MPLAB XC-DSC 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/) 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/.

Table of Contents

1. Preface

MPLAB® XC-DSC 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. In most cases, formatting conforms to the *OASIS Darwin Information Typing Architecture (DITA) Version 1.3 Part 3: All-Inclusive Edition*, 19 June 2018.

Table 1-1. Documentation Conventions

Description	Implementation	Examples
References	DITA: cite	*MPLAB XC_DSC Libraries Reference Manual.*
Emphasized text	Italics	...is the *only* compiler...
A window, window pane or dialog name.	DITA: wintitle	the **Output** window. the **New Watch** dialog.
A field name in a window or dialog.	DITA: uicontrol	Select the **Optimizations** option category.
A menu name or item.	DITA: uicontrol	Select the **File** menu and then **Save**.
A menu path.	DITA: menucascade, uicontrol	**File** > **Save**
A tab	DITA: uicontrol	Click the **Power** tab.
A software button.	DITA: uicontrol	Click the **OK** button.
A key on the keyboard.	DITA: uicontrol	Press the **F1** key.
File names and paths.	DITA: filepath	`C:\Users\User1\Projects`
Source code: inline.	DITA: codeph	Remember to `#define START` at the beginning of your code.

Source code: block.	DITA: codeblock	An example is:
		`#include <xc.h>`
		`main(void) {`
		`  while(1);`
		`}`

User-entered data.	DITA: userinput	Type in a device name, for example `PIC18F47Q10`.
Keywords	DITA: codeph	`static`, `auto`, `extern`
Command-line options.	DITA: codeph	`-Opa+, -Opa-`
Bit values	DITA: codeph	`0, 1`
Constants	DITA: codeph	`0xFF, 'A'`
A variable argument.	DITA: codeph + option	*`file`*`.o`, where *`file`* can be any valid file name.
Optional arguments	Square brackets [ ]	`xc8 [`*`options`*`]` *`files`*
Choice of mutually exclusive arguments; an OR selection.	Curly brackets and pipe character: { | }	`errorlevel {0|1}`
Replaces repeated text.	Ellipses...	*`var_name`* `[,` *`var_name`*`...]`

Represents code supplied by user.	Ellipses...	`void main (void)`
		`{ ...`
		`}`

A number in verilog format, where N is the total number of digits, R is the radix and n is a digit.	N'Rnnnn	4'b0010, 2'hF1
Device Dependent insignia. Specifies that a feature is not supported on all devices. Devices supported will be listed in the title or text.	[DD]	`xmemory` attribute

1.2 Recommended Reading

This guide describes how to use the MPLAB XC-DSC 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® XC-DSC C Compiler User's Guide (DS-50003589)

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

MPLAB® XC-DSC Assembler, Linker and Utilities User's Guide (DS-50003590)

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

Device-Specific Documentation

The Microchip website contains many documents that describe DSC 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® XC-DSC Assembler, Linker and Utilities User's Guide* (DS-50003590) for more information about making and using libraries.

Compiler Installation Locations

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

- Windows OS 64-bit - `C:\Program Files\Microchip\xc-dsc\vx.xx`
- Mac OS - `/Applications/microchip/xc-dsc/vx.xx`
- Linux OS - `/opt/microchip/xc-dsc/vx.xx`

where `vx.xx` is the version number.

Assembly Code Applications

Free versions of the MPLAB XC-DSC libraries are available from the Microchip web site. DSP and DSC 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 XC-DSC C Compiler.

C Code Applications

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

2.1 Specific Libraries and Startup Modules

Library files and start-up modules use the Executable and Linkable Format (ELF). The debugging format used for ELF object files is DWARF 2.0.

MPLAB XC-DSC tools will first look for generic library files when building your application. If these cannot be found, then ELF files will be used. As an example, if `libdsp.a` is not found then the file `libdsp-elf.a` will be used.

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 in `libc99-pic30-elf.a`; the source code for these modules is included with `libpic30.zip` found in the `src` directory of the installation. 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® XC-DSC Assembler, Linker and Utilities User's Guide* (DS-50003590) and for C applications, the *MPLAB® XC-DSC C Compiler User's Guide* (DS-50003589).

2.3 DSP Library

The DSP library (`libdsp-elf.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 provided. By default, the documentation is found in the `docs\dsp_lib` subdirectory of the MPLAB XC-DSC C Compiler install directory (see “Libraries Overview, Compiler Installation Locations”).

2.4 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.

