



---

## 16-Bit Language Tools Libraries Reference Manual

---

### Notice to Development Tools Customers

---



**Important:**

All documentation becomes dated, and Development Tools manuals are no exception. Our tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our website ([www.microchip.com/](http://www.microchip.com/)) to obtain the latest version of the PDF document.

Documents are identified with a DS number located on the bottom of each page. The DS format is DS<DocumentNumber><Version>, where <DocumentNumber> is an 8-digit number and <Version> is an uppercase letter.

**For the most up-to-date information**, find help for your tool at [onlinedocs.microchip.com/](http://onlinedocs.microchip.com/).

---



---

---

## Table of Contents

---

Notice to Development Tools Customers.....	1
1. Preface.....	4
1.1. Conventions Used in This Guide.....	4
1.2. Recommended Reading.....	5
2. Libraries Overview.....	6
2.1. OMF-Specific Libraries and Startup Modules.....	6
2.2. Startup Code.....	6
2.3. DSP Library.....	7
2.4. 16-Bit Peripheral Libraries.....	7
2.5. Standard C Libraries with Math and Support Functions.....	7
2.6. Fixed-Point Math Functions.....	7
2.7. Compiler Built-in Functions.....	7
3. Standard C and Math Libraries.....	8
3.1. Using the Standard C and Math Libraries.....	8
3.2. List of Standard C and Math Library Functions.....	8
4. Support Libraries.....	9
4.1. Rebuilding the libpic30 Library.....	9
4.2. Standard C Library Helper Functions.....	9
4.3. Standard C Library Functions That Require Modification.....	9
4.4. Functions/Constants to Support A Simulated UART.....	9
4.5. Functions for Erasing and Writing EEDATA Memory.....	13
4.6. Functions for Erasing and Writing Flash Memory.....	16
4.7. Functions for Specialized Copying and Initialization.....	20
4.8. Functions to Support Secondary Core PRAM.....	28
5. Fixed-Point Math Functions.....	31
5.1. Overview of Fixed-Point Data Formats.....	31
5.2. Using the Fixed-Point Libraries.....	35
5.3. <libq.h> Mathematical Functions.....	36
6. Revision History.....	59
6.1. Revision P (January 2022).....	59
6.2. Revision N (February 2021).....	59
6.3. Revision M (July 2020).....	59
6.4. Revision L (December 2019).....	59
6.5. Revision K (February 2018).....	59
6.6. Revision J (December 2014).....	60
6.7. Revision H (September 2013).....	60
6.8. Revision G (October 2010).....	60
6.9. Revision F (March 2009).....	60
6.10. Revision E (January 2008).....	61
6.11. Revision D (December 2006).....	61
6.12. Revision C (October 2005).....	62

6.13. Revision B (September 2004).....	63
6.14. Revision A (May 2004).....	63
The Microchip Website.....	64
Product Change Notification Service.....	64
Customer Support.....	64
Microchip Devices Code Protection Feature.....	64
Legal Notice.....	64
Trademarks.....	65
Quality Management System.....	66
Worldwide Sales and Service.....	67

## 1. Preface

MPLAB® XC16 Libraries documentation and support information is discussed in this section.

### 1.1 Conventions Used in This Guide

The following conventions may appear in this documentation:

**Table 1-1. Documentation Conventions**

Description	Represents	Examples
<b>Arial font:</b>		
Italic characters	Referenced books	<i>MPLAB® IDE User's Guide</i>
	Emphasized text	...is the <i>only</i> compiler...
Initial caps	A window	the Output window
	A dialog	the Settings dialog
	A menu selection	select Enable Programmer
Quotes	A field name in a window or dialog	"Save project before build"
Underlined, italic text with right angle bracket	A menu path	<u><i>File&gt;Save</i></u>
Bold characters	A dialog button	Click <b>OK</b>
	A tab	Click the <b>Power</b> tab
N'Rnnnn	A number in verilog format, where N is the total number of digits, R is the radix and n is a digit.	4'b0010, 2'hF1
Text in angle brackets < >	A key on the keyboard	Press <Enter>, <F1>
<b>Courier New font:</b>		
Plain Courier New	Sample source code	#define START
	Filenames	autoexec.bat
	File paths	c:\mcc18\h
	Keywords	_asm, _endasm, static
	Command-line options	-Opa+, -Opa-
	Bit values	0, 1
	Constants	0xFF, 'A'
Italic Courier New	A variable argument	<i>file.o</i> , where <i>file</i> can be any valid filename
Square brackets [ ]	Optional arguments	mcc18 [options] file [options]
Curly brackets and pipe character: {   }	Choice of mutually exclusive arguments; an OR selection	errorlevel {0 1}

.....continued		
Description	Represents	Examples
Ellipses...	Replaces repeated text	var_name [, var_name...]
	Represents code supplied by user	void main (void) { ... }

## 1.2 Recommended Reading

This guide describes how to use the MPLAB XC16 C Compiler libraries. Other useful documents are listed below. The following Microchip documents are available and recommended as supplemental reference resources.

### Release Notes (Readme Files)

For information on Microchip tools, read the associated Release Notes (HTML files) included with the software.

### MPLAB® XC16 C Compiler User's Guide (DS50002071)

A guide to using the 16-bit C compiler. The 16-bit linker is used with this tool.

### MPLAB® XC16 Assembler, Linker and Utilities User's Guide (DS50002106)

A guide to using the 16-bit assembler, object linker, object archiver/librarian and various utilities.

### Device-Specific Documentation

The Microchip website contains many documents that describe 16-bit device functions and features, including:

- Individual and family data sheets
- Family reference manuals
- Programmer's reference manuals

### C Standards Information

American National Standard for Information Systems – *Programming Language – C*. American National Standards Institute (ANSI), 11 West 42nd. Street, New York, New York, 10036.

This standard specifies the form and establishes the interpretation of programs expressed in the programming language C. Its purpose is to promote portability, reliability, maintainability and efficient execution of C language programs on a variety of computing systems.

### C Reference Manuals

Harbison, Samuel P. and Steele, Guy L., *C A Reference Manual*, Fourth Edition, Prentice-Hall, Englewood Cliffs, N.J. 07632.

Kernighan, Brian W. and Ritchie, Dennis M., *The C Programming Language*, Second Edition. Prentice Hall, Englewood Cliffs, N.J. 07632.

Kochan, Steven G., *Programming In ANSI C*, Revised Edition. Hayden Books, Indianapolis, Indiana 46268.

Plauger, P.J., *The Standard C Library*, Prentice-Hall, Englewood Cliffs, N.J. 07632.

Van Sickle, Ted., *Programming Microcontrollers in C*, First Edition. LLH Technology Publishing, Eagle Rock, Virginia 24085.

## 2. Libraries Overview

A library is a collection of functions grouped for reference and ease of linking. See the “*MPLAB® XC16 Assembler, Linker and Utilities User’s Guide*” (DS50002106) for more information about making and using libraries.

### Compiler Installation Locations

The majority of the libraries discussed in this manual come with the MPLAB XC16 C Compiler, which is installed by default in the following locations:

- Windows OS 64-bit - C:\Program Files\Microchip\xc16\x.xx
- Mac OS - Applications/microchip/xc16/x.xx
- Linux OS - /opt/microchip/xc16/x.xx

where x.xx is the version number.

### Assembly Code Applications

Free versions of the 16-bit language tool libraries are available from the Microchip web site. DSP and 16-bit peripheral libraries are provided with object files and source code. A math library (containing functions from the standard C header file `<math.h>`) is provided as an object file only. The complete standard C library is provided with the MPLAB XC16 C Compiler.

### C Code Applications

The 16-bit language tool libraries are included in the `lib` subdirectory of the MPLAB XC16 C Compiler install directory (see “Compiler Installation Locations”). These libraries can be linked directly into an application with a 16-bit linker.

## 2.1 OMF-Specific Libraries and Startup Modules

Library files and start-up modules are specific to OMF (Object Module Format). An OMF can be one of the following:

- ELF – Executable and Linkable Format (default). The debugging format used for ELF object files is DWARF 2.0.
- COFF – Common Object File Format.

There are two ways to select the OMF:

1. Set an environment variable called `XC16_OMF` for all tools.
2. Select the OMF on the command line when invoking the tool, i.e., `-omf=omf` or `-momf=omf`.

16-bit tools will first look for generic library files when building your application (no OMF specification). If these cannot be found, the tools will look at your OMF specifications and determine which library file to use.

As an example, if `libdsp.a` is not found and no environment variable or command-line option is set, the file `libdsp-coff.a` will be used by default.

## 2.2 Startup Code

In order to initialize variables in data memory, the linker creates a data initialization template. This template must be processed at startup, before the application proper takes control. For C programs, this function is performed by the startup modules `inlibpic30-coff.a` (either `crt0.o` or `crt1.o`) or `libpic30-elf.a` (either `crt0.eo` or `crt1.eo`). Assembly language programs can utilize these modules directly by linking with the desired startup module file. The source code for the startup modules is provided in corresponding `.s` files.

The primary startup module (`crt0`) initializes all variables (variables without initializers are set to zero as required by the ANSI standard) except for variables in the persistent data section. The alternate startup module (`crt1`) performs no data initialization.

For more information on start-up code, see the “*MPLAB® XC16 Assembler, Linker and Utilities User’s Guide*” (DS52106) and for C applications, the “*MPLAB® XC16 C Compiler User’s Guide*” (DS00052071).

## 2.3 DSP Library

The DSP library (`libdsp-omf.a`) provides a set of digital signal processing operations to a program targeted for execution on a dsPIC digital signal controller (DSC). In total, 49 functions are supported by the DSP Library.

Documentation for these libraries is provided in HTML Help files. Examples of use may also be provided. By default, the documentation is found in the `docs\dsp_lib` subdirectory of the MPLAB XC16 C Compiler install directory (see “Libraries Overview, Compiler Installation Locations”).

## 2.4 16-Bit Peripheral Libraries

The 16-bit software and hardware peripheral libraries provide functions and macros for setting up and controlling 16-bit peripherals. These libraries are processor-specific and of the form `libpDevice-omf.a`, where `Device` is the 16-bit device number (e.g., `libp30F6014-coff.a` for the dsPIC30F6014 device) and `omf` is either `elf` or `coff`.

Documentation for these libraries is provided in HTML Help files. Examples of use are also provided in each file. By default, the documentation is found in the `docs\periph_lib` subdirectory of the MPLAB XC16 C Compiler install directory (see “Libraries Overview, Compiler Installation Locations”).

## 2.5 Standard C Libraries with Math and Support Functions

All MPLAB XC C compilers now use the Microchip Unified Standard Library. This library encompasses the functions defined by the standard C99 language specification headers, as well as any types and preprocessor macros needed to facilitate their use. For details, see the “*Microchip Unified Standard Library Reference Guide*” (DS-50003209).

A typical C application will require these libraries.

## 2.6 Fixed-Point Math Functions

Fixed-point math functions may be found in the library file `libq-omf.a`.

## 2.7 Compiler Built-in Functions

The MPLAB XC16 C Compiler contains built-in functions that, to the developer, work like library functions. These functions are listed in the “*MPLAB<sup>®</sup> XC16 C Compiler Users’ Guide*” (DS50002071).

### 3. Standard C and Math Libraries

Standard ANSI C and math library functions are contained in the file `libc-omf.a`, where `omf` will be `elf` or `coff` depending upon the selected object module format.

#### Assembly Code Applications

A free version of the math functions library and header file is available from the Microchip web site. No source code is available with this free version.

#### C Code Applications

The MPLAB XC16 C Compiler install directory (see “Libraries Overview, Compiler Installation Locations”) contains the following subdirectories with library-related files:

- `lib` – standard C library files
- `src\libm` – source code for math library functions, batch file to rebuild the library
- `support\h` – header files for libraries

### 3.1 Using the Standard C and Math Libraries

Building an application which utilizes the standard C and math libraries requires two types of files: header files and library files.

#### Header Files

All standard C library entities are declared or defined in one or more standard headers. To make use of a library entity in a program, write an include directive that names the relevant standard header.

The contents of a standard header is included by naming it in an include directive, as in:

```
#include <stdio.h> /* include I/O facilities */
```

The standard headers can be included in any order. Do not include a standard header within a declaration. Do not define macros that have the same names as keywords before including a standard header.

A standard header never includes another standard header.

#### Library Files

The archived library files contain all the individual object files for each library function.

When linking an application, the library file must be provided as an input to the linker (using the `--library` or `-l` linker option) such that the functions used by the application may be linked into the application.

A typical C application will require these library files: `libc-omf.a` and `libpic30-omf.a` (see “Libraries Overview, OMF-Specific Libraries and Startup Modules” for more on OMF-specific libraries). These libraries will be included automatically if linking is performed using the compiler.

**Note:** Some standard library functions require a heap. These include the standard I/O functions that open files and the memory allocation functions. See the “MPLAB<sup>®</sup> XC16 Assembler, Linker and Utilities User’s Guide” (DS52106) and “MPLAB<sup>®</sup> XC16 C Compiler User’s Guide” (DS00052071) for more information on the heap.

### 3.2 List of Standard C and Math Library Functions

For a detailed list of Standard C and math library functions, see the “Microchip Unified Standard Library Reference Guide” (DS-50003209).



