16-BIT LANGUAGE TOOLS
LIBRARIES
Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip’s Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.
# Preface

# Chapter 1. Library Overview

1.1 Introduction ................................................................. 7
1.2 OMF-Specific Libraries/Start-up Modules .............................................. 7
1.3 Start-up Code ........................................................................ 8
1.4 DSP Library ........................................................................ 8
1.5 16-Bit Peripheral Libraries .............................................................. 8
1.6 Standard C Libraries (with Math Functions) ........................................... 8
1.7 MPLAB C30 Built-in Functions ....................................................... 8

# Chapter 2. DSP Library

2.1 Introduction ........................................................................ 9
2.2 Using the DSP Library ................................................................ 10
2.3 Vector Functions .................................................................... 13
2.4 Window Functions ................................................................... 26
2.5 Matrix Functions .................................................................... 31
2.6 Filtering Functions ................................................................... 38
2.7 Transform Functions ................................................................ 58
2.8 Control Functions ................................................................... 73
2.9 Miscellaneous Functions ............................................................ 78

# Chapter 3. 16-Bit Peripheral Libraries

3.1 Introduction ........................................................................ 81
3.2 Using the 16-Bit Peripheral Libraries ............................................ 82
3.3 External LCD Functions ........................................................... 82
3.4 CAN Functions ...................................................................... 89
3.5 ADC12 Functions .................................................................. 103
3.6 ADC10 Functions .................................................................. 110
3.7 Timer Functions ..................................................................... 117
3.8 Reset/Control Functions ............................................................ 125
3.9 I/O Port Functions .................................................................. 128
3.10 Input Capture Functions ............................................................ 133
3.11 Output Compare Functions ......................................................... 139
3.12 UART Functions ..................................................................... 149
3.13 SCI Functions ....................................................................... 158
3.14 SPI Functions ....................................................................... 166
3.15 QEI Functions ....................................................................... 174
3.16 PWM Functions ..................................................................... 179
3.17 I2C™ Functions ..................................................................... 191
Chapter 4. Standard C Libraries with Math Functions

4.1 Introduction ................................................................. 201
4.2 Using the Standard C Libraries ........................................ 202
4.3 <assert.h> diagnostics .................................................... 203
4.4 <ctype.h> character handling ........................................... 204
4.5 <errno.h> errors ............................................................ 213
4.6 <float.h> floating-point characteristics ............................... 214
4.7 <limits.h> implementation-defined limits .......................... 219
4.8 <locale.h> localization .................................................... 221
4.9 <setjmp.h> non-local jumps ............................................. 222
4.10 <signal.h> signal handling .............................................. 223
4.11 <stdarg.h> variable argument lists ..................................... 229
4.12 <stddef.h> common definitions ....................................... 231
4.13 <stdio.h> input and output ............................................. 233
4.14 <stdlib.h> utility functions ............................................ 278
4.15 <string.h> string functions ............................................ 302
4.16 <time.h> date and time functions ..................................... 325
4.17 <math.h> mathematical functions ..................................... 333
4.18 pic30-libs ................................................................. 374

Chapter 5. MPLAB C30 Built-in Functions

5.1 Introduction ................................................................. 383
5.2 Built-In Function List ..................................................... 384

Appendix A. ASCII Character Set ......................................................... 401

Index ................................................................................................. 403

Worldwide Sales and Service ................................................................. 424
Preface

NOTICE TO CUSTOMERS

All documentation becomes dated, and this manual is no exception. Microchip 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 web site (www.microchip.com) to obtain the latest documentation available.

Documents are identified with a “DS” number. This number is located on the bottom of each page, in front of the page number. The numbering convention for the DS number is “DSXXXXXA”, where “XXXXX” is the document number and “A” is the revision level of the document.

For the most up-to-date information on development tools, see the MPLAB IDE on-line help. Select the Help menu, and then Topics to open a list of available on-line help files.

INTRODUCTION

The purpose of this document is to define and describe the libraries that are available for use with Microchip Technology’s 16-bit language tools, based on GCC (GNU Compiler Collection) technology. The related language tools are:

- MPLAB® ASM30 Assembler
- MPLAB C30 C Compiler
- MPLAB LINK30 Linker
- MPLAB LIB30 Archiver/Librarian
- Other Utilities

Items discussed in this chapter include:

- About This Guide
- Recommended Reading
- Troubleshooting
- The Microchip Web Site
- Development Systems Customer Change Notification Service
- Customer Support
ABOUT THIS GUIDE

Document Layout

This document describes how to use GNU language tools to write code for 16-bit applications. The document layout is as follows:

- **Chapter 1: Library Overview** – gives an overview of libraries.
- **Chapter 2: DSP Library** – lists the library functions for DSP operation.
- **Chapter 3: 16-Bit Peripherals Libraries** – lists the library functions and macros for 16-bit device software and hardware peripheral operation.
- **Chapter 4: Standard C Library with Math Functions** – lists the library functions and macros for standard C operation.
- **Chapter 5: MPLAB C30 Built-in Functions** – lists the built-in functions of the C compiler, MPLAB C30.
Conventions Used in this Guide

This manual uses the following documentation conventions:

### DOCUMENTATION CONVENTIONS

<table>
<thead>
<tr>
<th>Description</th>
<th>Represents</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arial font:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Italic characters</td>
<td>Referenced books</td>
<td>MPLAB® IDE User’s Guide</td>
</tr>
<tr>
<td>Emphasized text</td>
<td>...is the only compiler...</td>
<td></td>
</tr>
<tr>
<td>Initial caps</td>
<td>A window</td>
<td>the Output window</td>
</tr>
<tr>
<td></td>
<td>A dialog</td>
<td>the Settings dialog</td>
</tr>
<tr>
<td></td>
<td>A menu selection</td>
<td>select Enable Programmer</td>
</tr>
<tr>
<td>Quotes</td>
<td>A field name in a window or dialog</td>
<td>“Save project before build”</td>
</tr>
<tr>
<td>Underlined, italic text with right angle bracket</td>
<td>A menu path</td>
<td>File&gt;Save</td>
</tr>
<tr>
<td>Bold characters</td>
<td>A dialog button</td>
<td>Click OK</td>
</tr>
<tr>
<td></td>
<td>A tab</td>
<td>Click the Power tab</td>
</tr>
<tr>
<td>‘bnnnn’</td>
<td>A binary number where n is a digit</td>
<td>‘b00100, ‘b10</td>
</tr>
<tr>
<td>Text in angle brackets &lt; &gt;</td>
<td>A key on the keyboard</td>
<td>Press &lt;Enter&gt;, &lt;F1&gt;</td>
</tr>
<tr>
<td><strong>Courier font:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plain Courier</td>
<td>Sample source code</td>
<td>#define START</td>
</tr>
<tr>
<td></td>
<td>Filenames</td>
<td>autoexec.bat</td>
</tr>
<tr>
<td></td>
<td>File paths</td>
<td>c:\mcc18\h</td>
</tr>
<tr>
<td></td>
<td>Keywords</td>
<td>_asm, _endasm, static</td>
</tr>
<tr>
<td></td>
<td>Command-line options</td>
<td>-Opa+, -Opa-</td>
</tr>
<tr>
<td></td>
<td>Bit values</td>
<td>0, 1</td>
</tr>
<tr>
<td>Italic Courier</td>
<td>A variable argument</td>
<td>file.o, where file can be any valid filename</td>
</tr>
<tr>
<td>0xnnnnn</td>
<td>A hexadecimal number where n is a hexadecimal digit</td>
<td>0xFFFF, 0x007A</td>
</tr>
<tr>
<td>Square brackets [ ]</td>
<td>Optional arguments</td>
<td>mcc18 [options] file [options]</td>
</tr>
<tr>
<td>Curly brackets and pipe character: {</td>
<td>Choice of mutually exclusive arguments; an OR selection</td>
<td>errorlevel {0</td>
</tr>
<tr>
<td>character: {</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ellipses...</td>
<td>Replaces repeated text</td>
<td>var_name [, var_name...]</td>
</tr>
<tr>
<td></td>
<td>Represents code supplied by user</td>
<td>void main (void) { ... }</td>
</tr>
</tbody>
</table>
RECOMMENDED READING

This document describes 16-bit library functions and macros. For more information on 16-bit language tools and the use of other tools, the following are recommended reading:

README Files
For the latest information on Microchip tools, read the associated README files (ASCII text files) included with the software.

dsPIC® Language Tools Getting Started (DS70094)
A guide to installing and working with the Microchip language tools (MPLAB ASM30, MPLAB LINK30 and MPLAB C30) for 16-bit devices. Examples using the 16-bit simulator, MPLAB SIM30, are provided.

MPLAB® ASM30, MPLAB® LINK30 and Utilities User’s Guide (DS51317)
A guide to using the 16-bit assembler, MPLAB ASM30, 16-bit linker, MPLAB LINK30 and various 16-bit utilities, including MPLAB LIB30 archiver/librarian.