2.5 Fixed-Point Math Functions

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

2.6 Compiler Built-in Functions

The MPLAB XC-DSC C Compiler contains built-in functions that, to the developer, work like library functions. These functions are listed in the *MPLAB® XC-DSC C Compiler User’s Guide* (DS-50003589).

3. Standard C and Math Libraries

Standard ANSI C and math library functions are contained in the file `libc99-elf.a`.

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 XC-DSC 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
- `include` – 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: `libc99-elf.a`, `libc99-pic30-elf.a` and `libm-elf.a`. 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® XC-DSC Assembler, Linker and Utilities User’s Guide* (DS-50003590) and *MPLAB® XC-DSC C Compiler User’s Guide* (DS-50003589) 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 `libc99-pic30-elf.a` archive and the source code is available in the `src` folder of the `libpic30.zip` file.

For details on using the standard C libraries, see the *Microchip Unified Standard Library Reference Guide* (DS-50003209).

4.1 Rebuilding the libc99-pic30 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 may be modified for your target application. These functions may be based on the source files provided (for location, see 4. Support Libraries) but it is not necessary to rebuild the library. Simply providing your own definition of the function in your project will override the basic version provided in the library.

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 `libc99-pic30-elf.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, refer to the “Syscall Interface” section in the *Microchip Unified Standard Library Reference Guide* (DS-50003209).

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 `libc99-pic30-elf.a` archive and the source code (for the compiler) is available in the `src\pic30` folder. These modules are not distributed as part of `libc99-elf.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 X 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
   DSC 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>

#ifndef __dsPIC33F__
#error this is a 33F demo for the explorer 16(tm) board
#endif
#inlcude <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

p	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

`cycles`	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

`time`	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.

Example 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 */
}
```

4.5.1 _erase_eedata Function

Erase EEDATA memory on dsPIC30F devices.

Include

`<libpic30.h>`

Prototype

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

Argument

dst	destination memory address
len	dsPIC30F: length may be _EE_WORD or _EE_ROW (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 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 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

`dst`	destination memory address
`src`	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 devices.

Include

`<libpic30.h>`

Prototype

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

Arguments

`dst`	destination memory address
`dat`	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.

Example of Use

Example 1: dsPIC30F DSCs

```
#include "libpic30.h"
#include "p30fxxxx.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 */
  _erase_flash(p); /* on dsPIC30F, only 1 row per page */
  _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

`dst`	destination memory address

Return Value

None.

Remarks

None.

Default Behavior

Erase a page of Flash memory.

File

`flash_helper.s`

4.6.2 _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

`dst`	destination memory address
`src`	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`

4.6.3 _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

`dst`	destination memory address
`src`	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`

4.6.4 _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 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:

- *dsPIC33FJXXXGPX06/X08/X10 Family Silicon Errata and Data Sheet Clarification* (DS-80000446)
- *dsPIC33FJXXXMCX06/X08/X10 Family Silicon Errata and Data Sheet Clarification* (DS-80000447)

Include

`<libpic30.h>`

Prototype

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

Arguments

`dst`	destination memory address
`dat1,dat2`	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 devices.

File

`flash_helper.c`

4.6.5 _write_flash_word48 Function

Write two words of FLASH memory with 24 bits of data per word. Word writes are supported dsPIC33E 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:

- *dsPIC33FJXXXGPX06/X08/X10 Family Silicon Errata and Data Sheet Clarification* (DS-80000446)
- *dsPIC33FJXXXMCX06/X08/X10 Family Silicon Errata and Data Sheet Clarification* (DS-80000447)

Include

`<libpic30.h>`

Prototype

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

Arguments

`dst`	destination memory address
`dat1,dat2`	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 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

a	variable of type `_prog_addressT`
b	initialization value for variable *a*

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

`dest`	destination memory address
`src`	address of data to be written
`len`	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

`dest`	destination memory address
`src`	address of data to be written
`len`	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

`dest`	destination memory address
`src`	address of data to be written
`len`	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

`dest`	destination memory address
`src`	address of data to be written
`len`	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

dest	destination memory address
src	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

dest	destination memory address
src	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 `strncpy` 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

`dest`	destination memory address
`src`	address of data to be written
`len`	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

`dest`	destination memory address
`src`	address of data to be written

`len`	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

`dest`	destination memory address
`src`	address of data to be written
`len`	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* _strcpy_packed( void *dest, __pack_upper_byte void *src, int len);
```

Arguments

`dest`	destination memory address
`src`	address of data to be written
`len`	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 DSC 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 intialization code
  _start_secondary(); // Start Secondary core

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

  while(1);
}
```

Reference Documents:

dsPIC33CH128MP508 Family Data Sheet (DS-70005319), Section 4.3

dsPIC33CH512MP508 Family Data Sheet (DS-70005371), 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

`secondary_number`	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

`secondary_number`	Identifier of the Secondary core to be programmed. The identifier for the first Secondary core is `1`.
`verify`	A `0` will load the entire Secondary image to the PRAM. A `1` will verify the entire image in the PRAM.
`*image`	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

`secondary_number`	Identifier of the Secondary core to be programmed. The identifier for the first Secondary core is `1`.
`verify`	A `0` will load the entire Secondary image to the PRAM. A `1` will verify the entire image in the PRAM.