## 4. Support Libraries

Support libraries contain support functions that either must be customized for correct operation of the Standard C Library in your target environment or are already customized for a Microchip target environment. The default behavior section describes what the function does, as it is distributed. The description and remarks describe what it typically should do. The corresponding object modules are distributed in the `libpic30-omf.a` archive and the source code (for the compiler) is available in the `src\pic30` folder.

For details on using the standard C libraries, see “Standard C Libraries.”

### 4.1 Rebuilding the libpic30 Library

By default, the Support Libraries helper functions were written to work with the `sim30` simulator. The header file `simio.h` defines the interface between the library and the simulator. It is provided so you can rebuild the libraries and continue to use the simulator. However, your application should not use this interface since the simulator will not be available to an embedded application.

The helper functions must be modified and rebuilt for your target application. The `libpic30-omf.a` library can be rebuilt with the batch file named `makelib.bat` which has been provided with the sources in `src\pic30`. Execute the batch file from a command window. Be sure you are in the `src\pic30` directory, then copy the newly compiled file (`libpic30-omf.a`) into the `lib` directory.

### 4.2 Standard C Library Helper Functions

These functions are called by other functions in the standard C library and must be modified for the target application. The corresponding object modules are distributed in the `libpic30-omf.a` archive and the source code (for the compiler) is available in the `src\pic30` folder.

For a detailed list of Standard C library helper functions, see the “*Microchip Unified Standard Library Reference Guide*” (DS-50003209), “Syscall Interface” chapter.

### 4.3 Standard C Library Functions That Require Modification

Although these functions are part of the Standard C Library, the object modules are distributed in the `libpic30-omf.a` archive and the source code (for the compiler) is available in the `src\pic30` folder. These modules are not distributed as part of `libc-omf.a`

- `getenv` - in `stdlib.h`
- `remove` - in `stdio.h`
- `rename` - in `stdio.h`
- `system` in `stdlib.h`
- `time` - in `time.h`

For a detailed list of Standard C library functions that require modification, see the “*Microchip Unified Standard Library Reference Guide*” (DS-50003209).

### 4.4 Functions/Constants to Support A Simulated UART

These functions and constants support UART functionality in the MPLAB SIM simulator.

#### Examples of Use

##### Example 1: UART1 I/O

```
#include <libpic30.h> /* a new header file for these definitions */
#include <stdio.h>
```

```

void main() {
    if (__attach_input_file("foo.txt")) {
        while (!feof(stdin)) {
            putchar(getchar());
        }
        __close_input_file();
    }
}

```

### Example 2: Using UART2

```

/* This program flashes a light and transmits a lot of messages at
   9600 8n1 through uart 2 using the default stdio provided by the
   16-bit compiler. This is for a dsPIC33F DSC on an Explorer 16(tm) board
   (and isn't very pretty) */
#include <libpic30.h> /* a new header file for these definitions */
#include <stdio.h>

#ifdef __dsPIC33F
#error this is a 33F demo for the explorer 16(tm) board
#endif
#include <p33Fxxxx.h>

_FOSCSEL(FNOSC_PRI );
_FOSC(FCKSM_CSDCMD & OSCIOFNC_OFF & POSCMD_XT);
_FWDT(FWDTEN_OFF);

main() {
    ODCA = 0;
    TRISAbits.TRISA6 = 0;
    C30_UART=2;
    U2BRG = 38;
    U2MODEbits.UARTEN = 1;
    while (1) {
        __builtin_btg(&LATA,6);
        printf("Hello world %d\n",U2BRG);
    }
}

```

### Example 3: Millisecond Delay

```

#define FCY 1000000UL
#include <libpic30.h>
int main() {
    /* at 1MHz, these are equivalent */
    __delay_ms(1);
    __delay32(1000);
}

```

### Example 4: Microsecond Delay

```

#define FCY 1000000UL
#include <libpic30.h>
int main() {
    /* at 1MHz, these are equivalent */
    __delay_us(1000);
    __delay32(1000);
}

```

#### 4.4.1 \_\_C30\_UART Constant

Constant that defines the default UART.

##### Include

N/A

##### Prototype

```
int __C30_UART;
```

##### Remarks

Defines the default UART that `read()` and `write()` will use for `stdin` (unless a file has been attached) and `stdout`.

#### Default Behavior

By default, or with a value of 1, UART 1 will be used. Otherwise, UART 2 will be used. `read()` and `write()` are the eventual destinations of the C standard I/O functions.

### 4.4.2 `__attach_input_file` Function

Attach a hosted file to the standard input stream.

#### Include

```
<libpic30.h>
```

#### Prototype

```
int __attach_input_file(const char *p);
```

#### Argument

<b>p</b>	pointer to file
----------	-----------------

#### Remarks

This function differs from the MPLAB X IDE mechanism of providing an input file because it provides “on-demand” access to the file. That is, data will only be read from the file upon request and the asynchronous nature of the UART is not simulated. This function may be called more than once; any opened file will be closed. It is only appropriate to call this function in a simulated environment.

#### Default Behavior

Allows the programmer to attach a hosted file to the standard input stream, `stdin`.

The function will return 0 to indicate failure. If the file cannot be opened for whatever reason, standard in will remain connected (or be re-connected) to the simulated UART.

#### File

`attach.c`

### 4.4.3 `__close_input_file` Function

Close a previously attached file.

#### Include

```
<libpic30.h>
```

#### Prototype

```
void __close_input_file(void);
```

#### Argument

None

#### Remarks

None.

#### Default Behavior

This function will close a previously attached file and re-attach `stdin` to the simulated UART. This should occur before a Reset to ensure that the file can be re-opened.

#### File

`close.c`

**4.4.4 \_\_delay32 Function**

Produce a delay of a specified number of clock cycles.

**Include**

```
<libpic30.h>
```

**Prototype**

```
void __delay32(unsigned long cycles);
```

**Argument**

<b>cycles</b>	number of cycles to delay
---------------	---------------------------

**Remarks**

None.

**Default Behavior**

This function will effect a delay of the requested number of cycles. The minimum supported delay is 12 cycles (an argument of less than or equal to 12 will result in 12 cycles). The delay includes the call and return statements, but not any cycles required to set up the argument (typically this would be two for a literal value).

**File**

delay32.s

**4.4.5 \_\_delay\_ms Function**

Produce a delay of a specified number of milliseconds (ms).

**Include**

```
<libpic30.h>
```

**Prototype**

```
void __delay_ms(unsigned int time);
```

**Argument**

<b>time</b>	number of ms to delay
-------------	-----------------------

**Remarks**

This function is implemented as a macro.

**Default Behavior**

This function relies on a user-supplied definition of FCY to represent the instruction clock frequency. FCY must be defined before header file `libpic30.h` is included. The specified delay is converted to the equivalent number of instruction cycles and passed to `__delay32()`. If FCY is not defined, then `__delay_ms()` is declared external, causing the link to fail unless the user provides a function with that name.

**File**

delay32.s

**4.4.6 \_\_delay\_us Function**

Produce a delay of a specified number of microseconds (us).

**Include**

```
<libpic30.h>
```

**Prototype**

```
void __delay_us(unsigned int time);
```

**Argument**

time	number of us to delay
------	-----------------------

**Remarks**

This function is implemented as a macro. The minimum delay is equivalent to 12 instruction cycles.

**Default Behavior**

This function relies on a user-supplied definition of FCY to represent the instruction clock frequency. FCY must be defined before header file `libpic30.h` is included. The specified delay is converted to the equivalent number of instruction cycles and passed to `__delay32()`. If FCY is not defined, then `__delay_ms()` is declared external, causing the link to fail unless the user provides a function with that name.

**File**

`delay32.s`

## 4.5 Functions for Erasing and Writing EEDATA Memory

These functions support the erasing and writing of EEDATA memory for devices that have this type of memory.

**Examples of Use****Example 1: dsPIC30F DSCs**

```
#include "libpic30.h"
#include "p30fxxxx.h"

char __attribute__((space(eedata), aligned(_EE_ROW))) dat[_EE_ROW];

int main() {
    char i, source[_EE_ROW];
    _prog_addressT p;

    for (i = 0; i < _EE_ROW; )
        source[i] = i++; /* initialize some data */

    _init_prog_address(p, dat); /* get address in program space */
    _erase_eedata(p, _EE_ROW); /* erase a row */
    _wait_eedata(); /* wait for operation to complete */
    _write_eedata_row(p, source); /* write a row */
}
```

**Example 2: PIC24FXXKA MCUs**

```
#include "libpic30.h" /* should use <> here */
#include "p24fxxxx.h"

int __attribute__((space(eedata), aligned(_EE_4WORDS))) dat[_EE_4WORDS/2];

int main() {
    _prog_addressT p;
    _init_prog_address(p, dat); /* get address in program space */
    _erase_eedata(p, _EE_4WORDS); /* erase the dat[] array */
    _wait_eedata(); /* wait to complete */
    _write_eedata_word(p, 0x1234); /* write a word to dat[0] */
    _wait_eedata();

    p += 2;
    _write_eedata_word(p, 0x5678); /* write a word to dat[1] */
    _wait_eedata();
}
```

### 4.5.1 \_erase\_eedata Function

Erase EEDATA memory on dsPIC30F and PIC24FXXKA devices.

**Include**

---

```
<libpic30.h>
```

**Prototype**

```
void _erase_eedata(_prog_addressT dst, int len);
```

**Argument**

<b>dst</b>	destination memory address
<b>len</b>	dsPIC30F: length may be _EE_WORD or _EE_ROW (bytes) PIC24FxxKA: length may be _EE_WORD, _EE_4WORDS or _EE_8WORDS (bytes)

**Return Value**

None.

**Remarks**

None.

**Default Behavior**

Erase EEDATA memory as specified by parameters.

**File**

eedata\_helper.c

#### 4.5.2 **\_erase\_eedata\_all Function**

Erase the entire range of EEDATA memory on dsPIC30F and PIC24FXXKA devices.

**Include**

```
<libpic30.h>
```

**Prototype**

```
void _erase_eedata_all(void);
```

**Argument**

None

**Return Value**

None.

**Remarks**

None.

**Default Behavior**

Erase all EEDATA memory for the selected device.

**File**

eedata\_helper.c

#### 4.5.3 **\_wait\_eedata Function**

Wait for an erase or write operation to complete on dsPIC30F and PIC24FXXKA devices.

**Include**

```
<libpic30.h>
```

**Prototype**

```
void _wait_eedata(void);
```

**Argument**

---

None

**Return Value**

None.

**Remarks**

None.

**Default Behavior**

Wait for an erase or write operation to complete.

**File**

eedata\_helper.c

**4.5.4    \_write\_eedata\_row Function**

Write \_EE\_ROW bytes of EEDATA memory on dsPIC30F devices.

**Include**

<libpic30.h>

**Prototype**

```
void _write_eedata_row(_prog_addressT dst, int *src);
```

**Arguments**

<b>dst</b>	destination memory address
<b>src</b>	points to the storage location of data to be written

**Return Value**

None.

**Remarks**

None.

**Default Behavior**

Write specified bytes of EEDATA memory.

**File**

eedata\_helper.c

**4.5.5    \_write\_eedata\_word Function**

Write 16 bits of EEDATA memory on dsPIC30F and PIC24FXXKA devices.

**Include**

<libpic30.h>

**Prototype**

```
void _write_eedata_word(_prog_addressT dst, int dat);
```

**Arguments**

<b>dst</b>	destination memory address
<b>dat</b>	integer data to be written

**Return Value**

None.

**Remarks**

None.

### Default Behavior

Write one word of EEDATA memory for dsPIC30F devices.

### File

eedata\_helper.c

## 4.6 Functions for Erasing and Writing Flash Memory

These functions support the erasing and writing of Flash memory for devices that have this type of memory.

### Examples of Use

#### Example 1: 16-Bit MCUs

```
#include "libpic30.h"
#include "p24Fxxxx.h"

int __attribute__((space(prog),aligned(_FLASH_PAGE*2))) dat[_FLASH_PAGE];

int main() {
    int i;
    int source1[_FLASH_ROW];
    long source2[_FLASH_ROW];
    _prog_addressT p;

    for (i = 0; i < _FLASH_ROW; ) {
        source1[i] = i;
        source2[i] = i++;
    } /* initialize some data */

    _init_prog_address(p, dat); /* get address in program space */
    _erase_flash(p); /* erase a page */
    _write_flash16(p, source1); /* write first row with 16-bit data */

    #if defined (__dsPIC30F__)
        _erase_flash(p); /* on dsPIC30F, only 1 row per page */
    #else
        p += (_FLASH_ROW * 2); /* advance to next row */
    #endif

    _write_flash24(p, source2); /* write second row with 24-bit data */
}
```

#### Example 2: PIC24FXXKA MCUs

```
#include "libpic30.h" /* should use <> here */
#include "p24Fxxxx.h"

int __attribute__((space(prog),aligned(_FLASH_2ROWS*2))) dat[_FLASH_2ROWS];

int main() {
    int i;
    int source1[_FLASH_ROW];
    long source2[_FLASH_ROW];
    _prog_addressT p;

    for (i = 0; i < _FLASH_ROW; ) {
        source1[i] = i;
        source2[i] = i++;
    } /* initialize some data */

    _init_prog_address(p, dat); /* get address in program space */
    _erase_flash(p, _FLASH_2ROWS); /* erase two rows */
    _write_flash16(p, source1); /* write first row with 16-bit data */

    p += (_FLASH_ROW * 2); /* advance to next row */
}
```



```

    _write_flash24(p, source2); /* write second row with 24-bit data */
}

```

#### 4.6.1 **\_erase\_flash Function**

Erase a page of Flash memory. The length of a page is `_FLASH_PAGE` words (1 word = 3 bytes = 2 PC address units).

