Pyramid
      ECLIB_SET_FIX32(widthPyramid, 25);
      ECLIB_SET_FIX32(heightPyramid, 15);
      ECLIB_SET_FIX32(volumePyramid, 0);
       //Cuboid
      ECLIB_SET_FIX32(widthCuboid, 10.202);
      ECLIB_SET_FIX32(heightCuboid, 10.38);
      ECLIB_SET_FIX32(lengthCuboid, 8.1);
      ECLIB_SET_FIX32(volumeCuboid, 0);
```

```
      //Calculate
      //Pyramid
    calculatevolumePyramide( &volumePyramid,
    &volumePyramid_sf, &heightPyramid,
    &heightPyramid_sf, &widthPyramid,
    &widthPyramid_sf);
      //Cuboid
    calculatevolumeCuboid(&volumeCuboid,
    &volumeCuboid_sf, &heightCuboid,
    &heightCuboid_sf, &widthCuboid,
    &widthCuboid_sf, &lengthCuboid,
    &lengthCuboid_sf);
      //Compare
    ECLIB_IS_LESS_OR_EQUAL(resComparison,
    volumePyramid, volumeCuboid);
     if (resComparison){
        printf("The pyramid is less than or equal
        to the cuboid in volume.");
     }
     else{
        printf("The pyramid is larger in volume
        than the cuboid.");
     }
     return 0;
}
```

```
//Calculate
//Pyramid
void calculatevolumePyramide(s32 *volume, const
s8 *volume_sf, s32 *height, const s8 *height_sf,
s32 *width, const s8 *width_sf){
      //Define
     ECLIB_FIX32_20SR(area);
     ECLIB_FIX32_19SR(res);
      //Calculate
     ECLIB_SQUARE_32(area, width);
     area = area/3;
     ECLIB_MUL_32(res, area, height);
     *volume = res;
}
//Cuboid
void calculatevolumeCuboid(s32 *volume, const s8
*volume_sf, s32 *height, const s8 *height_sf, s32
*width, const s8 *width_sf, s32 *length, const s8
*length_sf){
      //Define
     ECLIB_FIX32_22SR(area);
     ECLIB_FIX32_21SR(res);
      //Calculate
     ECLIB_MUL_32(area, width, length);
     ECLIB_MUL_32(res, area, height);
     *volume = res;
}
```

3 Structure of a Function and Macro Description

Function and macro descriptions are described systematically by using the following format:

Name of the Macro	Mathematical Operation	Short Description

Concept (optional)

Handling
- Macro Call
- Function Call
- Parameter Values of the Function

Implementation

Error Handling

Macro calls and function calls are represented differently in order to distinguish them easily: For macros, only upper case characters are used to describe the operation, e.g. `MUL`. For function calls, only the first letter of the operation name is upper case followed by lower case letters, e.g. `Mul`.

`ECLIB_MUL_16`	Macro call
`ECLIB_Mul_16`	Function call

4 Definition and Initialization

4.1 Fixed Point Definition with Shift Factor Left

ECLIB_FIX32_XSL	**Defines a fixed point parameter to an input parameter and given left shift factor X**

Concept

This macro defines the signed 32-bit parameter `par` and the constant signed 8-bit parameter `par_sf` and initializes `par_sf` with the given left shift factor X.

Handling

Macro Call:

```
ECLIB_FIX32_XSL(par);
```

Result

Data Type	Name	Explanation
`s32`	`par`	definition of the signed 32-bit parameter `par`
`const s8`	`par_sf`	definition and initialization of the signed 8-bit parameter `par_sf` with -X

Error Handling

-

4.2 Fixed Point Definition with Shift Factor Right

ECLIB_FIX32_XSR	**Defines a fixed point parameter to an input parameter and given right shift factor X**

Concept

This macro defines the signed 32-bit parameter `par` and the constant signed 8-bit parameter `par_sf` and initializes `par_sf` with the given right shift factor X.

Handling

Macro Call:

```
ECLIB_FIX32_XSR(par);
```

Result

Data Type	Name	Explanation
`s32`	`par`	definition of the signed 32-bit parameter `par`
`const s8`	`par_sf`	definition and initialization of the signed 8-bit parameter `par_sf` with X

Error Handling

-

4.3 Set Value

ECLIB_SET_FIX32	$f(x) = x$	**Sets fixed point number to the given floating value**

Concept

This macro sets the fixed point number `par` based on a floating point value.
Precondition: The parameter par is already defined as the first parameter of a `ECLIB_FIX32` data structure.

Handling

Macro Call: `ECLIB_SET_FIX32(par, par_float);`

Parameter Values

Direction	Data Type	Data		Explanation
output	`ECLIB_FIX32`	`s32`	`par`	parameter that needs to be set to the constant given floating value `par_float`
input		`const s8`	`par_sf`	shift factor of `par`
input	`double/float`		`par_float`	given floating value

Implementation

The result of the `ECLIB_SET_FIX32` macro is computed as followed:

$$result = par\_float \cdot 2^{par\_sf}$$

It yields:

$$par = \begin{cases} \text{S32\_NEG\_INF} & \text{if } result \leq -2147483647 \\ result & \text{if } -2147483647 < result < 2147483647 \\ \text{S32\_POS\_INF} & \text{if } 2147483647 \leq result \\ \text{S32\_NAN} & \text{if } par\_float \text{ is the floating value NAN} \end{cases}$$

Error Handling

-

5 Demo Version — Functions and Macros

5.1 Multiplication

ECLIB_MUL_32	$f(x, y) = x \cdot y$	**Product of two fixed point numbers**

Handling using Macro or Function

Macro Call: `ECLIB_MUL_32(res, par1, par2);`

This macro replaces