`*image`	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-elf.a` (standard) and `libq-dsp-elf.a` (DSP). 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 compliment to accommodate both positive and negative numbers in binary format. The two's compliment 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

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

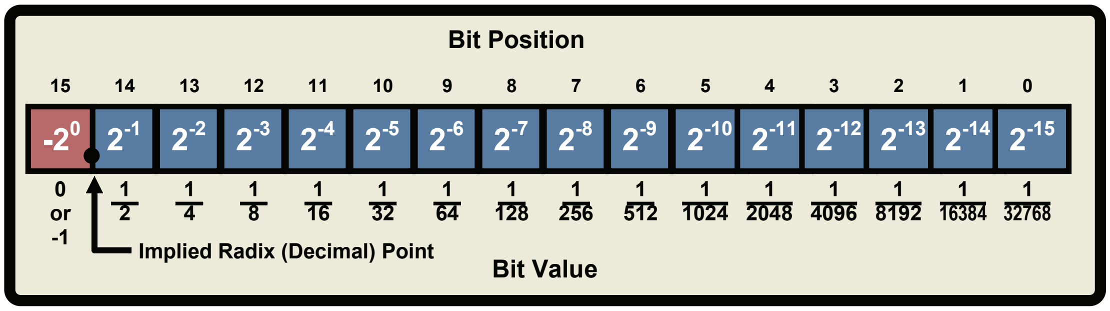
Figure 2: Equation Value

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

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 3.05176×10^{-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.

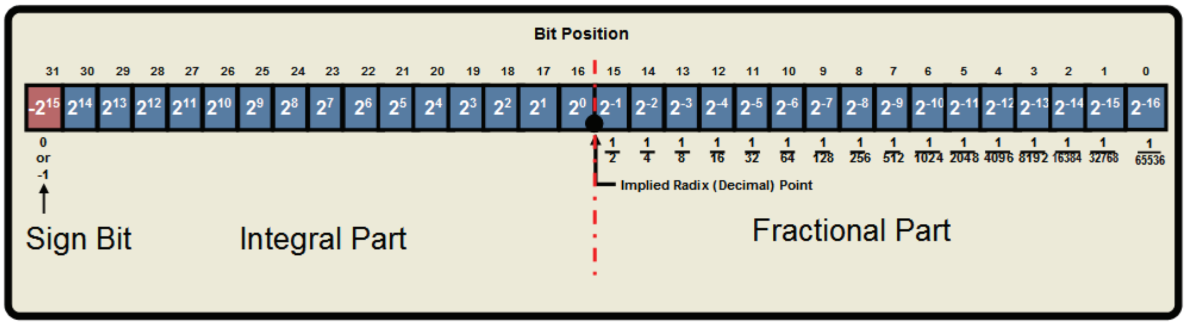
Table 5-1. Table1: Conversion of a Two's Complement 16-Bit Integer to Q15

Binary		Dec		Q15
0	0 x (-2^{15})	0	0 x (-2^{0})	0
			●	●
1	1 x 2^{14}	16384	1 x 2^{-1}	0.5
1	1 x 2^{13}	8192	1 x 2^{-2}	0.25
0	0 x 2^{12}	0	0 x 2^{-3}	0
0	0 x 2^{11}	0	0 x 2^{-4}	0
0	0 x 2^{10}	0	0 x 2^{-5}	0
0	0 x 2^{9}	0	0 x 2^{-6}	0
0	0 x 2^{8}	0	0 x 2^{-7}	0
0	0 x 2^{7}	0	0 x 2^{-8}	0
0	0 x 2^{6}	0	0 x 2^{-9}	0
0	0 x 2^{5}	0	0 x 2^{-10}	0
0	0 x 2^{4}	0	0 x 2^{-11}	0
0	0 x 2^{3}	0	0 x 2^{-12}	0
0	0 x 2^{2}	0	0 x 2^{-13}	0
0	0 x 2^{1}	0	0 x 2^{-14}	0
0	0 x 2^{0}	0	0 x 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 2-16.