MPLAB® C30 C Compiler User’s Guide (DS51284)
A guide to using the 16-bit C compiler. MPLAB LINK30 is used with this tool.

dsPIC30F Family Overview (DS70043)
An overview of the dsPIC30F devices and architecture.

dsPIC30F/33F Programmer’s Reference Manual (DS70157)
Programmer’s guide to dsPIC30F/33F devices. Includes the programmer’s model and instruction set.

Microchip Web Site
The Microchip web site (http://www.microchip.com) contains a wealth of documentation. Individual data sheets, application notes, tutorials and user’s guides are all available for easy download. All documentation is in Adobe Acrobat (PDF) format.

TROUBLESHOOTING

See the README files for information on common problems not addressed in this document.
THE MICROCHIP WEB SITE

Microchip provides online support via our WWW site at www.microchip.com. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **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 (FAQ), technical support requests, online discussion groups, Microchip consultant 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

DEVELOPMENT SYSTEMS CUSTOMER CHANGE NOTIFICATION SERVICE

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

To register, access the Microchip web site at www.microchip.com, click on Customer Change Notification and follow the registration instructions.

The Development Systems product group categories are:

- **Compilers** – The latest information on Microchip C compilers and other language tools. These include the MPLAB C18 and MPLAB C30 C compilers; MPASM™ and MPLAB ASM30 assemblers; MPLINK™ and MPLAB LINK30 object linkers; and MPLIB™ and MPLAB LIB30 object librarians.
- **Emulators** – The latest information on Microchip in-circuit emulators. This includes the MPLAB ICE 2000 and MPLAB ICE 4000.
- **In-Circuit Debuggers** – The latest information on the Microchip in-circuit debugger, MPLAB ICD 2.
- **MPLAB IDE** – The latest information on Microchip MPLAB IDE, the Windows® Integrated Development Environment for development systems tools. This list is focused on the MPLAB IDE, MPLAB SIM and MPLAB SIM30 simulators, MPLAB IDE Project Manager and general editing and debugging features.
- **Programmers** – The latest information on Microchip programmers. These include the MPLAB PM3 and PRO MATE® II device programmers and the PICSTART® Plus development programmer.
CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or field application engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://support.microchip.com
Chapter 1. Library Overview

1.1 INTRODUCTION

A library is a collection of functions grouped for reference and ease of linking. See the “MPLAB ASM30, MPLAB LINK30 and Utilities User’s Guide” (DS51317) for more information about making and using libraries.

1.1.1 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 C30 C compiler.

1.1.2 C Code Applications

The 16-bit language tool libraries are included in the c:\Program Files\Microchip\MPLAB C30\lib directory, where c:\Program Files\Microchip\MPLAB C30 is the MPLAB C30 C compiler install directory. These can be linked directly into an application with MPLAB LINK30.

1.1.3 Chapter Organization

This chapter is organized as follows:

- OMF-Specific Libraries/Start-up Modules
- Start-up Code
- DSP Library
- 16-Bit Peripheral Libraries
- Standard C Libraries (with Math Functions)
- MPLAB C30 Built-in Functions

1.2 OMF-SPECIFIC LIBRARIES/START-UP MODULES

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

- COFF – This is the default.
- ELF – The debugging format used for ELF object files is DWARF 2.0.

There are two ways to select the OMF:

1. Set an environment variable called PIC30_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.
1.3 START-UP CODE

In order to initialize variables in data memory, the linker creates a data initialization template. This template must be processed at start-up, before the application proper takes control. For C programs, this function is performed by the start-up modules in libpic30-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 start-up module file. The source code for the start-up modules is provided in corresponding .s files.

The primary start-up 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 start-up module (crt1) performs no data initialization.

For more on start-up code, see the “MPLAB ASM30, MPLAB LINK30 and Utilities User's Guide” (DS51317) and, for C applications, the “MPLAB C30 C Compiler User’s Guide” (DS51284).

1.4 DSP LIBRARY

The DSP library (libdsp-omf.a) provides a set of digital signal processing operations to a program targeted for execution on a dsPIC30F digital signal controller (DSC). In total, 49 functions are supported by the DSP Library.

1.5 16-BIT PERIPHERAL LIBRARIES

The 16-bit (software and hardware) peripheral libraries provide functions and macros for setting up and controlling 16-bit peripherals. Examples of use are also provided in each related chapter of this book.

These libraries are processor-specific and of the form libpDevice-omf.a, where Device = 16-bit device number (e.g., libp30F6014-coff.a for the dsPIC30F6014 device).

1.6 STANDARD C LIBRARIES (WITH MATH FUNCTIONS)

A complete set of ANSI-89 conforming libraries are provided. The standard C library files are libc-omf.a (written by Dinkumware, an industry leader) and libm-omf.a (math functions, written by Microchip).

Additionally, some 16-bit standard C library helper functions, and standard functions that must be modified for use with 16-bit devices, are in libpic30-omf.a.

A typical C application will require all three libraries.

1.7 MPLAB C30 BUILT-IN FUNCTIONS

The MPLAB C30 C compiler contains built-in functions that, to the developer, work like library functions.
Chapter 2. DSP Library

2.1 INTRODUCTION

The DSP Library provides a set of digital signal processing operations to a program targeted for execution on a dsPIC30F/33F digital signal controller. The library has been designed to provide you, the C software developer, with efficient implementation of the most common signal processing functions. In total, 52 functions are supported by the DSP Library.

A primary goal of the library is to minimize the execution time of each function. To achieve this goal, the DSP Library is predominantly written in optimized assembly language. By using the DSP Library, you can realize significant gains in execution speed over equivalent code written in ANSI C. Additionally, since the DSP Library has been rigorously tested, using the DSP Library will allow you to shorten your application development time.

2.1.1 Assembly Code Applications

A free version of this library and its associated header file is available from the Microchip web site. Source code is included.

2.1.2 C Code Applications

The MPLAB C30 C compiler install directory (c:\program files\microchip\mplab c30) contains the following subdirectories with library-related files:

- lib – DSP library/archive files
- src\dsp – source code for library functions and a batch file to rebuild the library
- support\h – header file for DSP library

2.1.3 Chapter Organization

This chapter is organized as follows:

- Using the DSP Library
- Vector Functions
- Window Functions
- Matrix Functions
- Filtering Functions
- Transform Functions
- Control Functions
- Miscellaneous Functions
2.2 USING THE DSP LIBRARY

2.2.1 Building with the DSP Library

Building an application which utilizes the DSP Library requires only two files: dsp.h and libdsp-omf.a. dsp.h is a header file which provides all the function prototypes, #defines and typedefs used by the library. libdsp-omf.a is the archived library file which contains all the individual object files for each library function. (See Section 1.2 “OMF-Specific Libraries/Start-up Modules” for more on OMF-specific libraries.)

When compiling an application, dsp.h must be referenced (using #include) by all source files which call a function in the DSP Library or use its symbols or typedefs. When linking an application, libdsp-omf.a must be provided as an input to the linker (using the --library or -l linker switch) such that the functions used by the application may be linked into the application.

The linker will place the functions of the DSP library into a special text section named .libdsp. This may be seen by looking at the map file generated by the linker.

2.2.2 Memory Models

The DSP Library is built with the “small code” and “small data” memory models to create the smallest library possible. Since a few DSP library functions are written in C and make use of the compiler’s floating-point library, the MPLAB C30 linker script files place the .libm and .libdsp text sections next to each other. This ensures that the DSP library may safely use the RCALL instruction to call the required floating-point routines in the floating-point library.

2.2.3 DSP Library Function Calling Convention

All the object modules within the DSP Library are compliant with the C compatibility guidelines for the dsPIC30F/33F DSC and follow the function call conventions documented in the Microchip “MPLAB® C30 C Compiler User’s Guide” (DS51284). Specifically, functions may use the first eight working registers (W0 through W7) as function arguments. Any additional function arguments are passed through the stack.

The working registers W0 to W7 are treated as scratch memory, and their values may not be preserved after the function call. On the other hand, if any of the working registers W8 to W13 are used by a function, the working register is first saved, the register is used and then its original value is restored upon function return. The return value of a (non void) function is available in working register W0 (also referred to as WREG).

When needed, the run time software stack is used following the C system stack rules described in the “MPLAB® C30 Compiler User’s Guide”. Based on these guidelines, the object modules of the DSP Library can be linked to either a C program, an assembly program or a program which combines code in both languages.

2.2.4 Data Types

The operations provided by the DSP Library have been designed to take advantage of the DSP instruction set and architectural features of the dsPIC30F/33F DSC. In this sense, most operations are computed using fractional arithmetic.