##### **Include**

```
<libpic30.h>
```

##### **Prototype**

```
void _erase_flash(_prog_addressT dst);
```

##### **Argument**

<b>dst</b>	destination memory address
------------	----------------------------

##### **Return Value**

None.

##### **Remarks**

None.

##### **Default Behavior**

Erase a page of Flash memory.

##### **File**

flash\_helper.s

#### 4.6.2 **\_erase\_flash (PIC24FXXKA Only) Function**

Erase rows of Flash memory, either one, two or four rows.

##### **Include**

```
<libpic30.h>
```

##### **Prototype**

```
void _erase_flash(_prog_addressT dst, int len);
```

##### **Arguments**

<b>dst</b>	destination memory address
<b>len</b>	length may be <code>_FLASH_ROW</code> , <code>_FLASH_2ROWS</code> or <code>_FLASH_4ROWS</code> (bytes)

##### **Return Value**

None.

##### **Remarks**

None.

##### **Default Behavior**

Erase rows of Flash memory.

##### **File**

flash\_helper2.s

#### 4.6.3 **\_write\_flash16 Function**

Write a row of Flash memory with 16-bit data. The length of a row is `_FLASH_ROW` words. The upper byte of each destination word is filled with 0xFF. Note that the row must be erased before any write can be successful.

---

**Include**

```
<libpic30.h>
```

**Prototype**

```
void _write_flash16(_prog_addressT dst, int *src);
```

**Arguments**

<b>dst</b>	destination memory address
<b>src</b>	points to the storage location of data to be written

**Return Value**

None.

**Remarks**

None.

**Default Behavior**

Write a row of Flash memory with 16-bit data.

**File**

```
flash_helper.c
```

#### 4.6.4 **\_write\_flash24 Function**

Write a row of Flash memory with 24-bit data. The length of a row is \_FLASH\_ROW words. Note that the row must be erased before any write can be successful.

**Include**

```
<libpic30.h>
```

**Prototype**

```
void _write_flash24(_prog_addressT dst, int *src);
```

**Arguments**

<b>dst</b>	destination memory address
<b>src</b>	points to the storage location of data to be written

**Return Value**

None.

**Remarks**

None.

**Default Behavior**

Write a row of Flash memory with 24-bit data.

**File**

```
flash_helper.c
```

#### 4.6.5 **\_write\_flash\_word16 Function**

Write a word of Flash memory with 16-bit data. The upper byte of the destination word is filled with 0xFF. Note that the word must be erased before any write can be successful. This function is currently available only for PIC24F devices (excluding PIC24FXXKA MCUs).

**Include**

```
<libpic30.h>
```

**Prototype**

```
void _write_flash_word16(_prog_addressT dst, int dat);
```

**Arguments**

<b>dst</b>	destination memory address
<b>dat</b>	integer data to be written

**Return Value**

None.

**Remarks**

None.

**Default Behavior**

Write a word of Flash memory with 16-bit data for most PIC24 devices.

**File**

flash\_helper.c

**4.6.6 \_write\_flash\_word24 Function**

Write a word of Flash memory with 24-bit data. Note that the word must be erased before any write can be successful. This function is currently available only for PIC24F devices (excluding PIC24FXXKA MCUs).

**Include**

```
<libpic30.h>
```

**Prototype**

```
void _write_flash_word24(_prog_addressT dst, long dat);
```

**Arguments**

<b>dst</b>	destination memory address
<b>dat</b>	integer data to be written

**Return Value**

None.

**Remarks**

None.

**Default Behavior**

Write a word of Flash memory with 24-bit data for most PIC24 devices.

**File**

flash\_helper.c

**4.6.7 \_write\_flash\_word32 Function**

Write two words of FLASH memory with 16 bits of data per word. The 16 bits are written to the low 16 bits of the word. Word writes are supported dsPIC33E and PIC24E devices. The row address is specified with type `_prog_addressT`. Note that the location must be erased before any write can be successful.

This function is currently disabled for devices subject to the Device ID errata as described in DS-80444, DS-80446, or DS-80447 (#32).

**Include**

```
<libpic30.h>
```

**Prototype**

```
void _write_flash_word32(_prog_addressT dst, int dat1, int dat2);
```

**Arguments**

<b>dst</b>	destination memory address
<b>dat1, dat2</b>	integer data to be written

**Return Value**

None.

**Remarks**

None.

**Default Behavior**

Write two words of Flash memory with 16-bit data for most dsPIC33E/PIC24E devices.

**File**

flash\_helper.c

**4.6.8 \_write\_flash\_word48 Function**

Write two words of FLASH memory with 24 bits of data per word. Word writes are supported 33E and 24E devices. The row address is specified with type `_prog_addressT`. Note that the location must be erased before any write can be successful.

This function is currently disabled for devices subject to the Device ID errata as described in DS-80444, DS-80446, or DS-80447 (#32).

**Include**

```
<libpic30.h>
```

**Prototype**

```
void _write_flash_word48(_prog_addressT dst, int dat1, int dat2);
```

**Arguments**

<b>dst</b>	destination memory address
<b>dat1, dat2</b>	integer data to be written

**Return Value**

None.

**Remarks**

None.

**Default Behavior**

Write two words of Flash memory with 48-bit data for most dsPIC33E/PIC24E devices.

**File**

flash\_helper.c

**4.7 Functions for Specialized Copying and Initialization**

These functions support specialized data copying and initialization.

**Example of Use**

```
#include "stdio.h"
#include "libpic30.h"

void display_mem(char *p, unsigned int len) {
    int i;
    for (i = 0; i < len; i++) {
        printf(" %d", *p++);
    }
    printf("\n");
}

char __attribute__((space(prog))) dat[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
char buf[10];

int main() {
    int i;
    _prog_addressT p;

    /* method 1 */
    _init_prog_address(p, dat);
    (void) _memcpy_p2d16(buf, p, 10);
    display_mem(buf, 10);

    /* method 2 */
    _init_prog_address(p, dat);
    p = _memcpy_p2d16(buf, p, 4);
    p = _memcpy_p2d16(&buf[4], p, 6);
    display_mem(buf, 10);
}
```

**4.7.1 \_init\_prog\_address Macro**

A macro that is used to initialize variables of type, `_prog_addressT`. These variables are not equivalent to C pointers.

**Include**

<libpic30.h>

**Prototype**

`_init_prog_address(a, b);`

**Arguments**

- |          |  |
|----------|--|
| <b>a</b> | variable of type <code>_prog_addressT</code> |
| <b>b</b> | initialization value for variable <i>a</i>   |

**Return Value**

N/A

**Remarks**

None.

**Default Behavior**

Initialize variable to specified value.

**File**

libpic30.c

**4.7.2 \_memcpy\_eds Function**

Copies bytes from one `__eds__` buffer to another `__eds__` buffer.

**Include**

<libpic30.h>

**Prototype**

```
__eds__ void* _memcpy_eds(__eds__ void *dest, __eds__ void *src, unsigned int len);
```

**Arguments**

<b>dest</b>	destination memory address
<b>src</b>	address of data to be written
<b>len</b>	length of program memory

**Return Value**

The memory block pointed to by `dest`.

**Remarks**

`__eds__` pointers are superset of unqualified data pointers; therefore these functions can be used to copy between `__eds__` and unqualified data memory.

**Default Behavior**

Copy `len` bytes of data from each address pointed to by the `src` pointer to the destination pointed to by the `dest` pointer.

**File**

`memcpy_helper.s`

**4.7.3 \_memcpy\_p2d16 Function**

Copy 16 bits of data from each address in program memory to data memory. The next unused source address is returned.

**Include**

```
<libpic30.h>
```

**Prototype**

```
_prog_addressT _memcpy_p2d16(char *dest, _prog_addressT src, unsigned int len);
```

**Arguments**

<b>dest</b>	destination memory address
<b>src</b>	address of data to be written
<b>len</b>	length of program memory

**Return Value**

The next unused source address.

**Remarks**

None.

**Default Behavior**

Copy 16 bits of data from each of the `len` bytes of addresses of `src` to the destination pointed to by the `dest` pointer.

**File**

`memcpy_helper.s`

**4.7.4 \_memcpy\_p2d24 Function**

Copy 24 bits of data from each address in program memory to data memory. The next unused source address is returned.

**Include**

```
<libpic30.h>
```

**Prototype**

```
_prog_addressT _memcpy_p2d24(char *dest, _prog_addressT src, unsigned int len);
```

**Arguments**

<b>dest</b>	destination memory address
<b>src</b>	address of data to be written
<b>len</b>	length of program memory

**Return Value**

The next unused source address.

**Remarks**

None.

**Default Behavior**

Copy 24 bits of data from each of the `len` bytes of addresses of `src` to the destination pointed to by the `dest` pointer.

**File**

`memcpy_helper.s`

**4.7.5 \_memcpy\_packed Function**

Copies bytes from one `__pack_upper_byte` Flash buffer to another buffer in RAM.

**Include**

```
<libpic30.h>
```

**Prototype**

```
void* _memcpy_packed(void *dest, __pack_upper_byte void *src, unsigned int len);
```

**Arguments**

<b>dest</b>	destination memory address
<b>src</b>	address of data to be written
<b>len</b>	length of program memory

**Return Value**

The memory block pointed to by `dest`.

**Remarks**

None.

**Default Behavior**

Copy `len` bytes of data from each address pointed to by the `src` pointer to the destination pointed to by the `dest` pointer.

**File**

`memcpy_helper.s`

**4.7.6 \_strcpy\_eds Function**

Copies a string from one `__eds__` buffer to another `__eds__` buffer.

**Include**

```
<libpic30.h>
```

**Prototype**

```
__eds__ char* _strcpy_eds(__eds__ char *dest, __eds__ char *src);
```

**Arguments**

<b>dest</b>	destination memory address
<b>src</b>	address of data to be written

**Return Value**

The string pointed to by `dest`.

**Remarks**

`__eds__` pointers are superset of unqualified data pointers; therefore these functions can be used to copy between `__eds__` and unqualified data memory.

**Default Behavior**

Copy the string `src` pointer to the destination pointed to by the `dest` pointer.

**File**

`memcpy_helper.s`

**4.7.7 \_strcpy\_packed Function**

Copies a string from one `__pack_upper_byte` Flash buffer to another buffer in RAM.

**Include**

```
<libpic30.h>
```

**Prototype**

```
void* _strcpy_packed( void *dest, __pack_upper_byte void *src);
```

**Arguments**

<b>dest</b>	destination memory address
<b>src</b>	address of data to be written

**Return Value**

The string pointed to by `dest`.

**Remarks**

None.

**Default Behavior**

Copy the string pointed to by `src` to the RAM buffer pointed to by `dest`. Unlike the standard `strcpy` function, this function does not zero fill the remaining space in the destination string.

**File**

`memcpy_helper.s`

**4.7.8 \_strncpy\_eds Function**

Copies at most `len` bytes string from one `__eds__` buffer to another `__eds__` buffer.

**Include**



---

```
<libpic30.h>
```

**Prototype**

```
__eds__ char* _strncpy_eds(__eds__ char *dest, __eds__ char *src, unsigned int len);
```

**Arguments**

<b>dest</b>	destination memory address
<b>src</b>	address of data to be written
<b>len</b>	length of program memory

**Return Value**

The string pointed to by *dest*.

**Remarks**

`__eds__` pointers are superset of unqualified data pointers; therefore these functions can be used to copy between `__eds__` and unqualified data memory.

**Default Behavior**

Copy *len* bytes of the string pointed to by the *src* pointer to the destination pointed to by the *dest* pointer. Unlike the standard `strncpy` function, this function does not zero fill the remaining space in the destination string.

**File**

`memcpy_helper.s`

**4.7.9 \_strncpy\_p2d16 Function**

Copy 16 bits of data from each address in program memory to data memory. The operation terminates early if a NULL char is copied. The next unused source address is returned.

**Include**

```
<libpic30.h>
```

**Prototype**

```
_prog_addressT _strncpy_p2d16(char *dest, _prog_addressT src, unsigned int len);
```

**Arguments**

<b>dest</b>	destination memory address
<b>src</b>	address of data to be written
<b>len</b>	length of program memory

**Return Value**

The next unused source address.

**Remarks**

None.

**Default Behavior**

Copy 16 bits of data from each of the *len* bytes of addresses of *src* to the destination pointed to by the *dest* pointer.

**File**

`memcpy_helper.s`

#### 4.7.10 **\_strncpy\_p2d24 Function**

Copy 24 bits of data from each address in program memory to data memory. The operation terminates early if a NULL char is copied. The next unused source address is returned.