```
"ECLIB_MUL_32"  with "ECLIB_Mul_32"
"res"           with "&res,&res_sf"
"par1"          with "par1, par1_sf"
"par2"          with "par2, par2_sf"
```

This is equivalent to the following
Function Call:

```
ECLIB_Mul_32(&res, &res_sf, par1, par1_sf, par2, par2_sf);
```

Parameter Values of the Function

Direction	Data Type	Data		Explanation
output	ECLIB_FIX32	s32	*res	pointer to the product of par1 and par2
input		const s8	*res_sf	pointer to the shift factor of res
input	ECLIB_FIX32	s32	par1	first factor
input		const s8	par1_sf	shift factor of par1
input	ECLIB_FIX32	s32	par2	second factor
input		const s8	par2_sf	shift factor of par2

Implementation

The product of two ECLIB_FIX32 numbers is calculated as:

$$result = \left(\frac{par1}{2^{par1\_sf}} \cdot \frac{par2}{2^{par2\_sf}} \right) \cdot 2^{res\_sf}$$

It yields:

$$res = \begin{cases} \text{S32\_NEG\_INF} & \text{if } result \leq -2147483647 \\ result & \text{if } -2147483647 < result < 2147483647 \\ \text{S32\_POS\_INF} & \text{if } 2147483647 \leq result \end{cases}$$

Error Handling

- **Structure of conditions**: Subsequent level conditions are only evaluated if the previous level condition is not fulfilled for the input combination.
- Simplification: As the multiplication is a **commutative** mathematical operation, we obtain `par1·par2 == par2 ·par1`. Hence, the conditions can also be read vice versa within this operation.
- Wording: The term **par** indicates a parameter.
  The term **INF** is an abbreviation for "Infinity" (`POS_INF` or `NEG_INF`).

Condition - 1st level (these primary conditions are checked first)	**Result `res`**
`S32_NAN · par`	`S32_NAN`
`S32_INF · 0`	`S32_NAN`

Condition - 2nd level (values and conditions not covered by 1st level)	**Result `res`**
`(S32_POS_INF · par) && (par >0)`	`S32_POS_INF`
`(S32_POS_INF · par) && (par <0)`	`S32_NEG_INF`
`(S32_NEG_INF · par) && (par >0)`	`S32_NEG_INF`
`(S32_NEG_INF · par) && (par <0)`	`S32_POS_INF`

5.2 Square

ECLIB_SQUARE_32	$f(x) = x^2$	**Square of a fixed point number**

Handling using Macro or Function

Macro Call:
```
ECLIB_SQUARE_32(res, par);
```
This macro replaces

```
"ECLIB_SQUARE_32" with "ECLIB_Square_32"
"res"             with "&res,&res_sf"
"par"             with "par, par_sf"
```

This is equivalent to the following
Function Call:

```
ECLIB_Square_32(&res, &res_sf, par, par_sf);
```

Parameter Values of the Function

Direction	Data Type	Data		Explanation
output	ECLIB_FIX32	s32	*res	pointer to the square of par
input		const s8	*res_sf	pointer to the shift factor of res
input	ECLIB_FIX32	s32	par	parameter
input		const s8	par_sf	shift factor of par

Implementation

The square of a given `ECLIB_FIX32` number is calculated as:

$$result = \left(\frac{par}{2^{par\_sf}}\right)^2 \cdot 2^{res\_sf}$$

It yields:

$$res = \begin{cases} \text{S32\_NEG\_INF} & \text{if } result \leq -2147483647 \\ result & \text{if } -2147483647 < result < 2147483647 \\ \text{S32\_POS\_INF} & \text{if } 2147483647 \leq result \end{cases}$$

Error Handling

- Wording: **S32_INF** is an abbreviation for "Infinity" (`S32_POS_INF` or `S32_NEG_INF`).

Condition - 1st level (these primary conditions are checked first)	**Result `res`**
`(S32_NAN)`^{2}	`S32_NAN`
`(S32_INF)`^{2}	`S32_POS_INF`

5.3 Is Equal

ECLIB_IS_EQUAL_32	$f(x, y) = \text{ECLIB\_TRUE} \Leftrightarrow x = y$	**Checks if two given fixed point numbers are equal**

Handling using Macro or Function

Macro Call: `ECLIB_IS_EQUAL_32(res, par1, par2);`
This macro replaces

`"ECLIB_IS_EQUAL_32"`	with	`"ECLIB_IsEqual_32"`
`"par1"`	with	`"par1, par1_sf"`
`"par2"`	with	`"par2, par2_sf"`

and sets `res` as the return value of the function.

This is equivalent to the following
Function Call:

```
res = ECLIB_IsEqual_32(par1, par1_sf, par2, par2_sf);
```

Parameter Values of the Function

Direction	Data Type	Data		Explanation
output	`ECLIB_Bool`		`res`	result containing either `ECLIB_TRUE` or `ECLIB_FALSE`
input	`ECLIB_FIX32`	`s32`	`par1`	first parameter
input		`const s8`	`par1_sf`	shift factor of `par1`
input	`ECLIB_FIX32`	`s32`	`par2`	second parameter
input		`const s8`	`par2_sf`	shift factor of `par2`

Implementation

We obtain the following:

$$res = \begin{cases} \text{ECLIB\_TRUE} & \text{if } shifted\_par1 = shifted\_par2 \\ \text{ECLIB\_FALSE} & \text{else} \end{cases}$$

Error Handling

- Simplification: As the is equal operation is a **commutative** mathematical operation, we obtain `par1 == par2 ⇔ par2 == par1`. Hence, the conditions can also be read vice versa within this operation.
- Wording: The term **par** indicates a parameter.
- **Note:** To check if a parameter is equal to `S32_NAN`, use the macro `ECLIB_IS_NAN_32`, as `ECLIB_IS_EQUAL_32(S32_NAN,S32_NAN) == ECLIB_FALSE`.

Condition - 1st level (these primary conditions are checked first)	**Result `res`**
`S32_NAN == par`	`ECLIB_FALSE`
`S32_POS_INF == S32_POS_INF`	`ECLIB_TRUE`
`S32_NEG_INF == S32_NEG_INF`	`ECLIB_TRUE`
`S32_POS_INF == S32_NEG_INF`	`ECLIB_FALSE`

5.4 Is Less or Equal

ECLIB_IS_LESS_OR_EQUAL_32	$f(x, y)$ $\text{ECLIB\_TRUE} \Leftrightarrow x \leq y$	=	**Checks if the first parameter is less than or equal to the second parameter**

Handling using Macro or Function

Macro Call: `ECLIB_IS_LESS_OR_EQUAL_32(res, par1, par2);`
This macro replaces

`"ECLIB_IS_LESS_OR_EQUAL_32"`	with	`"ECLIB_IsLessOrEqual_32"`
`"par1"`	with	`"par1, par1_sf"`
`"par2"`	with	`"par2, par2_sf"`

and sets `res` as the return value of the function.

This is equivalent to the following
Function Call:

```
res = ECLIB_IsLessOrEqual_32(par1, par1_sf, par2, par2_sf);
```

Parameter Values of the Function

Direction	Data Type	Data		Explanation
output	`ECLIB_Bool`		`res`	result containing either `ECLIB_TRUE` or `ECLIB_FALSE`
input	`ECLIB_FIX32`	`s32`	`par1`	first parameter
input		`const s8`	`par1_sf`	shift factor of `par1`
input	`ECLIB_FIX32`	`s32`	`par2`	second parameter
input		`const s8`	`par2_sf`	shift factor of `par2`

Implementation

We obtain the following:

$$res = \begin{cases} \text{ECLIB\_TRUE} & \text{if } shifted\_par1 \leq shifted\_par2 \\ \text{ECLIB\_FALSE} & \text{else} \end{cases}$$

Error Handling

- **Structure of conditions**: Subsequent level conditions are only evaluated if the previous level condition is not fulfilled for the input combination.
- Wording: The term **par** indicates a parameter.

Condition - 1st level (these primary conditions are checked first)	Result `res`
`S32_NAN <= par`	`ECLIB_FALSE`
`par <= S32_NAN`	`ECLIB_FALSE`

Condition - 2nd level (values and conditions not covered by 1st level)	**Result `res`**
`par <= S32_POS_INF`	`ECLIB_TRUE`
`S32_NEG_INF <= par`	`ECLIB_TRUE`

Condition - 3rd level (values and conditions not covered by 1st and 2nd level)	**Result `res`**
`S32_POS_INF <= par`	`ECLIB_FALSE`
`par <= S32_NEG_INF`	`ECLIB_FALSE`

eclipseina

EC-LIB©
FUNCTION LIBRARY

For further information about our products please visit eclipseina.com