The DSP Library defines a fractional type from an integer type:

```c
#ifndef fractional
typedef int fractional;
#endif
```

The fractional data type is used to represent data that has 1 sign bit, and 15 fractional bits. Data which uses this format is commonly referred to as “1.15” data.
For functions which use the multiplier, results are computed using the 40-bit accumulator, and “9.31” arithmetic is utilized. This data format has 9 sign/magnitude bits and 31 fractional bits, which provides for extra computational headroom above the range (-1.00 to ~+1.00) provided by the 1.15 format. Naturally when these functions provide a result, they revert to a fractional data type, with 1.15 format.

The use of fractional arithmetic imposes some constraints on the allowable set of values to be input to a particular function. If these constraints are ensured, the operations provided by the DSP Library typically produce numerical results correct to 14 bits. However, several functions perform implicit scaling to the input data and/or output results, which may decrease the resolution of the output values (when compared to a floating-point implementation).

A subset of operations in the DSP Library, which require a higher degree of numerical resolution, do operate in floating-point arithmetic. Nevertheless, the results of these operations are transformed into fractional values for integration with the application. The only exception to this is the MatrixInvert function which computes the inversion of a floating-point matrix in floating-point arithmetic, and provides the results in floating-point format.

### 2.2.5 Data Memory Usage

The DSP Library performs no allocation of RAM, and leaves this task to you. If you do not allocate the appropriate amount of memory and align the data properly, undesired results will occur when the function executes. In addition, to minimize execution time, the DSP Library will do no checking on the provided function arguments (including pointers to data memory), to determine if they are valid. The user may refer to example projects that utilize the DSP library functions, in order to ascertain proper usage of functions. MPLAB IDE-based example projects/workspaces have been provided in the installation folder of the MPLAB C30 toolsuite.

Most functions accept data pointers as function arguments, which contain the data to be operated on, and typically also the location to store the result. For convenience, most functions in the DSP Library expect their input arguments to be allocated in the default RAM memory space (X-Data or Y-Data), and the output to be stored back into the default RAM memory space. However, the more computational intensive functions require that some operands reside in X-Data and Y-Data (or program memory and Y-Data), so that the operation can take advantage of the dual data fetch capability of the 16-bit architecture.

### 2.2.6 CORCON Register Usage

Many functions of the DSP Library place the dsPIC30F/33F device into a special operating mode by modifying the CORCON register. On the entry of these functions, the CORCON register is pushed to the stack. It is then modified to correctly perform the desired operation, and lastly the CORCON register is popped from the stack to preserve its original value. This mechanism allows the library to execute as correctly as possible, without disrupting CORCON setting.

When the CORCON register is modified, it is typically set to 0x00F0. This places the dsPIC30F/33F device into the following operational mode:

- DSP multiplies are set to used signed and fractional data
- Accumulator saturation is enabled for Accumulator A and Accumulator B
- Saturation mode is set to 9.31 saturation (Super Saturation)
- Data Space Write Saturation is enabled
- Program Space Visibility disabled
- Convergent (unbiased) rounding is enabled
For a detailed explanation of the CORCON register and its effects, refer to the "dsPIC30F Family Reference Manual" (DS70046).

2.2.7 Overflow and Saturation Handling

The DSP Library performs most computations using 9.31 saturation, but must store the output of the function in 1.15 format. If during the course of operation the accumulator in use saturates (goes above 0x7F FFFF FFFF or below 0x80 0000 0000), the corresponding saturation bit (SA or SB) in the STATUS register will be set. This bit will stay set until it is cleared. This allows you to inspect SA or SB after the function executes and to determine if action should be taken to scale the input data to the function.

Similarly, if a computation performed with the accumulator results in an overflow (the accumulator goes above 0x00 7FFF FFFF or below 0xFF 8000 0000), the corresponding overflow bit (OA or OB) in the STATUS register will be set. Unlike the SA and SB status bits, OA and OB will not stay set until they are cleared. These bits are updated each time an operation using accumulator is executed. If exceeding this specified range marks an important event, you are advised to enable the Accumulator Overflow Trap via the OVATE, OVBTE and COVTE bits in the INTCON1 register. This will have the effect of generating an Arithmetic Error Trap as soon as the Overflow condition occurs, and you may then take the required action.

2.2.8 Integrating with Interrupts and an RTOS

The DSP Library may easily be integrated into an application which utilizes interrupts or an RTOS, yet certain guidelines must be followed. To minimize execution time, the DSP Library utilizes DO loops, REPEAT loops, Modulo addressing and Bit-Reversed addressing. Each of these components is a finite hardware resource on the 16-bit device, and the background code must consider the use of each resource when disrupting execution of a DSP Library function.

When integrating with the DSP Library, you must examine the Function Profile of each function description to determine which resources are used. If a library function will be interrupted, it is your responsibility to save and restore the contents of all registers used by the function, including the state of the DO, REPEAT and special addressing hardware. Naturally this also includes saving and restoring the contents of the CORCON and Status registers.

2.2.9 Rebuilding the DSP Library

A batch file named makedsplib.bat is provided to rebuild the DSP library. The MPLAB C30 compiler is required to rebuild the DSP library, and the batch file assumes that the compiler is installed in the default directory, c:\Program Files\Microchip\MPLAB C30\. If your language tools are installed in a different directory, you must modify the directories in the batch file to match the location of your language tools.
2.3 VECTOR FUNCTIONS

This section presents the concept of a fractional vector, as considered by the DSP Library, and describes the individual functions which perform vector operations.

2.3.1 Fractional Vector Operations

A fractional vector is a collection of numerical values, the vector elements, allocated contiguously in memory, with the first element at the lowest memory address. One word of memory (two bytes) is used to store the value of each element, and this quantity must be interpreted as a fractional number represented in the 1.15 data format.

A pointer addressing the first element of the vector is used as a handle which provides access to each of the vector values. The address of the first element is referred to as the base address of the vector. Because each element of the vector is 16 bits, the base address must be aligned to an even address.

The one dimensional arrangement of a vector accommodates to the memory storage model of the device, so that the nth element of an N-element vector can be accessed from the vector's base address BA as:

\[ BA + 2(n - 1), \text{ for } 1 \leq n \leq N. \]

The factor of 2 is used because of the byte addressing capabilities of the 16-bit device.

Unary and binary fractional vector operations are implemented in this library. The operand vector in a unary operation is called the source vector. In a binary operation the first operand is referred to as the source one vector, and the second as the source two vector. Each operation applies some computation to one or several elements of the source vector(s). Some operations produce a result which is a scalar value (also to be interpreted as a 1.15 fractional number), while other operations produce a result which is a vector. When the result is also a vector, this is referred to as the destination vector.

Some operations resulting in a vector allow computation in place. This means the results of the operation are placed back into the source vector (or the source one vector for binary operations). In this case, the destination vector is said to (physically) replace the source (one) vector. If an operation can be computed in place, it is indicated as such in the comments provided with the function description.

For some binary operations, the two operands can be the same (physical) source vector, which means the operation is applied to the source vector and itself. If this type of computation is possible for a given operation, it is indicated as such in the comments provided with the function description.

Some operations can be both self applicable and computed in place.

All the fractional vector operations in this library take as an argument the cardinality (number of elements) of the operand vector(s). Based on the value of this argument the following assumptions are made:

a) The sum of sizes of all the vectors involved in a particular operation falls within the range of available data memory for the target device.

b) In the case of binary operations, the cardinalities of both operand vectors must obey the rules of vector algebra (particularly, see remarks for the VectorConvolve and VectorCorrelate functions).

c) The destination vector must be large enough to accept the results of an operation.
2.3.2 User Considerations

a) No boundary checking is performed by these functions. Out of range cardinalities (including zero length vectors) as well as nonconforming use of source vector sizes in binary operations may produce unexpected results.

b) The vector addition and subtraction operations could lead to saturation if the sum of corresponding elements in the source vector(s) is greater than $1-2^{-15}$ or smaller than $-1.0$. Analogously, the vector dot product and power operations could lead to saturation if the sum of products is greater than $1-2^{-15}$ or smaller than $-1.0$.

c) It is recommended that the STATUS Register (SR) be examined after completion of each function call. In particular, users can inspect the SA, SB and SAB flags after the function returns to determine if saturation occurred.

d) All the functions have been designed to operate on fractional vectors allocated in default RAM memory space (X-Data or Y-Data).

e) Operations which return a destination vector can be nested, so that for instance if:

\[
\begin{align*}
    a &= \text{Op1} \ (b, c), \quad \text{with} \quad b &= \text{Op2} \ (d), \quad \text{and} \quad c &= \text{Op3} \ (e, f), \quad \text{then} \\
    a &= \text{Op1} \ (\text{Op2} \ (d), \ \text{Op3} \ (e, f))
\end{align*}
\]

2.3.3 Additional Remarks

The description of the functions limits its scope to what could be considered the regular usage of these operations. However, since no boundary checking is performed during computation of these functions, you have the freedom to interpret the operation and its results as it fits some particular needs.