##### **Include**

```
<libpic30.h>
```

##### **Prototype**

```
_prog_addressT _strncpy_p2d24(char *dest, _prog_addressT src, unsigned int len);
```

##### **Arguments**

<b>dest</b>	destination memory address
<b>src</b>	address of data to be written
<b>len</b>	length of program memory

##### **Return Value**

The next unused source address.

##### **Remarks**

None.

##### **Default Behavior**

Copy 24 bits of data from each of the `len` bytes of addresses of `src` to the destination pointed to by the `dest` pointer.

##### **File**

`memcpy_helper.s`

#### 4.7.11 **\_strncpy\_packed Function**

Copies at most `len` bytes of a string from one `__pack_upper_byte` Flash buffer to another buffer in RAM.

##### **Include**

```
<libpic30.h>
```

##### **Prototype**

```
void* _strncpy_packed( void *dest, __pack_upper_byte void *src, int len);
```

##### **Arguments**

<b>dest</b>	destination memory address
<b>src</b>	address of data to be written
<b>len</b>	length of program memory

##### **Return Value**

The string pointed to by `dest`.

##### **Remarks**

None.

##### **Default Behavior**

Copy at most `len` bytes of the string pointed to by `src` to the RAM buffer pointed to by `dest`. Unlike the standard `strncpy` function, this function does not zero fill the remaining space in the destination string.

##### **File**

`memcpy_helper.s`



## 4.8 Functions to Support Secondary Core PRAM

These utility functions program the Secondary core with the specified image which has been generated by an MPLAB X IDE project. These functions will not work unless using the XC16 model of programming. Outside of this paradigm, refer to the device programming specification.

### Examples of Use

#### Example 1: Secondary PRAM Load and Verify Routine

```
#include <libpic30.h>
//wipe memory before programming
_wipe_secondary(&secondary_image)

//_program_secondary(core#, verify, &secondary_image)
if (_program_secondary(1, 0, &secondary_image) == 0)
{
    /* now verify */
    if (_program_secondary(1, 1, &secondary_image) == ESEC_VERIFY_FAIL)
    {
        asm("reset") ; // try again
    }
}
```

#### Example 2: Secondary Core Start and Stop

```
#include <libpic30.h>
int main()
{
    // Main initialization code
    _start_secondary(); // Start Secondary core

    // Main application code
    _stop_secondary(); // Stop Secondary core

    while(1);
}
```

### Reference Documents:

[ww1.microchip.com/downloads/en/DeviceDoc/dsPIC33CH128MP508-Family-Data-Sheet-DS70005319D.pdf](http://ww1.microchip.com/downloads/en/DeviceDoc/dsPIC33CH128MP508-Family-Data-Sheet-DS70005319D.pdf) (See Section 4.3)

[ww1.microchip.com/downloads/en/DeviceDoc/dsPIC33CH512MP508-Family-Data-Sheet-DS70005371D.pdf](http://ww1.microchip.com/downloads/en/DeviceDoc/dsPIC33CH512MP508-Family-Data-Sheet-DS70005371D.pdf) (See Section 4.3)

### 4.8.1 \_wipe\_secondary Function

Erases the Secondary core to the erase state.

#### Include

```
<libpic30.h>
```

#### Prototype

```
int _wipe_secondary(int secondary_number)
```

#### Arguments

<b>secondary_number</b>	Identifier of the Secondary core to be programmed. The identifier for the first Secondary core is 1.
-------------------------	--

#### Return Value

- ESLV\_INVALID - returned if an invalid secondary number is given

#### Remarks

None.

**Default Behaviour**

The `_wipe_secondary` routine can be used to completely wipe the PRAM memory of the Secondary core. This routine will always wipe the entire memory range, even if dual partition mode is selected.

**File**

`wipe_secondary.c` (in `libpic30.zip>data_init_dualch.S`)

**4.8.2 \_program\_secondary Function**

Programs the Secondary core with the specified image.

**Include**

`<libpic30.h>`

**Prototype**

```
int _program_secondary(int secondary_number, int verify, __eds__ unsigned char *image)
```

**Arguments**

<b>secondary_number</b>	Identifier of the Secondary core to be programmed. The identifier for the first Secondary core is 1.
<b>verify</b>	A 0 will load the entire Secondary image to the PRAM. A 1 will verify the entire image in the PRAM.
<b>*image</b>	Pointer to the image to be programmed into PRAM. Secondary core PRAM images not following the Microchip language tool format will require a custom routine.

**Return Value**

- `ESLV_INVALID` - returned if an invalid Secondary core number is given
- `ESLV_BAD_IMAGE` - returned if there is an error decoding the Secondary core image
- `ESLV_VERIFY_FAIL` - returned if 'verify' fails

**Remarks**

None.

**Default Behavior**

The `_program_secondary` routine uses the `verify` parameter as a switch to either load or verify the Secondary core PRAM image using the `LDSEC` or `VFSEC` instructions.

**File**

`program_secondary.c` (in `libpic30.zip>data_init_dualch.S`)

**4.8.3 \_program\_inactive\_secondary Function**

Programs the inactive partition PRAM with the specified image.

**Include**

`<libpic30.h>`

**Prototype**

```
int _program_inactive_secondary(int secondary_number, int verify, __eds__ unsigned char *image)
```

**Arguments**

<b>secondary_number</b>	Identifier of the Secondary core to be programmed. The identifier for the first Secondary core is 1.
-------------------------	--

---



---

<b>verify</b>	A 0 will load the entire Secondary image to the PRAM. A 1 will verify the entire image in the PRAM.
<b>*image</b>	Pointer to the image to be programmed into PRAM. Secondary PRAM images not following the Microchip language tool format will require a custom routine.

**Return Value**

- ESLV\_INVALID - returned if an invalid Secondary core number is given
- ESLV\_BAD\_IMAGE - returned if there is an error decoding the Secondary core image
- ESLV\_VERIFY\_FAIL - returned if 'verify' fails

**Remarks**

The Main core loads the PRAM inactive partition while the Secondary core is running and then the Secondary core executes the `BOOTSWP` instruction to swap partitions.

**Default Behavior**

The `_program_inactive_secondary` routine uses the **verify** parameter as a switch to either load or verify the inactive partition image using the `LDSEC` or `VFSEC` instructions.

**File**

`program_inactive_secondary.c` (in `libpic30.zip>data_init_dualch.S`)

**4.8.4 \_start\_secondary and \_stop\_secondary Functions**

Start or stop the Secondary core after the image has been loaded by the Main core.

**Include**

`<libpic30.h>`

**Prototypes**

`void _start_secondary(void)`

`void _stop_secondary(void)`

**Arguments**

None.

**Return Value**

None.

**Remarks**

None.

**Default Behavior**

The `_start_secondary` and `_stop_secondary` routines perform the MNI1KEY unlock sequence and set or clear the SECEN bit (MNI1CON[15]) respectively.

**Files**

`start_secondary.c` (in `libpic30.zip>data_init_dualch.S`)

`stop_secondary.c` (in `libpic30.zip>data_init_dualch.S`)

## 5. Fixed-Point Math Functions

Fixed-point library math functions are contained in the files `libq-omf.a` (standard) and `libq-dsp-omf.a` (DSP), where `omf` will be `elf` or `coff` depending upon the selected object module format. The header file is named `libq.h` and is the same for standard or DSP versions of the library. Linker options `-lq` (standard and DSP) and `-lq-dsp` (DSP only) must be used when linking the respective libraries.

### 5.1 Overview of Fixed-Point Data Formats

The integer data is encoded as its two's complement to accommodate both positive and negative numbers in binary format. The two's complement can be represented using integer format or the fractional format.

#### Integer Format

The integer format data is represented as a signed two's complement value, where the Most Significant bit is defined as a sign bit. The range of an N-bit two's complement integer is  $-2^{N-1}$  to  $2^{N-1}-1$  with a resolution of 1. For a 16-bit integer, the data range is -32768 (0x8000) to +32767 (0x7FFF) with a resolution of 1. For a 32-bit integer, the data range is -2,147,483,648 (0x8000 0000) to +2,147,483,647 (0x7FFF FFFF) with a resolution of 1.

#### Fractional Format

The fractional data format (Qn.m) has integral part (n) and fractional part (m) and the Most Significant bit represents the sign, thus consisting of (m+n+1) bits. It represents a signed two's complement value. Qn.m format data has a range of  $[-2^n, (2^n-2^{-m})]$  with  $2^{-m}$  resolution.

The binary representation of an N-bit (m+n+1 bits) number in Qn.m is shown in Figure 1. The value is given by the equation shown in Figure 2.

Figure 1: Binary Representation

$$\underbrace{b_{m+n} \ b_{m+n-1} \ \dots \ b_m}_{N-1} . b_{m-1} \ \dots \ b_1 \ b_0$$

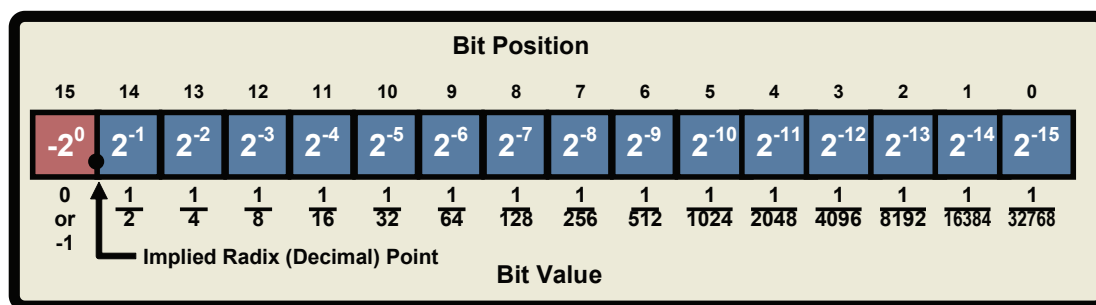
Figure 2: Equation Value

$$Value = -b_{N-1}2^n + \sum_{l=0}^{N-2} b_l 2^{l-m}$$

#### Q15 (1.15) Format




In Q15 format, the Most Significant bit is defined as a sign bit and the radix point is implied to lie just after the sign bit followed by the fractional value. This format is commonly referred to as 1.15 or Q15 format, where 1 is the number of bits used to represent the integer portion of the number, and 15 is the number of bits used to represent the fractional portion. The range of an N-bit two's complement fraction with this implied radix point is -1.0 to  $(1 - 2^{1-N})$ . For a 16-bit fraction, the 1.15 data range is -1.0 (0x8000) to +0.999969482 (0x7FFF) with a precision of  $3.05176 \times 10^{-5}$ .

Figure 3: Fractional Format (16 bits)



The following table shows the conversion of a two's complement 16-bit integer +24576 to Q15 value +0.75.

**Table 5-1. Table1: Conversion of a Two's Complement 16-Bit Integer to Q15**

Binary		Dec		Q15
0	$0 \times (-2^{15})$	0	$0 \times (-2^0)$	0
				
1	$1 \times 2^{14}$	16384	$1 \times 2^{-1}$	0.5
1	$1 \times 2^{13}$	8192	$1 \times 2^{-2}$	0.25
0	$0 \times 2^{12}$	0	$0 \times 2^{-3}$	0
0	$0 \times 2^{11}$	0	$0 \times 2^{-4}$	0
0	$0 \times 2^{10}$	0	$0 \times 2^{-5}$	0
0	$0 \times 2^9$	0	$0 \times 2^{-6}$	0
0	$0 \times 2^8$	0	$0 \times 2^{-7}$	0
0	$0 \times 2^7$	0	$0 \times 2^{-8}$	0
0	$0 \times 2^6$	0	$0 \times 2^{-9}$	0
0	$0 \times 2^5$	0	$0 \times 2^{-10}$	0
0	$0 \times 2^4$	0	$0 \times 2^{-11}$	0
0	$0 \times 2^3$	0	$0 \times 2^{-12}$	0
0	$0 \times 2^2$	0	$0 \times 2^{-13}$	0
0	$0 \times 2^1$	0	$0 \times 2^{-14}$	0
0	$0 \times 2^0$	0	$0 \times 2^{-15}$	0
	SUM	+24576	SUM	+0.75
 = Radix Point				

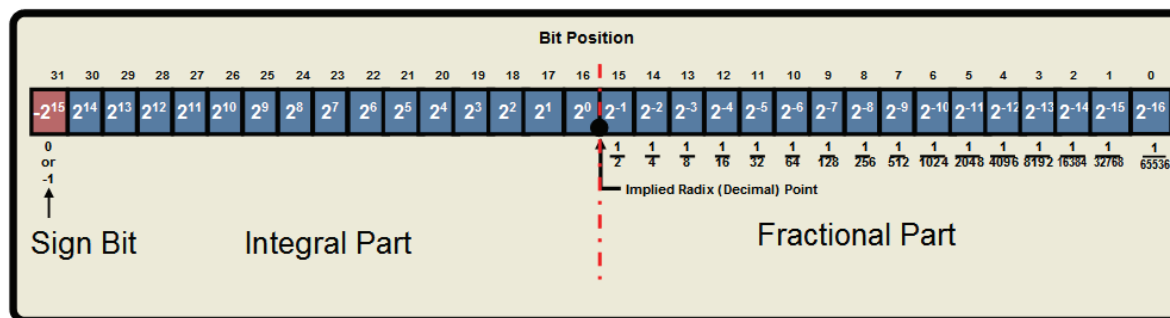
#### Q15.16 Format

In the Q15.16 format, the Most Significant bit is defined as a sign bit followed by 16 bits of the integral part. The radix point is implied to lie just after the integral part, followed by 16 bits of the fractional value. This format



is commonly referred to as Q15.16 format. The range of Q15.16 numbers is from -32768.0 (0x8000 0000) to +32767.9999847412109375 (0x7FFF FFFF) and has a precision of 2<sup>-16</sup>.