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 x (-2^{31})	-2147483648	1 x (-2^{15})	-32768
1	1 x 2^{30}	1073741824	1 x 2^{14}	16384
0	0 x 2^{29}	0	0 x 2^{13}	0
1	1 x 2^{28}	268435456	1 x 2^{12}	4096
0	0 x 2^{27}	0	0 x 2^{11}	0
1	1 x 2^{26}	67108864	1 x 2^{10}	1024
0	0 x 2^{25}	0	0 x 2^{9}	0
1	1 x 2^{24}	16777216	1 x 2^{8}	256
0	0 x 2^{23}	0	0 x 2^{7}	0
1	1 x 2^{22}	4194304	1 x 2^{6}	64
0	0 x 2^{21}	0	0 x 2^{5}	0
1	1 x 2^{20}	1048576	1 x 2^{4}	16
0	0 x 2^{19}	0	0 x 2^{3}	0
1	1 x 2^{18}	262144	1 x 2^{2}	4
0	0 x 2^{17}	0	0 x 2^{1}	0
1	1 x 2^{16}	65536	1 x 2^{0}	1
			●	●
0	0 x 2^{15}	0	0 x 2^{-1}	0
1	1 x 2^{14}	16384	1 x 2^{-2}	0.25
0	0 x 2^{13}	0	0 x 2^{-3}	0
1	0 x 2^{12}	4096	1 x 2^{-4}	0.0625
0	0 x 2^{11}	0	0 x 2^{-5}	0
1	1 x 2^{10}	1024	1 x 2^{-6}	0.015625
0	0 x 2^{9}	0	0 x 2^{-7}	0
1	1 x 2^{8}	256	1 x 2^{-8}	0.00390625
0	0 x 2^{7}	0	0 x 2^{-9}	0
1	1 x 2^{6}	64	1 x 2^{-10}	0.000976563
0	0 x 2^{5}	0	0 x 2^{-11}	0
1	1 x 2^{4}	16	1 x 2^{-12}	0.000244141
0	0 x 2^{3}	0	0 x 2^{-13}	0
1	1 x 2^{2}	4	1 x 2^{-14}	6.10352E-05
1	1 x 2^{1}	2	1 x 2^{-15}	3.01576E-05
0	0 x 2^{0}	0	0 x 2^{-16}	0

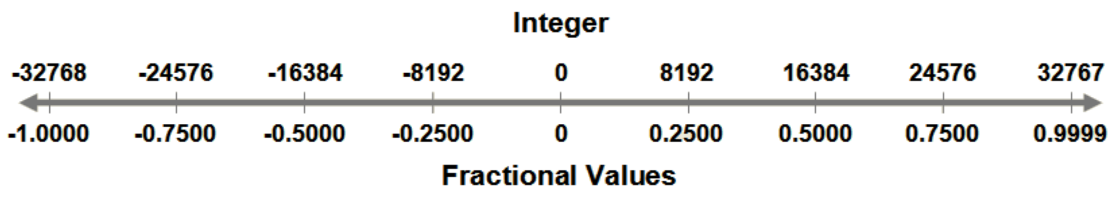
...........continued

Binary		Dec		Q15.16
	SUM	-715827882	SUM	-10922.6666564941
● = Radix Point				

Integer - Fractional Format Mapping

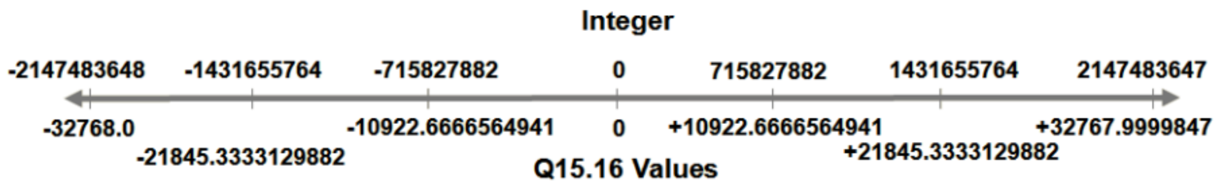
The same twos complement N-bit word may represent an integer format value or a fractional format value. For example., the 16-bit integer range [-32768, +32767] format maps to Q15 range of [-1.0, +0.999969482]. Figure 5 shows the mapping between these formats.

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-elf.a` or `libq-dsp-elf.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 these library files: `libc99-elf.a`, `libm-elf.a` and `libc99-pic30-elf.a`. 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
_Q0_15	16	_Q15
_Q15_16	32	_Q16

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
_Q16reciprocalQ15	_Q16	_Q15
_Q16reciprocalQ16	_Q16	_Q16

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

x	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

x	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 (π).