For instance, while computing the VectorMax function, the length of the source vector could be greater than `numElems`. In this case, the function would be used to find the maximum value only among the first `numElems` elements of the source vector.

As another example, you may be interested in replacing `numElems` elements of a destination vector located between \( N \) and \( N+\text{numElems}-1 \), with `numElems` elements from a source vector located between elements \( M \) and \( M+\text{numElems}-1 \). Then, the VectorCopy function could be used as follows:

\[
\begin{align*}
    \text{fractional}^* & \ dstV[\text{DST\_ELEMS}] = \{\ldots\}; \\
    \text{fractional}^* & \ srcV[\text{SRC\_ELEMS}] = \{\ldots\}; \\
    \text{int} & \ n = \text{NUM\_ELEMS}; \\
    \text{int} & \ N = \text{N\_PLACE}; \quad /* \text{NUM\_ELEMS}+N \leq \text{DST\_ELEMS} */ \\
    \text{int} & \ M = \text{M\_PLACE}; \quad /* \text{NUM\_ELEMS}+M \leq \text{SRC\_ELEMS} */ \\
    \text{fractional}^* & \ dstVector = \ dstV+N; \\
    \text{fractional}^* & \ srcVector = \ srcV+M; \\

dstVector = \text{VectorCopy} \ (n, \ dstVector, \ srcVector);
\]

Also in this context, the VectorZeroPad function can operate in place, where now \( dstV = srcV, \text{numElems} \) is the number of elements at the beginning of source vector to preserve, and \( \text{numZeros} \) the number of elements at the vector tail to set to zero.

Other possibilities can be exploited from the fact that no boundary checking is performed.
2.3.4 Individual Functions

In what follows, the individual functions implementing vector operations are described.

VectorAdd

**Description:** VectorAdd adds the value of each element in the source one vector with its counterpart in the source two vector, and places the result in the destination vector.

**Include:** dsp.h

**Prototype:**

```c
extern fractional* VectorAdd (  
    int numElems,  
    fractional* dstV,  
    fractional* srcV1,  
    fractional* srcV2  
);
```

**Arguments:**
- `numElems` number of elements in source vectors
- `dstV` pointer to destination vector
- `srcV1` pointer to source one vector
- `srcV2` pointer to source two vector

**Return Value:** Pointer to base address of destination vector.

**Remarks:** If the absolute value of `srcV1[n] + srcV2[n]` is larger than $1 - 2^{-15}$, this operation results in saturation for the n-th element.
This function can be computed in place.
This function can be self applicable.

**Source File:** vadd.s

**Function Profile:**

System resources usage:
- W0..W4 used, not restored
- ACCA used, not restored
- CORCON saved, used, restored

DO and REPEAT instruction usage:
- 1 level DO instructions
- no REPEAT instructions

Program words (24-bit instructions):
- 13

Cycles (including C-function call and return overheads):
- $17 + 3(numElems)$
VectorConvolve

**Description:** VectorConvolve computes the convolution between two source vectors, and stores the result in a destination vector. The result is computed as follows:

\[
y(n) = \sum_{k=0}^{n} x(k)h(n-k), \text{ for } 0 \leq n < M
\]

\[
y(n) = \sum_{k=n-M+1}^{n} x(k)h(n-k), \text{ for } M \leq n < N
\]

\[
y(n) = \sum_{k=n-M+1}^{N-1} x(k)h(n-k), \text{ for } N \leq n < N + M - 1
\]

where \(x(k)\) = source one vector of size \(N\), \(h(k)\) = source two vector of size \(M\) (with \(M \leq N\)).

**Include:**

dsp.h

**Prototype:**

```c
extern fractional* VectorConvolve (int numElems1, int numElems2, fractional* dstV, fractional* srcV1, fractional* srcV2);
```

**Arguments:**
- `numElems1` number of elements in source one vector
- `numElems2` number of elements in source two vector
- `dstV` pointer to destination vector
- `srcV1` pointer to source one vector
- `srcV2` pointer to source two vector

**Return Value:** Pointer to base address of destination vector.

**Remarks:**
- The number of elements in the source two vector must be less than or equal to the number of elements in the source one vector.
- The destination vector must already exist, with exactly `numElems1 + numElems2 - 1` number of elements.
- This function can be self applicable.

**Source File:** vcon.s
VectorConvolve (Continued)

Function Profile: System resources usage:
- W0..W7 used, not restored
- W8..W10 saved, used, restored
- ACCA used, not restored
- CORCON saved, used, restored

DO and REPEAT instruction usage:
- 2 level DO instructions
- no REPEAT instructions

Program words (24-bit instructions):
- 2 level DO instructions
- 1 level REPEAT instructions
- 6

Cycles (including C-function call and return overheads):
- For N = numElems1, and M = numElems2,
  \[ 28 + 13M + \sum_{m=1}^{M} m + (N - M)(7 + 3M), \text{ for } M < N \]
  \[ 28 + 13M + \sum_{m=1}^{M} m, \text{ for } M = N \]

VectorCopy

Description: VectorCopy copies the elements of the source vector into the
beginning of an (already existing) destination vector, so that:
\[ dstV[n] = srcV[n], 0 \leq n < numElems \]

Include: dsp.h

Prototype:

\[
\text{extern fractional* VectorCopy (}
  \text{int numElems,}
  \text{fractional* dstV,}
  \text{fractional* srcV});
\]

Arguments:
- numElems number of elements in source vector
- dstV pointer to destination vector
- srcV pointer to source vector

Return Value: Pointer to base address of destination vector.

Remarks: The destination vector must already exist. Destination vectors must
have, at least, numElems elements, but could be longer.
This function can be computed in place. See Additional Remarks at the
end of the section for comments on this mode of operation.

Source File: vcopy.s

Function Profile: System resources usage:
- W0..W3 used, not restored

DO and REPEAT instruction usage:
- no DO instructions
- 1 level REPEAT instructions

Program words (24-bit instructions):
- 6

Cycles (including C-function call and return overheads):
- 12 + numElems
VectorCorrelate

Description: VectorCorrelate computes the correlation between two source vectors, and stores the result in a destination vector. The result is computed as follows:

\[ r(n) = \sum_{k=0}^{N-1} x(k) y(k+n), \text{ for } 0 \leq n < N + M - 1 \]

where \( x(k) = \) source one vector of size \( N \), \( y(k) = \) source two vector of size \( M \) (with \( M \leq N \)).

Include: dsp.h

Prototype:

\[
\text{extern fractional* VectorCorrelate (}
\begin{align*}
\text{int numElems1,} \\
\text{int numElems2,} \\
\text{fractional* dstV,} \\
\text{fractional* srcV1,} \\
\text{fractional* srcV2}
\end{align*}
\);
\]

Arguments:

- \( \text{numElems1} \): number of elements in source one vector
- \( \text{numElems2} \): number of elements in source two vector
- \( \text{dstV} \): pointer to destination vector
- \( \text{srcV1} \): pointer to source one vector
- \( \text{srcV2} \): pointer to source two vector

Return Value: Pointer to base address of destination vector.

Remarks:

- The number of elements in the source two vector must be less than or equal to the number of elements in the source one vector.
- The destination vector must already exist, with exactly \( \text{numElems1} + \text{numElems2} - 1 \) number of elements.
- This function can be self applicable.
- This function uses VectorConvolve.

Source File: vcor.s.s

Function Profile:

- System resources usage:
  - W0..W7 used, not restored,
  - plus resources from VectorConvolve

- DO and REPEAT instruction usage:
  - 1 level DO instructions
  - no REPEAT instructions,
  - plus DO/REPEAT instructions from VectorConvolve

- Program words (24-bit instructions):
  - 14,
  - plus program words from VectorConvolve

- Cycles (including C-function call and return overheads):
  - \( 19 + \text{floor}(M / 2) * 3 \), with \( M = \text{numElems2} \),
  - plus cycles from VectorConvolve.

Note: In the description of VectorConvolve the number of cycles reported includes 4 cycles of C-function call overhead. Thus, the number of actual cycles from VectorConvolve to add to VectorCorrelate is 4 less than whatever number is reported for a stand-alone VectorConvolve.
### VectorDotProduct

**Description:** VectorDotProduct computes the sum of the products between corresponding elements of the source one and source two vectors.

**Include:**

dsp.h

**Prototype:**

```c
extern fractional VectorDotProduct (  
    int numElems,  
    fractional* srcV1,  
    fractional* srcV2  
);
```

**Arguments:**

- `numElems` number of elements in source vectors
- `srcV1` pointer to source one vector
- `srcV2` pointer to source two vector

**Return Value:** Value of the sum of products.

**Remarks:** If the absolute value of the sum of products is larger than $1-2^{-15}$, this operation results in saturation.

This function can be self applicable.