**Figure 4: Fractional Format (32 bits)**



The following table shows the conversion of a two's complement 32-bit integer, -715827882 to Q15.16 value -10922.6666564941.

**Table 5-2. Table 2: Conversion of a Two's Complement 32-Bit Integer to Q15.16**

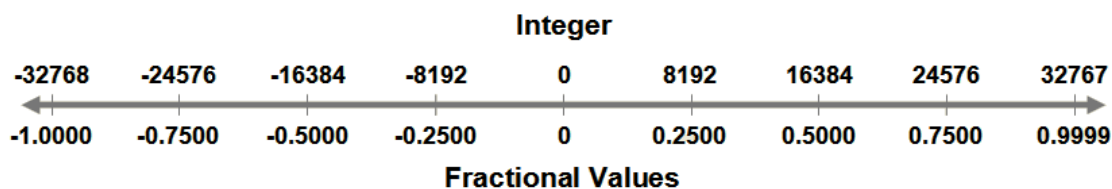
Binary		Dec		Q15.16
1	$1 \times (-2^{31})$	-2147483648	$1 \times (-2^{15})$	-32768
1	$1 \times 2^{30}$	1073741824	$1 \times 2^{14}$	16384
0	$0 \times 2^{29}$	0	$0 \times 2^{13}$	0
1	$1 \times 2^{28}$	268435456	$1 \times 2^{12}$	4096
0	$0 \times 2^{27}$	0	$0 \times 2^{11}$	0
1	$1 \times 2^{26}$	67108864	$1 \times 2^{10}$	1024
0	$0 \times 2^{25}$	0	$0 \times 2^9$	0
1	$1 \times 2^{24}$	16777216	$1 \times 2^8$	256
0	$0 \times 2^{23}$	0	$0 \times 2^7$	0
1	$1 \times 2^{22}$	4194304	$1 \times 2^6$	64
0	$0 \times 2^{21}$	0	$0 \times 2^5$	0
1	$1 \times 2^{20}$	1048576	$1 \times 2^4$	16
0	$0 \times 2^{19}$	0	$0 \times 2^3$	0
1	$1 \times 2^{18}$	262144	$1 \times 2^2$	4
0	$0 \times 2^{17}$	0	$0 \times 2^1$	0
1	$1 \times 2^{16}$	65536	$1 \times 2^0$	1
0	$0 \times 2^{15}$	0	$0 \times 2^{-1}$	0
1	$1 \times 2^{14}$	16384	$1 \times 2^{-2}$	0.25
0	$0 \times 2^{13}$	0	$0 \times 2^{-3}$	0
1	$0 \times 2^{12}$	4096	$1 \times 2^{-4}$	0.0625
0	$0 \times 2^{11}$	0	$0 \times 2^{-5}$	0
1	$1 \times 2^{10}$	1024	$1 \times 2^{-6}$	0.015625

.....continued				
Binary		Dec		Q15.16
0	$0 \times 2^9$	0	$0 \times 2^{-7}$	0
1	$1 \times 2^8$	256	$1 \times 2^{-8}$	0.00390625
0	$0 \times 2^7$	0	$0 \times 2^{-9}$	0
1	$1 \times 2^6$	64	$1 \times 2^{-10}$	0.000976563
0	$0 \times 2^5$	0	$0 \times 2^{-11}$	0
1	$1 \times 2^4$	16	$1 \times 2^{-12}$	0.000244141
0	$0 \times 2^3$	0	$0 \times 2^{-13}$	0
1	$1 \times 2^2$	4	$1 \times 2^{-14}$	6.10352E-05
1	$1 \times 2^1$	2	$1 \times 2^{-15}$	3.01576E-05
0	$0 \times 2^0$	0	$0 \times 2^{-16}$	0
	SUM	-715827882	SUM	-10922.6666564941
● = Radix Point				

### Integer - Fractional Format Mapping

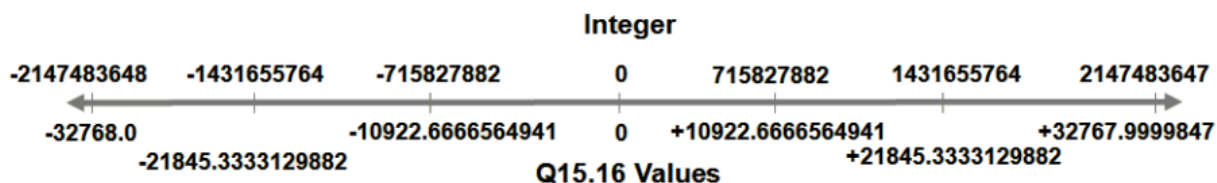
The same two's complement N-bit word may represent an integer format value or a fractional format value. For example., the 16-bit integer range [-32768, +32767] format maps to Q15 range of [-1.0, +0.999969482]. Figure 5 shows the mapping between these formats.

Figure 5: Mapping between 16-bit integer format and Q15 fractional format



A similar relationship exists between the 32-bit integer format and the Q15.16 format, where the integer range [-2147483648, +2147483647] is mapped to the Q15.16 range [-32768.0, +32767.9999847412109375].

Figure 6: Mapping between 32-bit integer format and Q15.16 format



### libq Library and Fixed-Point Data Format

The functions in the `libq` library use the fixed-point data format. The parameters passed and the results generated by the functions are fractional in nature. There are two similar sets of math functions which perform the same math operations. One set supports Q15 operands and the other supports Q15.16 operands. Q15.16 operand functions, naturally, have better precision and range compared to Q15 operand functions.

## 5.2 Using the Fixed-Point Libraries

Building an application which utilizes the fixed-point libraries requires two types of files: header files and library files. Understanding fixed-point function naming conventions is needed for use.

### Header Files

All standard C library entities are declared or defined in one or more standard headers. To make use of a library entity in a program, write an include directive that names the relevant standard header. The contents of a standard header are included by naming them in an include directive, as in:

```
#include <libq.h> /* include fixed-point library */
```

The standard headers can be included in any order. Do not include a standard header within a declaration. Do not define macros that have the same names as keywords before including a standard header.

A standard header never includes another standard header.

### Library Files

The archived library files contain all the individual object files for each library function.

When linking an application, the library file (`libq-omf.a` or `libq-dsp-omf.a`) must be provided as an input to the linker (using the `--library` or `-l` linker option), such that the functions used by the application may be linked into the application. Also, linker options `-lq` and `-lq-dsp` must be used when linking the respective libraries.

A typical C application will require three library files: `libc-omf.a`, `libm-omf.a` and `libpic30-omf.a` (see “OMF-Specific Libraries and Startup Modules” for more on OMF-specific libraries). These libraries will be included automatically if linking is performed using the compiler.

### Function Naming Conventions

Signed fixed-point types are defined as follows:

`Qn_m`

where:

- `n` is the number of data bits to the left of the radix point
- `m` is the number of data bits to the right of the radix point

**Note:** A sign bit is implied.

For convenience, short names are also defined:

Exact Name	# Bits Required	Short Name
<code>_Q0_15</code>	16	<code>_Q15</code>
<code>_Q15_16</code>	32	<code>_Q16</code>

In this document, the terms `Q15.16` and `Q16` are used interchangeably; however, both imply `Q15.16` format. Functions in the library are prefixed with the type of the return value. For example, `_Q15acos` returns a `Q15` value equal to the arc cosine of its argument.

Argument types do not always match the return type. Refer to the function prototype for a specification of its arguments. In cases where the return value is not a fixed-point type, the argument type is appended to the function name. For example, function `_itoaQ15` accepts a type `Q15` argument.

In cases where two versions of a function are provided with the same return type but different argument types, the argument type is appended to the function name. For example:

Function Name	Return Type	Argument Type
<code>_Q16reciprocalQ15</code>	<code>_Q16</code>	<code>_Q15</code>
<code>_Q16reciprocalQ16</code>	<code>_Q16</code>	<code>_Q16</code>

## 5.3 <libq.h> Mathematical Functions

The header file libq.h consists of macro definitions and various functions that calculate fixed-point mathematical operations.

**Q15 Functions:** Many of these functions use fixed-point Q15 (1.15) format. For each function, the entire range may not be used.

**Q16 Functions:** Many of these functions use fixed-point Q16 (15.16) format. For each function, the entire range may not be used.

### 5.3.1 \_Q15abs Function

The function finds the absolute value of a Q15 value.

#### Include

<libq.h>

#### Prototype

```
_Q15 _Q15abs(_Q15 x);
```

#### Argument

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.
----------	---

#### Return Value

This function returns the absolute value of  $x$  in Q15 format. The value ranges from 0 to 32767.

**Note:** `_Q15abs(-32768) = 32767`.

Also `abs` (smallest negative number) = largest positive number.

### 5.3.2 \_Q15acos Function

This function finds the arc cosine of a Q15 value.

#### Include

<libq.h>

#### Prototype

```
_Q15 _Q15acos(_Q15 x);
```

#### Argument

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . Therefore the value of this argument ranges from 17705 to 32767.
----------	--

#### Return Value

This function returns the arc cosine of  $x$  in Q15 format. The value ranges from 256 to 32767.

### 5.3.3 \_Q15acosByPI Function

This function finds the arc cosine of a Q15 value and then divides by PI ( $\pi$ ).

#### Include

<libq.h>

#### Prototype

```
_Q15 _Q15acosByPI(_Q15 x);
```

#### Argument

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.
----------	---

**Return Value**

This function returns the arc cosine of  $x$ , divided by PI, in Q15 format. The value ranges from 82 to 32767.

**5.3.4 \_Q15add Function**

This function finds the sum value of two Q15 values. The function takes care of saturation during overflow and underflow occurrences.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15add(_Q15 x, _Q15 y);
```

**Argument**

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.
<b>y</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.

**Return Value**

This function returns the sum of  $x$  and  $y$  in Q15 format. The value ranges from -32768 to 32767.

**5.3.5 \_Q15asin Function**

This function finds the arc sine of a Q15 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15asin(_Q15 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.
----------	---

**Return Value**

This function returns the arc sine of  $x$  in Q15 format. The value ranges from -32768 to 32767.

**5.3.6 \_Q15asinByPI Function**

This function finds the arc sine of a Q15 value and then divides by PI ( $\pi$ ).

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15asinByPI(_Q15 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.
----------	---

**Return Value**

This function returns the arc sine of  $x$ , divided by PI, in Q15 format. The value ranges from -16384 to 16303.

**5.3.7 \_Q15atan Function**

This function finds the arc tangent of a Q15 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15atan(_Q15 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.
----------	---

**Return Value**

This function returns the arc tangent of  $x$  in Q15 format. The value ranges from -25736 to 25735.

**5.3.8 \_Q15atanByPI Function**

This function finds the arc tangent of a Q15 value and then divides by PI ( $\pi$ ).

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15atanByPI(_Q15 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.
----------	---

**Return Value**

This function returns the arc tangent of  $x$ , divided by PI, in Q15 format. The value ranges from -8192 to 8192.

**5.3.9 \_Q15atanYByX Function**

This function finds the arc tangent of a Q15 value divided by a second Q15 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15atanYByX(_Q15 x, _Q15 y);
```

**Arguments**

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.
<b>y</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.

**Return Value**

This function returns the arc tangent of  $y$  divided by  $x$  in Q15 format. The value ranges from -25736 to 25735.

**5.3.10 \_Q15atanYByXByPI Function**

This function finds the arc tangent of a Q15 value divided by a second Q15 value and then divides the result by PI ( $\pi$ ).

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15atanYByXByPI(_Q15 x, _Q15 y);
```

**Arguments**

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.
<b>y</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.

**Return Value**

This function returns the arc tangent of y divided by x, divided by PI, in Q15 format. The value ranges from -8192 to 8192.

**5.3.11 \_Q15atoi Function**

This function takes a string which holds the ASCII representation of decimal digits and converts it into a single Q15 number.

Note: The decimal digit should not be beyond the range: -32768 to 32767.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15atoi(const char *s);
```

**Argument**

<b>s</b>	a buffer holding the ASCII values of each decimal digit.
----------	--

**Return Value**

This function returns the integer equivalent of s in Q15 format, which range is from -32768 to 32767.

**5.3.12 \_Q15cos Function**

This function finds the cosine of a Q15 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15cos(_Q15 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.
----------	---

**Return Value**

This function returns the cosine of x in Q15 format. The value ranges from 17705 to 32767.

**5.3.13 \_Q15cosPI Function**

This function finds the cosine of PI ( $\pi$ ) times a Q15 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15cosPI(_Q15 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.
----------	---

**Return Value**

This function returns the cosine of PI times  $x$  in Q15 format. The value ranges from -32768 to 32767.