Include

`<libq.h>`

Prototype

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

Argument

x	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

x	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.
y	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

x	a fixed-point number in Q15 format, which ranges from -2^{15} to 2^{15}-1. The value of this argument ranges from -27573 to 27573.

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 (π).

Include

```
<libq.h>
```

Prototype

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

Argument

x	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

x	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 (π).

Include

`<libq.h>`

Prototype

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

Argument

x	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

x	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.
y	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 (π).

Include

```
<libq.h>
```

Prototype

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

Arguments

x	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.
y	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

s	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

x	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 (π) times a Q15 value.

Include

`<libq.h>`

Prototype

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

Argument

x	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

x	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

x	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

`x`	a fixed-point number in Q15 format, which ranges from -215 to 215-1. The value of this argument ranges from -32768 to 32767.
`s`	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

`x`	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 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

`x`	a fixed-point number in Q15 format, which ranges from -2^{15} to 2^{15}-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

x	a fixed-point number in Q15 format, which ranges from -2^{15} to 2^{15}-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

x	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

x	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

x	a fixed-point number in Q15 format, which ranges from 1 to 215-1. The value of this argument ranges from 1 to 32767.

y	a fixed-point number in Q15 format, which ranges from 1 to 215-1. 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

x	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.
num	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

x	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.
num	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

x	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.
num	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

x	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.
num	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

x	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 (π) times a Q15 value.

Include

`<libq.h>`

Prototype

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

Argument

x	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

f	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.
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.

5.3.31 _Q15sqrt Function

This function finds the square root of a Q15 value.

Include

`<libq.h>`

Prototype

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

Argument

`x`	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

`x`	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.
`y`	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

`x`	a fixed-point number in Q15 format, which ranges from -2^{15} to 2^{15}-1. The value of this argument ranges from -25736 to 25735.

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 (π) times a Q15 value.

Include

`<libq.h>`

Prototype

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

Argument

x	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

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 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 (π).

Include

`<libq.h>`

Prototype

```
_Q16 _Q16acosByPI(_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 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

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` 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 (π).

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 (π).

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 (π).

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 (π) times a Q16 value.

Include

`<libq.h>`

Prototype

```
_Q16 _Q16cosPI(_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 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

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

`dividend`	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
`divisor`	a fixed-point number in Q16 format. The value of this argument ranges from 0 to 2147483647.
`remainder`	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

`x`	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

`x`	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

`x`	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

x	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

x	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

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

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

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

x	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.

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

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

RndNum(n) = (RndNum(n-1) * RAN_MULT) + 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

x	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

x	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

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

x	a fixed-point number in Q16 format, which ranges from -216 to 216-1. The value of this argument ranges from -2147483648 to 2147483647.
Y	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

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

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

x	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 (π) times a Q16 value.

Include

`<libq.h>`

Prototype

```
_Q16 _Q16sinPI(_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 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 (π) times a Q16 value.

Include

`<libq.h>`

Prototype

```
_Q16 _Q16tanPI(_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 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 A (October 2023)

- Initial release of this document.

The Microchip Website

Microchip provides online support via our website at 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 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

Product Identification System

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.

PART NO.	[X][1]	-	X	/XX	XXX
Device	Tape and Reel Option		Temperature Range	Package	Pattern

Device:	PIC16F18313, PIC16LF18313, PIC16F18323, PIC16LF18323	
Tape and Reel Option:	Blank	= Standard packaging (tube or tray)
	T	= Tape and Reel[1]
Temperature Range:	I	= -40°C to +85°C (Industrial)
	E	= -40°C to +125°C (Extended)
Package:[2]	JQ	= UQFN
	P	= PDIP
	ST	= TSSOP
	SL	= SOIC-14
	SN	= SOIC-8
	RF	= UDFN
Pattern:	QTP, SQTP, Code or Special Requirements (blank otherwise)	

Examples:

- PIC16LF18313- I/P Industrial temperature, PDIP package
- PIC16F18313- E/SS Extended temperature, SSOP package

Notes:

1. Tape and Reel identifier only appears in the catalog part number description. This identifier is used for ordering purposes and is not printed on the device package. Check with your Microchip Sales Office for package availability with the Tape and Reel option.
2. Small form-factor packaging options may be available. Please check www.microchip.com/packaging for small-form factor package availability, or contact your local Sales Office.

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

Trademarks

ISBN: 978-1-6683-3269-6

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4485-5910
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-72400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra’anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-72884388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820