**Source File:** vdot.s

**Function Profile:**

- **System resources usage:**
  - W0..W2 used, not restored
  - W4..W5 used, not restored
  - ACCA used, not restored
  - CORCON saved, used, restored

- **DO and REPEAT instruction usage:**
  - 1 level DO instructions
  - no REPEAT instructions

- **Program words (24-bit instructions):**
  - 13

- **Cycles (including C-function call and return overheads):**
  - $17 + 3(numElems)$

### VectorMax

**Description:** VectorMax finds the last element in the source vector whose value is greater than or equal to any previous vector element. Then, it outputs that maximum value and the index of the maximum element.

**Include:**

dsp.h

**Prototype:**

```c
extern fractional VectorMax (  
    int numElems,  
    fractional* srcV,  
    int* maxIndex  
);
```

**Arguments:**

- `numElems` number of elements in source vector
- `srcV` pointer to source vector
- `maxIndex` pointer to holder for index of (last) maximum element

**Return Value:** Maximum value in vector.

**Remarks:** If $srcV[i] = srcV[j] = maxVal$, and $i < j$, then $*maxIndex = j$.

**Source File:** vmax.s
VectorMax (Continued)

Function Profile: System resources usage:
W0..W5 used, not restored

DO and REPEAT instruction usage:
no DO instructions
no REPEAT instructions

Program words (24-bit instructions):
13

Cycles (including C-function call and return overheads):
14
if numElems = 1
20 + 8(numElems - 2)
if srcV[n] ≥ srcV[n + 1], 0 ≤ n < numElems - 1
19 + 7(numElems - 2)
if srcV[n] < srcV[n + 1], 0 ≤ n < numElems - 1

VectorMin

Description: VectorMin finds the last element in the source vector whose value is
less than or equal to any previous vector element. Then, it outputs that
minimum value and the index of the minimum element.

Include: dsp.h

Prototype: extern fractional VectorMin (int numElems,
 fractional* srcV,
 int* minIndex );

Arguments: numElems number of elements in source vector
srcV pointer to source vector
minIndex pointer to holder for index of (last) minimum element

Return Value: Minimum value in vector.

Remarks: If srcV[i] = srcV[j] = minVal, and i < j, then
*minIndex = j.

Source File: vmin.s

Function Profile: System resources usage:
W0..W5 used, not restored

DO and REPEAT instruction usage:
no DO instructions
no REPEAT instructions

Program words (24-bit instructions):
13

Cycles (including C-function call and return overheads):
14
if numElems = 1
20 + 8(numElems - 2)
if srcV[n] ≥ srcV[n + 1], 0 ≤ n < numElems - 1
19 + 7(numElems - 2)
if srcV[n] < srcV[n + 1], 0 ≤ n < numElems - 1
### VectorMultiply

**Description:** VectorMultiply multiplies the value of each element in source one vector with its counterpart in source two vector, and places the result in the corresponding element of destination vector.

**Include:**

dsp.h

**Prototype:**

```c
extern fractional* VectorMultiply (
    int numElems,
    fractional* dstV,
    fractional* srcV1,
    fractional* srcV2
);
```

**Arguments:**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numElems</td>
<td>number of elements in source vector</td>
</tr>
<tr>
<td>dstV</td>
<td>pointer to destination vector</td>
</tr>
<tr>
<td>srcV1</td>
<td>pointer to source one vector</td>
</tr>
<tr>
<td>srcV2</td>
<td>pointer to source two vector</td>
</tr>
</tbody>
</table>

**Return Value:** Pointer to base address of destination vector.

**Remarks:** This operation is also known as vector element-by-element multiplication.
This function can be computed in place.
This function can be self applicable.

**Source File:** vmul.s

**Function Profile:**

- **System resources usage:**
  - W0..W5 used, not restored
  - ACCA used, not restored
  - CORCON saved, used, restored

- **DO and REPEAT instruction usage:**
  - 1 level DO instructions
  - no REPEAT instructions

- **Program words (24-bit instructions):**
  - 14

- **Cycles (including C-function call and return overheads):**
  - 17 + 4(numElems)

### VectorNegate

**Description:** VectorNegate negates (changes the sign of) the values of the elements in the source vector, and places them in the destination vector.

**Include:**

dsp.h

**Prototype:**

```c
extern fractional* VectorNeg (
    int numElems,
    fractional* dstV,
    fractional* srcV
);
```

**Arguments:**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numElems</td>
<td>number of elements in source vector</td>
</tr>
<tr>
<td>dstV</td>
<td>pointer to destination vector</td>
</tr>
<tr>
<td>srcV</td>
<td>pointer to source vector</td>
</tr>
</tbody>
</table>

**Return Value:** Pointer to base address of destination vector.

**Remarks:**

- The negated value of 0x8000 is set to 0x7FFF.
- This function can be computed in place.

**Source File:** vneg.s
VectorNegate (Continued)

Function Profile:

System resources usage:
- W0..W5 used, not restored
- ACCA used, not restored
- CORCON saved, used, restored

DO and REPEAT instruction usage:
- 1 level DO instructions
- no REPEAT instructions

Program words (24-bit instructions):
- 16

Cycles (including C-function call and return overheads):
- \(19 + 4(\text{numElems})\)

VectorPower

Description:
VectorPower computes the power of a source vector as the sum of the squares of its elements.

Include:
dsp.h

Prototype:

```
extern fractional VectorPower (
    int numElems,
    fractional* srcV
);
```

Arguments:
- numElems number of elements in source vector
- srcV pointer to source vector

Return Value:
Value of the vector's power (sum of squares).

Remarks:
If the absolute value of the sum of squares is larger than \(1 - 2^{-15}\), this operation results in saturation.
This function can be self applicable.

Source File:
vpow.s

Function Profile:

System resources usage:
- W0..W2 used, not restored
- W4 used, not restored
- ACCA used, not restored
- CORCON saved, used, restored

DO and REPEAT instruction usage:
- no DO instructions
- 1 level REPEAT instructions

Program words (24-bit instructions):
- 12

Cycles (including C-function call and return overheads):
- \(16 + 2(\text{numElems})\)
### VectorScale

**Description:** VectorScale scales (multiplies) the values of all the elements in the source vector by a scale value, and places the result in the destination vector.

**Include:**

dsp.h

**Prototype:**

```c
extern fractional* VectorScale (  
    int numElems,  
    fractional* dstV,  
    fractional* srcV,  
    fractional sclVal  
);
```

**Arguments:**

- `numElems` number of elements in source vector
- `dstV` pointer to destination vector
- `srcV` pointer to source vector
- `sclVal` value by which to scale vector elements

**Return Value:** Pointer to base address of destination vector.

**Remarks:**

- `sclVal` must be a fractional number in 1.15 format.
- This function can be computed in place.

**Source File:**

vscl.s

**Function Profile:**

- System resources usage:
  - `W0..W5` used, not restored
  - `ACCA` used, not restored
  - `CORCON` saved, used, restored

- DO and REPEAT instruction usage:
  - 1 level DO instructions
  - no REPEAT instructions

- Program words (24-bit instructions):
  - 14

- Cycles (including C-function call and return overheads):
  - $18 + 3(numElems)$

### VectorSubtract

**Description:** VectorSubtract subtracts the value of each element in the source two vector from its counterpart in the source one vector, and places the result in the destination vector.

**Include:**


dsp.h

**Prototype:**

```c
extern fractional* VectorSubtract (  
    int numElems,  
    fractional* dstV,  
    fractional* srcV1,  
    fractional* srcV2  
);
```

**Arguments:**

- `numElems` number of elements in source vectors
- `dstV` pointer to destination vector
- `srcV1` pointer to source one vector (minuend)
- `srcV2` pointer to source two vector (subtrahend)

**Return Value:** Pointer to base address of destination vector.

**Remarks:**

- If the absolute value of $srcV1[n] - srcV2[n]$ is larger than $1-2^{-15}$, this operation results in saturation for the $n$-th element.
- This function can be computed in place.
- This function can be self applicable.
VectorZeroPad

**Description:**
VectorZeroPad copies the source vector into the beginning of the (already existing) destination vector, and then fills with zeros the remaining `numZeros` elements of destination vector:

\[
\text{dstV}[n] = \text{srcV}[n], \quad 0 \leq n < \text{numElems} \\
\text{dstV}[n] = 0, \quad \text{numElems} \leq n < \text{numElems} + \text{numZeros}
\]

**Include:**
dsp.h

**Prototype:**

\[
\text{extern fractional* VectorZeroPad (}
\text{int numElems,} \\
\text{int numZeros,} \\
\text{fractional* dstV,} \\
\text{fractional* srcV})
\]

**Arguments:**
- `numElems`: number of elements in source vector
- `numZeros`: number of elements to fill with zeros at the tail of destination vector
- `dstV`: pointer to destination vector
- `srcV`: pointer to source vector

**Return Value:**
Pointer to base address of destination vector.

**Remarks:**
The destination vector must already exist, with exactly `numElems` + `numZeros` number of elements.
This function can be computed in place. See Additional Remarks at the beginning of the section for comments on this mode of operation.
This function uses VectorCopy.