**5.3.14 \_Q15exp Function**

This function finds the exponential value of a Q15 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15exp(_Q15 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 0.
----------	---

**Return Value**

This function returns the exponent value of  $x$  in Q15 format. The value ranges from 12055 to 32767.

**5.3.15 \_Q15ftoi Function**

This function converts a single precision floating-point value into its corresponding Q15 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15ftoi(float x);
```

**Argument**

<b>x</b>	a floating-point equivalent number. The corresponding floating-point range is -1 to 0.99996.
----------	--

**Return Value**

This function returns a fixed-point number in Q15 format. The value ranges from -32768 to 32767.

**5.3.16 \_itoaQ15 Function**

This function converts each decimal digit of a Q15 value to its representation in ASCII. For example, 1 is converted to 0x31, which is the ASCII representation of 1.

**Include**

```
<libq.h>
```

**Prototype**



---

```
void _itoaQ15(_Q15 x, char *s);
```

**Arguments**

- |          |   |
|----------|---|
| <b>x</b> | a fixed-point number in Q15 format, which ranges from -2 <sup>15</sup> to 2 <sup>15</sup> -1. The value of this argument ranges from -32768 to 32767. |
| <b>s</b> | a buffer holding values in ASCII, at least 8 characters long.   |

**Return Value**

None.

**5.3.17 \_itofQ15 Function**

This function converts a Q15 value into its corresponding floating-point value.

**Include**

```
<libq.h>
```

**Prototype**

```
float _itofQ15(_Q15 x);
```

**Argument**

- |          |   |
|----------|---|
| <b>x</b> | a fixed-point number in Q15 format, which ranges from -2 <sup>15</sup> to 2 <sup>15</sup> -1. The value of this argument ranges from -32768 to 32767. |
|----------|---|

**Return Value**

This function returns a floating-point equivalent number. The corresponding floating-point range is -1 to 0.99996.

**5.3.18 \_Q15log Function**

This function finds the natural log of a Q15 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15log(_Q15 x);
```

**Argument**

- |          |  |
|----------|--|
| <b>x</b> | a fixed-point number in Q15 format, which ranges from -2 <sup>15</sup> to 2 <sup>15</sup> -1. The value of this argument ranges from 12055 to 32767. |
|----------|--|

**Return Value**

This function returns the natural log of x in Q15 format. The value ranges from -32768 to -1.

**5.3.19 \_Q15log10 Function**

This function finds the log (base 10) of a Q15 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15log10(_Q15 x);
```

**Argument**

- |          |   |
|----------|---|
| <b>x</b> | a fixed-point number in Q15 format, which ranges from -2 <sup>15</sup> to 2 <sup>15</sup> -1. The value of this argument ranges from 3277 to 32767. |
|----------|---|

**Return Value**

This function returns the log of  $x$  in Q15 format. The value ranges from -32768 to 0.

**5.3.20 \_Q15neg Function**

This function negates a Q15 value with saturation. The value is saturated in the case where the input is -32768.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15neg(_Q15 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.
----------	---

**Return Value**

This function returns  $-x$  in Q15 format. The value ranges from -32768 to 32767.

**5.3.21 \_Q15norm Function**

This function finds the normalized value of a Q15 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15norm(_Q15 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.
----------	---

**Return Value**

This function returns the square root of  $x$  in Q15 format. The value ranges from 16384 to -32767 for a positive number and -32768 to -16384 for a negative number.

**5.3.22 \_Q15power Function**

This function finds the power result given the base value and the power value in Q15 format.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15power(_Q15 x, _Q15 y);
```

**Argument**

<b>x</b>	a fixed-point number in Q15 format, which ranges from 1 to $2^{15}-1$ . The value of this argument ranges from 1 to 32767.
<b>y</b>	a fixed-point number in Q15 format, which ranges from 1 to $2^{15}-1$ . The value of this argument ranges from 1 to 32767.

**Return Value**

This function returns  $x$  to the power of  $y$  in Q15 format. The value ranges from 1 to 32767.

**5.3.23 \_Q15random Function**

This function generates a random number in the range from -32768 to 32767. The random number generation is periodic with period 65536. The function uses the `_Q15randomSeed` variable as a random seed value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15random(void);
```

**Argument**

None.

**Return Value**

This function returns a random number in Q15 format. The value ranges from -32768 to 32767.

**5.3.24 \_Q15shl Function**

This function shifts a Q15 value by `num` bits, to the left if `num` is positive or to the right if `num` is negative. The function takes care of saturating the result, in case of underflow or overflow.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15shl(_Q15 x, short num);
```

**Arguments**

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.
<b>num</b>	an integer number, which ranges from -15 to 15.

**Return Value**

This function returns the shifted value of `x` in Q15 format. The value ranges from -32768 to 32767.

**5.3.25 \_Q15shlNoSat Function**

This function shifts a Q15 value by `num` bits, to the left if `num` is positive or to the right if `num` is negative. The function sets the `_Q15shlSatFlag` variable in case of underflow or overflow but does not take care of saturation.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15shlNoSat(_Q15 x, short num);
```

**Arguments**

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.
<b>num</b>	an integer number, which ranges from -15 to 15.

**Return Value**

This function returns the shifted value of `x` in Q15 format. The value ranges from -32768 to 32767.

**5.3.26 \_Q15shr Function**

This function shifts a Q15 value by `num` bits, to the right if `num` is positive or to the left if `num` is negative. The function takes care of saturating the result, in case of underflow or overflow.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15shr(_Q15 x, short num);
```

**Arguments**

- |            |   |
|------------|---|
| <b>x</b>   | a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767. |
| <b>num</b> | an integer number, which ranges from -15 to 15.   |

**Return Value**

This function returns the shifted value of `x` in Q15 format. The value ranges from -32768 to 32767.

**5.3.27 \_Q15shrNoSat Function**

This function shifts a Q15 value by `num` bits, to the right if `num` is positive or to the left if `num` is negative. The function sets the `_Q15shrSatFlag` variable in case of underflow or overflow but does not take care of saturation.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15shrNoSat(_Q15 x, short num);
```

**Arguments**

- |            |   |
|------------|---|
| <b>x</b>   | a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767. |
| <b>num</b> | an integer number, which ranges from -15 to 15.   |

**Return Value**

This function returns the shifted value of `x` in Q15 format. The value ranges from -32768 to 32767.

**5.3.28 \_Q15sin Function**

This function finds the sine of a Q15 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15sin(_Q15 x);
```

**Argument**

- |          |   |
|----------|---|
| <b>x</b> | a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767. |
|----------|---|

**Return Value**

This function returns the sine of `x` in Q15 format. The value ranges from -27573 to 27573.

**5.3.29 \_Q15sinPI Function**

This function finds the sine of PI ( $\pi$ ) times a Q15 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15sinPI(_Q15 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.
----------	---

**Return Value**

This function returns the sine of PI times *x* in Q15 format. The value ranges from -32768 to 32767.

**5.3.30 \_Q15sinSeries Function**

Generates the sine series with the given normalizing frequency, *f*, and the given number of samples, *num*, starting from *start*. Stores the result in buffer, *buf*.

**Include**

```
<libq.h>
```

**Prototype**

```
short _Q15sinSeries(_Q15 f, short start, short num, _Q15 *buf);
```

**Arguments**

<b>f</b>	a fixed-point number in Q15 format, which ranges from 0 to $(2^{31}-1)$ . The valid range of values for this argument is from -16384 to 16384. The argument represents the Normalizing frequency.
<b>start</b>	a fixed-point number in Q16 format, which ranges from 0 to $(2^{31}-1)$ . The valid range of values for this argument is from 1 to 32767. This argument represents the Starting Sample number in the Sine Series.
<b>num</b>	a fixed-point number in Q16 format, which ranges from 0 to $(2^{31}-1)$ . The valid range of values for this argument is from 1 to 32767. This argument represents the Number of Sine Samples the function is called to generate. <b>Note:</b> <i>num</i> should not be more than 16383 for dsPIC and 32767 for PIC devices.
<b>buf</b>	a pointer to the buffer where the generated sine samples would get copied into.

**Return Value**

This function returns *num*, the number of generated sine samples.

**5.3.31 \_Q15sqrt Function**

This function finds the square root of a Q15 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q15 _Q15sqrt(_Q15 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from 1 to 32767.
----------	--

**Return Value**

This function returns the square root of  $x$  in Q15 format. The value ranges from 1 to 32767.

### 5.3.32 **\_Q15sub Function**

This function finds the difference of two Q15 values. The function takes care of saturation during overflow and underflow occurrences.

#### **Include**

```
<libq.h>
```

#### **Prototype**

```
_Q15 _Q15sub(_Q15 x, _Q15 y);
```

#### **Arguments**

- |          |   |
|----------|---|
| <b>x</b> | a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767. |
| <b>y</b> | a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767. |

#### **Return Value**

This function returns  $x$  minus  $y$  in Q15 format. The value ranges from -32768 to 32767.

### 5.3.33 **\_Q15tan Function**

This function finds the tangent of a Q15 value.

#### **Include**

```
<libq.h>
```

#### **Prototype**

```
_Q15 _Q15tan(_Q15 x);
```

#### **Argument**

- |          |   |
|----------|---|
| <b>x</b> | a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767. |
|----------|---|

#### **Return Value**

This function returns the tangent of  $x$  in Q15 format. The value ranges from -32768 to 32767.

### 5.3.34 **\_Q15tanPI Function**

This function finds the tangent of  $\pi$  times a Q15 value.

#### **Include**

```
<libq.h>
```

#### **Prototype**

```
_Q15 _Q15tanPI(_Q15 x);
```

#### **Argument**

- |          |   |
|----------|---|
| <b>x</b> | a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767. |
|----------|---|

#### **Return Value**

This function returns the tangent of  $\pi$  times  $x$  in Q15 format. The value ranges from -32768 to 32767.

**5.3.35 \_Q16acos Function**

This function finds the arc cosine of a Q16 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16acos(_Q16 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from -65536 to 65535.
----------	---

**Return Value**

This function returns the arc cosine of *x* in Q16 format. The value ranges from 205887 to 363 respectively.

**5.3.36 \_Q16acosByPI Function**

This function finds the arc cosine of a Q16 value and then divides by PI ( $\pi$ ).

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16acosByPI(_Q16 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from -65536 to 65535.
----------	---

**Return Value**

This function returns the arc cosine of *x*, divided by PI, in Q16 format. The value ranges from 65536 to 115 respectively.

**5.3.37 \_Q16asin Function**

This function finds the arc sine of a Q16 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16asin(_Q16 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from -65536 to 65535.
----------	---

**Return Value**

This function returns the arc sine of *x* in Q16 format. The value ranges from -102944 to 102579 respectively.

**5.3.38 \_Q16asinByPI Function**

This function finds the arc sine of a Q16 value and then divides by PI ( $\pi$ ).

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16asinByPI(_Q16 x);
```

**Argument**

**x** a fixed-point number in Q16 format. The value of this argument ranges from -65536 to 65535.

**Return Value**

This function returns the arc sine of  $x$ , divided by  $\pi$ , in Q16 format. The value ranges from -32768 to 32653 respectively.

**5.3.39 \_Q16atan Function**

This function finds the arc tangent of a Q16 value.

**Include**

`<libq.h>`

**Prototype**

`_Q16 _Q16atan(_Q16 x);`

**Argument**

**x** a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

**Return Value**

This function returns the arc tangent of  $x$  in Q16 format. The value ranges from -2147483648 to 2147483647.

**5.3.40 \_Q16atanByPI Function**

This function finds the arc tangent of a Q16 value and then divides by  $\pi$  ( $\pi$ ).

**Include**

`<libq.h>`

**Prototype**

`_Q16 _Q16atanByPI(_Q16 x);`

**Argument**

**x** a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

**Return Value**

This function returns the arc tangent of  $x$ , divided by  $\pi$ , in Q16 format. The value ranges from -2147483648 to 2147483647.

**5.3.41 \_Q16atanYByX Function**

This function finds the arc tangent of a Q16 value divided by a second Q16 value.

**Include**

`<libq.h>`

**Prototype**

`_Q16 _Q16atanYByX(_Q16 x, _Q16 y);`

**Arguments**

**x** a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.



**y** a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

**Return Value**

This function returns the arc tangent of *y* divided by *x* in Q16 format. The value ranges from -2147483648 to 2147483647.

**5.3.42 \_Q16atanYByXByPI Function**

This function finds the arc tangent of a Q16 value divided by a second Q16 value and then divides the result by PI ( $\pi$ ).

**Include**

<libq.h>

**Prototype**

```
_Q16 _Q16atanYByXByPI(_Q16 x, _Q16 y);
```

**Arguments**

**x** a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

**y** a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

**Return Value**

This function returns the arc tangent of *y* divided by *x*, divided by PI, in Q16 format. The value ranges from -2147483648 to 2147483647.

**5.3.43 \_Q16cos Function**

This function finds the cosine of a Q16 value.

**Include**

<libq.h>

**Prototype**

```
_Q16 _Q16cos(_Q16 x);
```

**Argument**

**x** a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

**Return Value**

This function returns the cosine of *x* in Q16 format. The value ranges from -65536 to 65535.

**5.3.44 \_Q16cosPI Function**

This function finds the cosine of PI ( $\pi$ ) times a Q16 value.

**Include**

<libq.h>

**Prototype**

```
_Q16 _Q16cosPI(_Q16 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
----------	---

**Return Value**

This function returns the cosine of  $\pi$  times  $x$  in Q16 format. The value ranges from -65536 to 65535.

**5.3.45 \_Q16div Function**

This function returns the quotient of its arguments.

**Include**

<libq.h>

**Prototype**

```
_Q16 _Q16div(_Q16 dividend, _Q16 divisor);
```

**Arguments**

<b>dividend</b>	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
<b>divisor</b>	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.

**Return Value**

This function returns the quotient of its arguments. The value ranges from 0 to 2147483647.

**5.3.46 \_Q16divmod Function**

This function returns the quotient and remainder of its arguments.

**Include**

<libq.h>

**Prototype**

```
_Q16 _Q16divmod(_Q16 dividend, _Q16 divisor, _Q16 *remainder);
```

**Arguments**

<b>dividend</b>	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
<b>divisor</b>	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
<b>remainder</b>	a pointer to an object large enough to hold the remainder. This must be provided by the caller.

**Return Value**

This function returns the quotient and remainder of its arguments. The values range from 0 to 2147483647.

**5.3.47 \_Q16exp Function**

This function finds the exponential value of a Q16 value.

**Include**

<libq.h>

**Prototype**

```
_Q16 _Q16exp(_Q16 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from -772244 to 681391.
----------	---

**Return Value**

This function returns the exponent value of  $x$  in Q16 format. The value ranges from 0 to 2147483647.

**5.3.48 \_Q16ftoi Function**

This function converts a single precision floating-point value into its corresponding Q16 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16ftoi(float x);
```

**Argument**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from -32768 to 32768.
----------	---

**Return Value**

This function returns a fixed-point number in Q16 format. The value ranges from -2147483648 to 2147483647.

**5.3.49 \_itofQ16 Function**

This function converts a Q16 value into its corresponding floating-point value.

**Include**

```
<libq.h>
```

**Prototype**

```
float _itofQ16(_Q16 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
----------	---

**Return Value**

This function returns a floating-point equivalent number. The corresponding floating-point range is -32768 to 32768.

**5.3.50 \_Q16log Function**

This function finds the natural log of a Q16 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16log(_Q16 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from 1 to 2147483647.
----------	---

**Return Value**

This function returns the natural log of x in Q16 format. The value ranges from -726817 to 681391.

**5.3.51 \_Q16log10 Function**

This function finds the log (base 10) of a Q16 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16log10(_Q16 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from 1 to 2147483647.
----------	---

**Return Value**

This function returns the log of *x* in Q16 format. The value ranges from -315653 to 295925.

**5.3.52 \_Q16mac Function**

This function multiplies the two 32-bit inputs, *x* and *y*, and accumulates the product with *prod*. The function takes care of saturating the result in case of underflow or overflow.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16mac(_Q16 x, _Q16 y, _Q16 prod);
```

**Arguments**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
<b>y</b>	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
<b>prod</b>	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.

**Return Value**

This function returns the multiplied and accumulated value *prod* in Q16 format. The value ranges from 0 to 2147483647.

**5.3.53 \_Q16macNoSat Function**

This function multiplies the two 32 bit inputs, *x* and *y* and accumulates the product with *prod*. This function only sets the *\_Q16macSatFlag* variable in case of an overflow or underflow and does not take care of saturation.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16macNoSat(_Q16 x, _Q16 y, _Q16 prod);
```

**Arguments**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
<b>y</b>	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
<b>prod</b>	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.

**Return Value**

This function returns the multiplied and accumulated value *prod* in Q16 format. The value ranges from 0 to 2147483647.

**5.3.54 \_Q16mpy Function**

This function returns the product of its arguments.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16mpy(_Q16 a, _Q16 b);
```

**Arguments**

<b>a</b>	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
<b>b</b>	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.

**Return Value**

This function returns the product of its arguments. The value ranges from 0 to 2147483647.

**5.3.55 \_Q16neg Function**

This function negates a Q16 value with saturation. The value is saturated in the case where the input is -2147483648.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16neg(_Q16 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
----------	---

**Return Value**

This function returns -x in Q16 format. The value ranges from -2147483648 to 2147483647.

**5.3.56 \_Q16norm Function**

This function finds the normalized value of a Q16 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16norm(_Q16 x);
```

**Argument**

x - a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
----------	---

**Return Value**

This function returns the square root of x in Q16 format. The value ranges from 16384 to -32767 for a positive number and -2147483648 to -1073741824 for a negative number.

**5.3.57 \_Q16power Function**

This function finds the power result given the base value and the power value in Q16 format.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16power(_Q16 x, _Q16 y);
```

**Argument**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
<b>y</b>	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.

**Return Value**

This function returns  $x$  to the power of  $y$  in Q16 format. The value ranges from 0 to 2147483647.

**5.3.58 \_Q16random Function**

This function generates a pseudo random number with a period of 2147483648. The function uses the `_Q16randomSeed` variable as a random seed value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16random(void);
```

**Argument**

None.

**Return Value**

This function returns a random number in Q16 format. The value ranges from -2147483648 to 2147483647.

**Remarks**

$\text{RndNum}(n) = (\text{RndNum}(n-1) * \text{RAN\_MULT}) + \text{RAN\_INC}$

SEED VALUE = 21845

RAN\_MULT = 1664525

RAN\_INC = 1013904223

**5.3.59 \_Q16reciprocalQ15 Function**

This function returns the reciprocal of a Q15 value. Since the input range lies in the -1 to +1 region, the output is always greater than the -1 or +1 region. So, Q16 format is used to represent the output.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16reciprocalQ15(_Q15 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q15 format, which ranges from $-2^{15}$ to $2^{15}-1$ . The value of this argument ranges from -32768 to 32767.
----------	---

**Return Value**

This function returns the reciprocal of  $x$  in Q16 format. The value ranges from -2147483648 to 2147418112.

**5.3.60 \_Q16reciprocalQ16 Function**

This function returns the reciprocal value of the input.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16reciprocalQ16(_Q16 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
----------	---

**Return Value**

This function returns the reciprocal of *x* in Q16 format. The value of this output ranges from -2147483648 to 2147483647.

**5.3.61 \_Q16shl Function**

This function shifts a Q16 value by *y* bits, to the left if *y* is positive or to the right if *y* is negative. The function takes care of saturating the result, in case of underflow or overflow.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16shl(_Q16 x, short y);
```

**Arguments**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
<b>y</b>	an integer number, which ranges from -32 to 32.

**Return Value**

This function returns the shifted value of *x* in Q16 format. The value ranges from -2147483648 to 2147483647.

**5.3.62 \_Q16shlNoSat Function**

This function shifts a Q16 value by *y* bits, to the left if *y* is positive or to the right if *y* is negative. The function sets the `_Q16shlSatFlag` variable in case of underflow or overflow but does not take care of saturation.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16shlNoSat(_Q16 x, short y);
```

**Arguments**

<b>x</b>	a fixed-point number in Q16 format, which ranges from -216 to 216-1. The value of this argument ranges from -2147483648 to 2147483647.
<b>y</b>	an integer number, which ranges from -32 to 32.

**Return Value**

This function returns the shifted value of *x* in Q16 format. The value ranges from -2147483648 to 2147483647.

**5.3.63 \_Q16shr Function**

This function shifts a Q16 value by *y* bits, to the right if *y* is positive or to the left if *y* is negative. The function takes care of saturating the result, in case of underflow or overflow.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16shr(_Q16 x, short y);
```

**Arguments**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
<b>y</b>	an integer number, which ranges from -32 to 32.

**Return Value**

This function returns the shifted value of *x* in Q16 format. The value ranges from -2147483648 to 2147483647.

**5.3.64 \_Q16shrNoSat Function**

This function shifts a Q16 value by *y* bits, to the right if *y* is positive or to the left if *y* is negative. The function sets the `_Q16shrSatFlag` variable in case of underflow or overflow but does not take care of saturation.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16shrNoSat(_Q16 x, short num);
```

**Arguments**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
<b>y</b>	an integer number, which ranges from -32 to 32.

**Return Value**

This function returns the shifted value of *x* in Q16 format. The value ranges from -2147483648 to 2147483647.

**5.3.65 \_Q16sin Function**

This function finds the sine of a Q16 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16sin(_Q16 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
----------	---

**Return Value**

This function returns the sine of *x* in Q16 format. The value ranges from -65566 to 65565.

**5.3.66 \_Q16sinPI Function**

This function finds the sine of PI ( $\pi$ ) times a Q16 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16sinPI(_Q16 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
----------	---



**Return Value**

This function returns the sine of  $\pi$  times  $x$  in Q16 format. The value ranges from -65536 to 65535.

**5.3.67 \_Q16sinSeries Function**

Generates the sine series with the given normalizing frequency,  $f$ , and the given number of samples,  $num$ , starting from  $start$ . Stores the result in buffer,  $buf$ .

**Include**

```
<libq.h>
```

**Prototype**

```
short _Q16sinSeries(_Q16 f, short start, short num, _Q16 *buf);
```

**Arguments**

- f** a fixed-point number in Q16 format, which ranges from 0 to  $(2^{31}-1)$ . The valid range of values for this argument is from -32768 to 32768. The argument represents the Normalizing frequency.
- start** a fixed-point number in Q16 format, which ranges from 0 to  $(2^{31}-1)$ . The valid range of values for this argument is from 1 to 32767. This argument represents the Starting Sample number in the Sine Series.
- num** a fixed-point number in Q16 format, which ranges from 0 to  $(2^{31}-1)$ . The valid range of values for this argument is from 1 to 32767. This argument represents the Number of Sine Samples the function is called to generate. **Note:**  $num$  should not be more than 16383 for dsPIC and 32767 for PIC devices.
- buf** a pointer to the buffer where the generated sine samples would get copied into.

**Return Value**

This function returns  $num$ , the number of generated sine samples.

**5.3.68 \_Q16tan Function**

This function finds the tangent of a Q16 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16tan(_Q16 x);
```

**Argument**

- x** a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.

**Return Value**

This function returns the tangent of  $x$  in Q16 format. The value ranges from -2147483648 to 2147483647.

**5.3.69 \_Q16tanPI Function**

This function finds the tangent of  $\pi$  times a Q16 value.

**Include**

```
<libq.h>
```

**Prototype**

```
_Q16 _Q16tanPI(_Q16 x);
```

**Argument**

<b>x</b>	a fixed-point number in Q16 format. The value of this argument ranges from -2147483648 to 2147483647.
----------	---

### Return Value

This function returns the tangent of PI times  $x$  in Q16 format. The value ranges from -2147483648 to 2147483647.

## 6. Revision History

The following is a list of changes by version to this document.

**Note:** Some revision letters are not used - the letters **I** and **O** - as they can be confused for numbers in some fonts.

### 6.1 Revision P (January 2022)

- **Preface:** Added.
- **1.5 Standard C Libraries with Math and Support Functions:** Now section 2.5. References the new “*Microchip Unified Standard Library Reference Guide*” (DS-50003209).
- **2. Standard C Libraries** and **3. Math Libraries:** Overviews merged into section 3.0. Updated based on Microchip Unified Standard Library. **2.1 Using the Standard C Libraries:** Now section 3.1. Added reference to Math Libraries.
- **3.2 List of Standard C and Math Library Functions:** New section. Points to the “*Microchip Unified Standard Library Reference Guide*” where all function descriptions now reside.
- **4.2 Standard C Library Helper Functions:** References the new “*Microchip Unified Standard Library Reference Guide*” (DS-50003209), “Syscall Interface” Chapter.
- **4.3 Standard C Library Functions That Require Modification:** References the new “*Microchip Unified Standard Library Reference Guide*” (DS-50003209), “Syscall Interface” Chapter.

### 6.2 Revision N (February 2021)

**Section 2.17.3 “clock Function”:** Stated that `clock()` function uses device timer2 and timer3.

**Section 4.8 “Functions to Support Secondary PRAM Program Memory”:** Terminology updated and added `_wipe_secondary` function.

### 6.3 Revision M (July 2020)

**Section 2.14 “<stdio.h> Input and Output”:** Update to `printf` function, additional `ll` modifier.

**Section 4.8 “Functions to Support Slave PRAM Program Memory”:** Section added from content in dsPIC33CH128MP508 (DS70005319) and dsPIC33CH512MP508 (DS70005371) family data sheets, Section 4.3 “Slave PRAM Program Memory”.

### 6.4 Revision L (December 2019)

- **Section 2.3 “<assert.h> Diagnostics”:** Added `__conditional_software_breakpoint` macro.
- **Section 2.4 “<ctype.h> Character Handling”:** Added `isblank` function.
- **Section 2.5 “<errno.h> Errors”:** Added `EILSEQ` macro.
- **Section 2.7 “<iso646.h> Alternate Spellings”:** Added a table of alternate spelling macros.
- **Section 2.15 “<stdlib.h> Utility Functions”:** Added “`strtol` family functions” section.
- **Chapter 5. “Fixed-Point Math Functions”:** Updated prototypes for `_Q16div` and `_Q16divmod`. Removed `_Q16reciprocal` as it is not in the library. This function is similar to `_Q16reciprocalQ16`.
- Other minor typographical, wording, and format changes.