**Source File:**
vzpad.s
**VectorZeroPad (Continued)**

<table>
<thead>
<tr>
<th>Function Profile</th>
<th>System resources usage:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>W0..W6 used, not restored</td>
</tr>
<tr>
<td></td>
<td>plus resources from VectorCopy</td>
</tr>
</tbody>
</table>

DO and REPEAT instruction usage:
- no DO instructions
- 1 level REPEAT instructions
  - plus DO/REPEAT from VectorCopy

Program words (24-bit instructions):
- 13,  
  - plus program words from VectorCopy

Cycles (including C-function call and return overheads):
- 18 + numZeros  
  - plus cycles from VectorCopy

**Note:** In the description of VectorCopy, the number of cycles reported includes 3 cycles of C-function call overhead. Thus, the number of actual cycles from VectorCopy to add to VectorCorrelate is 3 less than whatever number is reported for a stand-alone VectorCopy.
2.4 WINDOW FUNCTIONS

A window is a vector with a specific value distribution within its domain \((0 \leq n < \text{numElems})\). The particular value distribution depends on the characteristics of the window being generated.

Given a vector, its value distribution may be modified by applying a window to it. In these cases, the window must have the same number of elements as the vector to modify.

Before a vector can be windowed, the window must be created. Window initialization operations are provided which generate the values of the window elements. For higher numerical precision, these values are computed in floating-point arithmetic, and the resulting quantities stored as 1.15 fractionals.

To avoid excessive overhead when applying a window operation, a particular window could be generated once and used many times during the execution of the program. Thus, it is advisable to store the window returned by any of the initialization operations in a permanent (static) vector.

2.4.1 User Considerations

a) All the window initialization functions have been designed to generate window vectors allocated in default RAM memory space (X-Data or Y-Data).

b) The windowing function is designed to operate on vectors allocated in default RAM memory space (X-Data or Y-Data).

c) It is recommended that the STATUS Register (SR) be examined after completion of each function call.

d) Since the window initialization functions are implemented in C, consult the electronic documentation included in the release for up-to-date cycle count information.

2.4.2 Individual Functions

In what follows, the individual functions implementing window operations are described.

**BartlettInit**

**Description:** BartlettInit initializes a Bartlett window of length \(\text{numElems}\).

**Include:**

dsp.h

**Prototype:**

```c
extern fractional* BartlettInit (int \text{numElems}, fractional* \text{window});
```

**Arguments:**

- \text{numElems} number of elements in window
- \text{window} pointer to window to be initialized

**Return Value:** Pointer to base address of initialized window.

**Remarks:** The window vector must already exist, with exactly \(\text{numElems}\) number of elements.

**Source File:** initbart.c
### BlackmanInit

**Description:**
BlackmanInit initializes a Blackman (3 terms) window of length `numElems`.

**Include:**
dsp.h

**Prototype:**
```
extern fractional* BlackmanInit (
    int numElems,
    fractional* window
);
```

**Arguments:**
- `numElems` number of elements in window
- `window` pointer to window to be initialized

**Return Value:**
Pointer to base address of initialized window.

**Remarks:**
The window vector must already exist, with exactly `numElems` number of elements.

**Source File:**
initblck.c

**Function Profile:**
- **System resources usage:**
  - W0..W7 used, not restored
  - W8..W14 saved, used, not restored

  **DO and REPEAT instruction usage:**
  None

  **Program words (24-bit instructions):**
  See the file “Readme for dsPIC Language Tools Libraries.txt” for this information.

  **Cycles (including C-function call and return overheads):**
  See the file “Readme for dsPIC Language Tools Libraries.txt” for this information.
HammingInit

Description: HammingInit initializes a Hamming window of length numElems.

Include: dsp.h

Prototype: extern fractional* HammingInit (  
    int numElems,  
    fractional* window  
);

Arguments: numElems number of elements in window  
            window pointer to window to be initialized

Return Value: Pointer to base address of initialized window.

Remarks: The window vector must already exist, with exactly numElems number  
         of elements.

Source File: inithamm.c

Function Profile: System resources usage:  
                    W0..W7 used, not restored  
                    W8..W14 saved, used, not restored

            DO and REPEAT instruction usage:  
                                               None

            Program words (24-bit instructions):  
                                                       See the file “readme.txt” in pic30_tools\src\dsp for this information.

            Cycles (including C-function call and return overheads):  
                                                       See the file “readme.txt” in pic30_tools\src\dsp for this information.

HanningInit

Description: HanningInit initializes a Hanning window of length numElems.

Include: dsp.h

Prototype: extern fractional* HanningInit (  
    int numElems,  
    fractional* window  
);

Arguments: numElems number of elements in window  
            window pointer to window to be initialized

Return Value: Pointer to base address of initialized window.

Remarks: The window vector must already exist, with exactly numElems number  
         of elements.

Source File: inithann.c

Function Profile: System resources usage:  
                    W0..W7 used, not restored  
                    W8..W14 saved, used, not restored

            DO and REPEAT instruction usage:  
                                               None

            Program words (24-bit instructions):  
                                                       See the file “readme.txt” in pic30_tools\src\dsp for this information.

            Cycles (including C-function call and return overheads):  
                                                       See the file “readme.txt” in pic30_tools\src\dsp for this information.
KaiserInit

Description: KaiserInit initializes a Kaiser window with shape determined by argument betaVal and of length numElems.

Include: dsp.h

Prototype: extern fractional* KaiserInit (int numElems, fractional* window, float betaVal);

Arguments: numElems number of elements in window window pointer to window to be initialized betaVal window shaping parameter

Return Value: Pointer to base address of initialized window.

Remarks: The window vector must already exist, with exactly numElems number of elements.

Source File: initkais.c

Function Profile: System resources usage:

W0..W7 used, not restored
W8..W14 saved, used, not restored

DO and REPEAT instruction usage: None

Program words (24-bit instructions):
See the file “readme.txt” in pic30_tools\src\dsp for this information.

Cycles (including C-function call and return overheads):
See the file “readme.txt” in pic30_tools\src\dsp for this information.

VectorWindow

Description: VectorWindow applies a window to a given source vector, and stores the resulting windowed vector in a destination vector.

Include: dsp.h

Prototype: extern fractional* VectorWindow (int numElems, fractional* dstV, fractional* srcV, fractional* window);

Arguments: numElems number of elements in source vector dstV pointer to destination vector srcV pointer to source vector window pointer to initialized window

Return Value: Pointer to base address of destination vector.

Remarks: The window vector must have already been initialized, with exactly numElems number of elements.
This function can be computed in place.
This function can be self applicable.
This function uses VectorMultiply.

Source File: dowindow.s
### Function Profile:

**System resources usage:**
- resources from `VectorMultiply`

**DO and REPEAT instruction usage:**
- no DO instructions
- no REPEAT instructions,
- plus DO/REPEAT from `VectorMultiply`

**Program words (24-bit instructions):**
- 3,
- plus program words from `VectorMultiply`

**Cycles (including C-function call and return overheads):**
- 9,
- plus cycles from `VectorMultiply`.

**Note:** In the description of `VectorMultiply` the number of cycles reported includes 3 cycles of C-function call overhead. Thus, the number of actual cycles from `VectorMultiply` to add to `VectorWindow` is 3 less than whatever number is reported for a stand-alone `VectorMultiply`. 
2.5 MATRIX FUNCTIONS

This section presents the concept of a fractional matrix, as considered by the DSP Library, and describes the individual functions which perform matrix operations.

2.5.1 Fractional Matrix Operations

A fractional matrix is a collection of numerical values, the matrix elements, allocated contiguously in memory, with the first element at the lowest memory address. One word of memory (two bytes) is used to store the value of each element, and this quantity must be interpreted as a fractional number represented in 1.15 format.

A pointer addressing the first element of the matrix is used as a handle which provides access to each of the matrix values. The address of the first element is referred to as the base address of the matrix. Because each element of the matrix is 16 bits, the base address must be aligned to an even address.

The two dimensional arrangement of a matrix is emulated in the memory storage area by placing its elements organized in row major order. Thus, the first value in memory is the first element of the first row. It is followed by the rest of the elements of the first row. Then, the elements of the second row are stored, and so on, until all the rows are in memory. This way, the element at row \( r \) and column \( c \) of a matrix with \( R \) rows and \( C \) columns is located from the matrix base address \( BA \) at:

\[
BA + 2(C(r - 1) + c - 1), \quad \text{for} \quad 1 \leq r \leq R, \quad 1 \leq c \leq C.
\]

Note that the factor of 2 is used because of the byte addressing capabilities of the 16-bit device.

Unary and binary fractional matrix operations are implemented in this library. The operand matrix in a unary operation is called the source matrix. In a binary operation the first operand is referred to as the source one matrix, and the second matrix as the source two matrix. Each operation applies some computation to one or several elements of the source matrix(ces). The operations result in a matrix, referred to as the destination matrix.

Some operations resulting in a matrix allow computation in place. This means the results of the operation is placed back into the source matrix (or the source one matrix for a binary operation). In this case, the destination matrix is said to (physically) replace the source (one) matrix. If an operation can be computed in place, it is indicated as such in the comments provided with the function description.

For some binary operations, the two operands can be the same (physical) source matrix, which means the operation is applied to the source matrix and itself. If this type of computation is possible for a given operation, it is indicated as such in the comments provided with the function description.

Some operations can be self applicable and computed in place.
All the fractional matrix operations in this library take as arguments the number of rows and the number of columns of the operand matrix(ces). Based on the values of these argument the following assumptions are made:

a) The sum of sizes of all the matrices involved in a particular operation falls within the range of available data memory for the target device.

b) In the case of binary operations the number of rows and columns of the operand matrices must obey the rules of vector algebra; i.e., for matrix addition and subtraction the two matrices must have the same number of rows and columns, while for matrix multiplication, the number of columns of the first operand must be the same as the number of rows of the second operand. The source matrix to the inversion operation must be square (the same number of rows as of columns), and non-singular (its determinant different than zero).

c) The destination matrix must be large enough to accept the results of an operation.

2.5.2 User Considerations

a) No boundary checking is performed by these functions. Out of range dimensions (including zero row and/or zero column matrices) as well as nonconforming use of source matrix sizes in binary operations may produce unexpected results.

b) The matrix addition and subtraction operations could lead to saturation if the sum of corresponding elements in the source(s) matrix(ces) is greater than $1-2^{-15}$ or smaller than $-1$.

c) The matrix multiplication operation could lead to saturation if the sum of products of corresponding row and column sets results in a value greater than $1-2^{-15}$ or smaller than $-1$.

d) It is recommended that the STATUS Register (SR) is examined after completion of each function call. In particular, users can inspect the SA, SB and SAB flags after the function returns to determine if saturation occurred.

e) All the functions have been designed to operate on fractional matrices allocated in default RAM memory space (X-Data or Y-Data).

f) Operations which return a destination matrix can be nested, so that for instance if:

\[
\begin{align*}
  a &= \text{Op1} (b, c), \text{ with } b = \text{Op2} (d), \text{ and } c = \text{Op3} (e, f), \\
  a &= \text{Op1} (\text{Op2} (d), \text{Op3} (e, f))
\end{align*}
\]

2.5.3 Additional Remarks

The description of the functions limits its scope to what could be considered the regular usage of these operations. However, since no boundary checking is performed during computation of these functions, you have the freedom to interpret the operation and its results as it fits some particular needs.

For instance, while computing the MatrixMultiply function, the dimensions of the intervening matrices does not necessarily need to be \{numRows1, numCols1Rows2\} for source one matrix, \{numCols1Rows2, numCols2\} for source two matrix, and \{numRows1, numCols2\} for destination matrix. In fact, all that is needed is that their sizes are large enough so that during computation the pointers do no exceed over their memory range.

As another example, when a source matrix of dimension \{numRows, numCols\} is transposed, the destination matrix has dimensions \{numCols, numRows\}. Thus, properly speaking the operation can be computed in place only if source matrix is square. Nevertheless, the operation can be successfully applied in place to non square matrices; all that needs to be kept in mind is the implicit change of dimensions.
Other possibilities can be exploited from the fact that no boundary checking is performed.

### 2.5.4 Individual Functions

In what follows, the individual functions implementing matrix operations are described.

#### MatrixAdd

<table>
<thead>
<tr>
<th>Description:</th>
<th>MatrixAdd adds the value of each element in the source one matrix with its counterpart in the source two matrix, and places the result in the destination matrix.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>dsp.h</td>
</tr>
<tr>
<td>Prototype:</td>
<td><code>extern fractional* MatrixAdd (int numRows, int numCols, fractional* dstM, fractional* srcM1, fractional* srcM2);</code></td>
</tr>
</tbody>
</table>
| Arguments:   | **numRows** number of rows in source matrices  
**numCols** number of columns in source matrices  
**dstM** pointer to destination matrix  
**srcM1** pointer to source one matrix  
**srcM2** pointer to source two matrix |
| Return Value:| Pointer to base address of destination matrix.                                                                                     |
| Remarks:     | If the absolute value of \( srcM1[r][c] + srcM2[r][c] \) is larger than \( 1 - 2^{-15} \), this operation results in saturation for the \( (r,c) \)-th element.  
This function can be computed in place.  
This function can be self applicable. |
| Source File: | madd.s                                                                                                                               |
| Function Profile: | System resources usage:  
- **W0..W4** used, not restored  
- **ACCA** used, not restored  
- **CORCON** saved, used, restored  
  
- **DO** and **REPEAT** instruction usage:  
  - 1 level **DO** instructions  
  - no **REPEAT** instructions  
  
- Program words (24-bit instructions):  
  - 14  
  
- Cycles (including C-function call and return overheads):  
  - \( 20 + 3(\text{numRows} \times \text{numCols}) \) |
MatrixMultiply

Description: MatrixMultiply performs the matrix multiplication between the source one and source two matrices, and places the result in the destination matrix. Symbolically:

\[ \text{dstM}[i][j] = \sum_{k} (\text{srcM1}[i][k])(\text{srcM2}[k][j]) \]

where:

0 \leq i < \text{numRows1} \\
0 \leq j < \text{numCols2} \\
0 \leq k < \text{numCols1Rows2}

Include: dsp.h

Prototype: extern fractional* MatrixMultiply ( 
    int numRows1, 
    int numCols1Rows2, 
    int numCols2, 
    fractional* dstM, 
    fractional* srcM1, 
    fractional* srcM2 
); 

Arguments: 

numRows1 number of rows in source one matrix 
numCols1Rows2 number of columns in source one matrix; which must be the same as number of rows in source two matrix 
numCols2 number of columns in source two matrix 
dstM pointer to destination matrix 
srcM1 pointer to source one matrix 
srcM2 pointer to source two matrix 

Return Value: Pointer to base address of destination matrix.

Remarks: If the absolute value of

\[ \sum_{k} (\text{srcM1}[i][k])(\text{srcM2}[k][j]) \]

is larger than 1-2\(^{-15}\), this operation results in saturation for the \((i,j)\)-th element.

If the source one matrix is squared, then this function can be computed in place and can be self applicable. See Additional Remarks at the beginning of the section for comments on this mode of operation.

Source File: mmul.s

Function Profile: System resources usage:

- W0..W7 used, not restored
- W8..W13 saved, used, restored
- ACCA used, not restored
- CORCON saved, used, restored

DO and REPEAT instruction usage:

2 level DO instructions
no REPEAT instructions

Program words (24-bit instructions):
35

Cycles (including C-function call and return overheads):

\[ 36 + \text{numRows1} \times (8 + \text{numCols2} \times (7 + 4 \times \text{numCols1Rows2})) \]
MatrixScale

Description: MatrixScale scales (multiplies) the values of all elements in the source matrix by a scale value, and places the result in the destination matrix.

Include: dsp.h

Prototype: extern fractional* MatrixScale (  
    int numRows,
    int numCols,
    fractional* dstM,
    fractional* srcM,
    fractional sclVal
);

Arguments: numRows number of rows in source matrix
numCols number of columns in source matrix
dstM pointer to destination matrix
srcM pointer to source matrix
sclVal value by which to scale matrix elements

Return Value: Pointer to base address of destination matrix.

Remarks: This function can be computed in place.

Source File: mscl.s

Function Profile: System resources usage:
- W0..W5 used, not restored
- ACCA used, not restored
- CORCON saved, used, restored

DO and REPEAT instruction usage:
- 1 level DO instructions
- no REPEAT instructions

Program words (24-bit instructions):
- 14

Cycles (including C-function call and return overheads):
- 20 + 3(numRows * numCols)

MatrixSubtract

Description: MatrixSubtract subtracts the value of each element in the source two matrix from its counterpart in the source one matrix, and places the result in the destination matrix.

Include: dsp.h

Prototype: extern fractional* MatrixSubtract (  
    int numRows,
    int numCols,
    fractional* dstM,
    fractional* srcM1,
    fractional* srcM2
);

Arguments: numRows number of rows in source matrix(ces)
numCols number of columns in source matrix(ces)
dstM pointer to destination matrix
srcM1 pointer to source one matrix (minuend)
srcM2 pointer to source two matrix (subtrahend)

Return Value: Pointer to base address of destination matrix.
MatrixSubtract (Continued)

Remarks: If the absolute value of $srcM1_{[r][c]} - srcM2_{[r][c]}$ is larger than $1-2^{15}$, this operation results in saturation for the $(r,c)$-th element.
This function can be computed in place.
This function can be self applicable.