### 6.5 Revision K (February 2018)

- **Section 4.8 “Functions for Specialized Copying and Initialization”** - added `_memcpy_eds`, `_strcpy_eds`, `_strncpy_eds`, `_memcpy_packed`, `_strcpy_packed`, `_strncpy_packed`. Also referenced in **Section 2.14 “<string.h> String Functions”** under `_memcpy`, `_strcpy`, `_strncpy`.

- 
- **Section 5.3.1 “Q15 Functions”** - updated `_Q15abs` return value.
  - **Section 5.3.2 “Q16 Functions”** - updated `_Q16norm` return value.

## 6.6 Revision J (December 2014)

- **Section 2.13 “<stdlib.h> Utility Functions”** - updated `malloc` function.
- **Section 4.7 “Functions for Erasing and Writing Flash Memory”** - updated descriptions for `_erase_flash` (PIC24FXXKA Only) and `_write_flash_word24()`.
- **Section 4.8 “Functions for Specialized Copying and Initialization”** - the following function descriptions have been updated:
  - `_memcpy_p2d24`
  - `_memcpy_p2d16`
  - `_strncpy_p2d24`
  - `_strncpy_p2d16`

## 6.7 Revision H (September 2013)

- **Section 1.1 “Introduction”** - added “Compiler Installation Locations”
- **Section 2.9 “<signal.h> Signal Handling”** - updated `signal` function.
- **Section 2.12 “<stdio.h> Input and Output”** - updated Customizing STDIO handles and descriptions for `fdopen` and `vasprintf`.
- **Section 4.3 “Standard C Library Helper Functions”** - updated `_dump_heap_info` function.
- **Section 4.7 “Functions for Erasing and Writing Flash Memory”** - updated descriptions for `_write_flash_word32` and `_write_flash_word48`.
- **Section 5.1 “Overview of Fixed-Point Data Formats”** - inserted Table 5-1 and Table 5-2.
- **Section 5.3 “<libq.h> Mathematical Functions”** - the following function descriptions have been updated:
  - `_Q16div`
  - `_Q16divmod`
  - `_Q16mpy`
- Minor typographical changes.

## 6.8 Revision G (October 2010)

- **Section 2.12 “<stdio.h> Input and Output”** - updated `scanf` function
- **Section 2.14 “<string.h> String Functions”** - updated `memchr` function
- Inserted **Section 5.1 “Overview of Fixed-Point Data Formats”**
- Minor typographical changes.

## 6.9 Revision F (March 2009)

- **Section 4.5 “Functions/Constants to Support A Simulated UART”** - the following function descriptions have been updated:
  - `_delay_ms`
  - `_delay_us`
  - `_C30UART`
- **Section 4.6 “Functions for Erasing and Writing EEDATA Memory”** - updated `_write_eedata_row` function.
- **Section 4.7 “Functions for Erasing and Writing Flash Memory”** - updated descriptions for `_erase_flash` and `_write_flash_word24`.

- **Section 5.2 “Using the Fixed-Point Libraries”** - updated descriptions for `Q16ftoi` and `itofQ16`.
- Minor typographical changes.

## 6.10 Revision E (January 2008)

**Section 2.3 “<ctype.h> Character Handling”** - the following function descriptions have been updated:

- `islower`
- `isxdigit`
- `tolower`
- `toupper`

**Section 2.5 “<float.h> Floating-Point Characteristics”** - the following function descriptions have been updated:

- `FLT_MAX_EXP`
- `FLT_MIN`
- `FLT_MIN_EXP`
- `LDBL_MIN`
- `LDBL_MIN_EXP`
- `UCHAR_MAX`
- `UINT_MAX`
- `ULLONG_MAX`
- `ULONG_MAX`
- `USHRT_MAX`

**Section 2.8 “<setjmp.h> Non-Local Jumps”** - the following function descriptions have been updated:

- `jmp_buf`
- `longjmp`
- `SIGILL`
- `SIGSEGV`
- `raise`
- `signal`

Divided **Chapter 3. “Standard C Libraries - Math Functions”** into **Chapter 4. “Standard C Libraries - Support Functions”** and **Chapter 5. “Fixed-Point Math Functions”**

Minor typographical changes.

## 6.11 Revision D (December 2006)

- **Section 3.7 “<limits.h> implementation-defined limits”** - the following function descriptions have been updated:
  - `LLONG_MIN`
  - `LONG_MAX`
  - `LONG_MIN`
  - `MB_LEN_MAX`
  - `SCHAR_MAX`
  - `SCHAR_MIN`
  - `SHRT_MAX`
  - `SHRT_MIN`
  - `UCHAR_MAX`
- **Section 3.9 “<setjmp.h> non-local jumps”** - the following function descriptions have been updated:

- 
- 
- – jmp\_buf
  - setjmp
  - longjmp
  - **Section 3.10 “<signal.h> signal handling”** - the following function descriptions have been updated:
  - – SIGFPE
  - SIGILL
  - SIGINT
  - raise
  - **Section 3.13 “<stdio.h> input and output”** - the following function descriptions have been updated:
  - – FILE
  - fpos\_t
  - \_IOFBF
  - \_IOLBF
  - \_IONBF
  - BUFSIZ
  - FOPEN\_MAX
  - L\_tmpnam
  - SEEK\_CUR
  - stderr
  - stdout
  - TMP\_MAX
  - clearerr
  - feof
  - fgetpos
  - fopen
  - fprintf
  - fread
  - Minor typographical changes

## 6.12 Revision C (October 2005)

- Updated **Section 2.7 “<locale.h> Localization”** and the following function descriptions:
- – PIDInit
- PIDCoeffCalc
- PID
- Updated **Section 2.8 “<setjmp.h> Non-Local Jumps”** and the following function descriptions:
- – Fract2Float
- Float2Fract
- Updated **Section 3.2 “Using the Standard C Libraries”**
- **Section 3.1 “<math.h> Mathematical Functions”** - the following function descriptions have been updated:
- – OpenXLCD
- putsXLCD
- SetDDRamAddr
- WriteDataXLCD
- WriteCmdXLCD
- Divided **Section 3.1 “<math.h> Mathematical Functions”** into **Section 3.4 “CAN Functions”**. The following function descriptions have been updated:
- – CAN1AbortAll

- CAN1GetRXErrorCount
  - CAN1GetTXErrorCount
  - CAN1IsBusOff
  - CAN1IsRXReady
  - CAN1IsRXPassive
  - CAN1IsTXPassive
- **Section 5.1 “Overview of Fixed-Point Data Formats”** - the following function descriptions have been updated:
  - `_builtin_addab`
  - `_builtin_add`
  - `_builtin_btg`
  - `_builtin_clr`
  - `_builtin_clr_prefetch`
  - `_builtin_dmaoffset`
  - `_builtin_ed`
  - `_builtin_edac`
  - `_builtin_fbcl`
  - `_builtin_mac`
  - `_builtin_movsac`
  - `_builtin_mpy`
  - `_builtin_mpy_n`
  - `_builtin_msc`
  - `_builtin_mulss`
  - `_builtin_mulsu`
  - `_builtin_mulus`
  - `_builtin_muluu`
  - `_builtin_nop`
  - `_builtin_psvpage`
- Minor typographical changes

### 6.13 Revision B (September 2004)

- Added "NOTICE TO CUSTOMERS"
- Divided **Chapter 4. “Standard C Libraries - Support Functions”** into **Chapter 5. “Fixed-Point Math Functions”**
- Added **Section 1.1 “OMF-Specific Libraries/Start-up Modules”**
- Minor typographical changes

### 6.14 Revision A (May 2004)

- Initial release of this document.

---

## The Microchip Website

---

Microchip provides online support via our website at [www.microchip.com/](http://www.microchip.com/). This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

## Product Change Notification Service

---

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to [www.microchip.com/pcn](http://www.microchip.com/pcn) and follow the registration instructions.

## Customer Support

---

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: [www.microchip.com/support](http://www.microchip.com/support)

## Microchip Devices Code Protection Feature

---

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip product is strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable". Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.

## Legal Notice

---

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. Contact your local Microchip sales office for additional support or, obtain additional support at [www.microchip.com/en-us/support/design-help/client-support-services](http://www.microchip.com/en-us/support/design-help/client-support-services).



---

THIS INFORMATION IS PROVIDED BY MICROCHIP "AS IS". MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

## Trademarks

---

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, CryptoMemory, CryptoRF, dsPIC, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AgileSwitch, APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, Flashtec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet- Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, TrueTime, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, Augmented Switching, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, GridTime, IdealBridge, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, Inter-Chip Connectivity, JitterBlocker, Knob-on-Display, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, NVM Express, NVMe, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SmartHLS, SMART-I.S., storClad, SQL, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, TSHARC, USBCheck, VariSense, VectorBlox, VeriPHY, ViewSpan, WiperLock, XpressConnect, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, Symmcom, and Trusted Time are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

© 2022, Microchip Technology Incorporated and its subsidiaries. All Rights Reserved.

ISBN: 978-1-5224-9638-0

---

## Quality Management System

---

For information regarding Microchip's Quality Management Systems, please visit [www.microchip.com/quality](http://www.microchip.com/quality).

## Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
<b>Corporate Office</b> 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: <a href="http://www.microchip.com/support">www.microchip.com/support</a> Web Address: <a href="http://www.microchip.com">www.microchip.com</a>	<b>Australia - Sydney</b> Tel: 61-2-9868-6733 <b>China - Beijing</b> Tel: 86-10-8569-7000 <b>China - Chengdu</b> Tel: 86-28-8665-5511 <b>China - Chongqing</b> Tel: 86-23-8980-9588 <b>China - Dongguan</b> Tel: 86-769-8702-9880 <b>China - Guangzhou</b> Tel: 86-20-8755-8029 <b>China - Hangzhou</b> Tel: 86-571-8792-8115 <b>China - Hong Kong SAR</b> Tel: 852-2943-5100 <b>China - Nanjing</b> Tel: 86-25-8473-2460 <b>China - Qingdao</b> Tel: 86-532-8502-7355 <b>China - Shanghai</b> Tel: 86-21-3326-8000 <b>China - Shenyang</b> Tel: 86-24-2334-2829 <b>China - Shenzhen</b> Tel: 86-755-8864-2200 <b>China - Suzhou</b> Tel: 86-186-6233-1526 <b>China - Wuhan</b> Tel: 86-27-5980-5300 <b>China - Xian</b> Tel: 86-29-8833-7252 <b>China - Xiamen</b> Tel: 86-592-2388138 <b>China - Zhuhai</b> Tel: 86-756-3210040	<b>India - Bangalore</b> Tel: 91-80-3090-4444 <b>India - New Delhi</b> Tel: 91-11-4160-8631 <b>India - Pune</b> Tel: 91-20-4121-0141 <b>Japan - Osaka</b> Tel: 81-6-6152-7160 <b>Japan - Tokyo</b> Tel: 81-3-6880-3770 <b>Korea - Daegu</b> Tel: 82-53-744-4301 <b>Korea - Seoul</b> Tel: 82-2-554-7200 <b>Malaysia - Kuala Lumpur</b> Tel: 60-3-7651-7906 <b>Malaysia - Penang</b> Tel: 60-4-227-8870 <b>Philippines - Manila</b> Tel: 63-2-634-9065 <b>Singapore</b> Tel: 65-6334-8870 <b>Taiwan - Hsin Chu</b> Tel: 886-3-577-8366 <b>Taiwan - Kaohsiung</b> Tel: 886-7-213-7830 <b>Taiwan - Taipei</b> Tel: 886-2-2508-8600 <b>Thailand - Bangkok</b> Tel: 66-2-694-1351 <b>Vietnam - Ho Chi Minh</b> Tel: 84-28-5448-2100	<b>Austria - Wels</b> Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 <b>Denmark - Copenhagen</b> Tel: 45-4485-5910 Fax: 45-4485-2829 <b>Finland - Espoo</b> Tel: 358-9-4520-820 <b>France - Paris</b> Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 <b>Germany - Garching</b> Tel: 49-8931-9700 <b>Germany - Haan</b> Tel: 49-2129-3766400 <b>Germany - Heilbronn</b> Tel: 49-7131-72400 <b>Germany - Karlsruhe</b> Tel: 49-721-625370 <b>Germany - Munich</b> Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 <b>Germany - Rosenheim</b> Tel: 49-8031-354-560 <b>Israel - Ra'anana</b> Tel: 972-9-744-7705 <b>Italy - Milan</b> Tel: 39-0331-742611 Fax: 39-0331-466781 <b>Italy - Padova</b> Tel: 39-049-7625286 <b>Netherlands - Drunen</b> Tel: 31-416-690399 Fax: 31-416-690340 <b>Norway - Trondheim</b> Tel: 47-72884388 <b>Poland - Warsaw</b> Tel: 48-22-3325737 <b>Romania - Bucharest</b> Tel: 40-21-407-87-50 <b>Spain - Madrid</b> Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 <b>Sweden - Gothenberg</b> Tel: 46-31-704-60-40 <b>Sweden - Stockholm</b> Tel: 46-8-5090-4654 <b>UK - Wokingham</b> Tel: 44-118-921-5800 Fax: 44-118-921-5820