Source File: msub.s

Function Profile:

System resources usage:

- W0..W4 used, not restored
- ACCA used, not restored
- ACCB used, not restored
- CORCON saved, used, restored

DO and REPEAT instruction usage:

- 1 level DO instructions
- no REPEAT instructions

Program words (24-bit instructions):

- 15

Cycles (including C-function call and return overheads):

$$20 + 4(numRows \times numCols)$$

MatrixTranspose

Description: MatrixTranspose transposes the rows by the columns in the source matrix, and places the result in destination matrix. In effect:
$$dstM[i][j] = srcM[j][i],$$
$$0 \leq i < numRows, 0 \leq j < numCols.$$

Include:

dsp.h

Prototype:

extern fractional* MatrixTranspose (
    int numRows,
    int numCols,
    fractional* dstM,
    fractional* srcM
);  

Arguments:

- numRows: number of rows in source matrix
- numCols: number of columns in source matrix
- dstM: pointer to destination matrix
- srcM: pointer to source matrix

Return Value: Pointer to base address of destination matrix.

Remarks: If the source matrix is square, this function can be computed in place.
See Additional Remarks at the beginning of the section for comments on this mode of operation.

Source File: mtrp.s

Function Profile:

System resources usage:

- W0..W5 used, not restored

DO and REPEAT instruction usage:

- 2 level DO instructions
- no REPEAT instructions

Program words (24-bit instructions):

- 14

Cycles (including C-function call and return overheads):

$$16 + numCols \times (6 + (numRows-1) \times 3)$$
2.5.5 Matrix Inversion

The result of inverting a non-singular, square, fractional matrix is another square matrix (of the same dimension) whose element values are not necessarily constrained to the discrete fractional set \{-1, ..., 1-2^{-15}\}. Thus, no matrix inversion operation is provided for fractional matrices.

However, since matrix inversion is a very useful operation, an implementation based on floating-point number representation and arithmetic is provided within the DSP Library. Its description follows.

MatrixInvert

Description: MatrixInvert computes the inverse of the source matrix, and places the result in the destination matrix.

Include: dsp.h

Prototype: extern float* MatrixInvert (int numRowsCols, float* dstM, float* srcM, float* pivotFlag, int* swappedRows, int* swappedCols);

Arguments:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numRowsCols</td>
<td>number of rows and columns in (square) source matrix</td>
</tr>
<tr>
<td>dstM</td>
<td>pointer to destination matrix</td>
</tr>
<tr>
<td>srcM</td>
<td>pointer to source matrix</td>
</tr>
</tbody>
</table>

Required for internal use:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pivotFlag</td>
<td>pointer to a length numRowsCols vector</td>
</tr>
<tr>
<td>swappedRows</td>
<td>pointer to a length numRowsCols vector</td>
</tr>
<tr>
<td>swappedCols</td>
<td>pointer to a length numRowsCols vector</td>
</tr>
</tbody>
</table>

Return Value: Pointer to base address of destination matrix, or NULL if source matrix is singular.

Remarks:

Even though the vectors pivotFlag, swappedRows, and swappedCols, are for internal use only, they must be allocated prior to calling this function.

If source matrix is singular (determinant equal to zero) the matrix does not have an inverse. In this case the function returns NULL.

This function can be computed in place.

Source File: minv.s (assembled from C code)

Function Profile:

System resources usage:

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0..W7</td>
<td>used, not restored</td>
</tr>
<tr>
<td>W8, W14</td>
<td>saved, used, restored</td>
</tr>
</tbody>
</table>

DO and REPEAT instruction usage:

None

Program words (24-bit instructions):

See the file “readme.txt” in pic30_tools\src\dsp for this information.

Cycles (including C-function call and return overheads):

See the file “readme.txt” in pic30_tools\src\dsp for this information.
2.6 FILTERING FUNCTIONS

This section presents the concept of a fractional filter, as considered by the DSP Library, and describes the individual functions which perform filter operations. The user may refer to example projects that utilize the DSP library filtering functions, in order to ascertain proper usage of functions. MPLAB IDE-based example projects/workspaces have been provided in the installation folder of the MPLAB C30 toolsuite.

2.6.1 Fractional Filter Operations

Filtering the data sequence represented by fractional vector \( x[n] \) (\( 0 \leq n < N \)) is equivalent to solving the difference equation:

\[
y[n] + \sum_{p=1}^{P-1} (-a[p])(y[n-p]) = \sum_{m=0}^{M-1} (b[m])(x[n-m])
\]

for every \( n \)th sample, which results into the filtered data sequence \( y[n] \). In this sense, the fractional filter is characterized by the fractional vectors \( a[p] \) (\( 0 \leq p < P \)) and \( b[m] \) (\( 0 \leq m < M \)), referred to as the set of filter coefficients, which are designed to induce some pre-specified changes in the signal represented by the input data sequence.

When filtering it is important to know and manage the past history of the input and output data sequences \( (x[n], -M + 1 \leq n < 0) \) and \( (y[n], -P + 1 \leq n < 0) \), which represent the initial conditions of the filtering operation. Also, when repeatedly applying the filter to contiguous sections of the input data sequence it is necessary to remember the final state of the last filtering operation \( (x[n], N - M + 1 \leq n < N - 1) \) and \( (y[n], N - P + 1 \leq n < N - 1) \). This final state is then taken into consideration for the calculations of the next filtering stage. Accounting for the past history and current state is required in order to perform a correct filtering operation.

The management of the past history and current state of a filtering operation is commonly implemented via additional sequences (also fractional vectors), referred to as the filter delay line. Prior to applying the filter operation, the delay describes the past history of the filter. After performing the filtering operation, the delay contains a set of the most recently filtered data samples, and of the most recent output samples. (Note that to ensure correct operation of a particular filter implementation, it is advisable to initialize the delay values to zero by calling the corresponding initialization function.)

In the filter implementations provided with the DSP Library the input data sequence is referred to as the sequence of source samples, while the resulting filtered sequence is called the destination samples. The filter coefficients \( (a, b) \) and delay are usually thought of as making up a filter structure. In all filter implementations, the input and output data samples may be allocated in default RAM memory space (X-Data or Y-Data). Filter coefficients may reside either in X-Data memory or program memory, and filter delay values must be accessed only from Y-Data.

2.6.2 FIR and IIR Filter Implementations

The properties of a filter depend on the value distribution of its coefficients. In particular, two types of filters are of special interest: Finite Impulse Response (FIR) filters, for which \( a[m] = 0 \) when \( 1 \leq m < M \), and Infinite Impulse Response (IIR) filters, those such that \( a[0] \neq 0 \), and \( a[m] \neq 0 \) for some \( m \) in \( \{1, \ldots, M\} \). Other classifications within the FIR and IIR filter families account for the effects that the operation induces on input data sequences.

Furthermore, even though filtering consists on solving the difference equation stated above, several implementations are available which are more efficient than direct computation of the difference equation. Also, some other implementations are designed to execute the filtering operation under the constrains imposed by fractional arithmetic.
All these considerations lead to a proliferation of filtering operations, of which a subset is provided by the DSP Library.

### 2.6.3 Single Sample Filtering

The filtering functions provided in the DSP Library are designed for block processing. Each filter function accepts an argument named `numSamps` which indicates the number of words of input data (block size) to operate on. If single sample filtering is desired, you may set `numSamps` to 1. This will have the effect of filtering one input sample, and the function will compute a single output sample from the filter.

### 2.6.4 User Considerations

All the fractional filtering operations in this library rely on the values of either input parameters or data structure elements to specify the number of samples to process, and the sizes of the coefficients and delay vectors. Based on these values the following assumptions are made:

- a) The sum of sizes of all the vectors (sample sequences) involved in a particular operation falls within the range of available data memory for the target device.
- b) The destination vector must be large enough to accept the results of an operation.
- c) No boundary checking is performed by these functions. Out of range sizes (including zero length vectors) as well as nonconforming use of source vectors and coefficient sets may produce unexpected results.
- d) It is recommended that the STATUS Register (SR) is examined after completion of each function call. In particular, users can inspect the SA, SB and SAB flags after the function returns to determine if saturation occurred.
- e) Operations which return a destination vector can be nested, so that for instance:
  
a = Op1 (b, c), with b = Op2 (d), and c = Op3 (e, f), then

  
a = Op1 (Op2 (d), Op3 (e, f))
2.6.5 Individual Functions

In what follows, the individual functions implementing filtering operations are
described. For further discussions on digital filters, please consult Alan Oppenheim and
Ronald Schafer’s “Discrete-Time Signal Processing”, Prentice Hall, 1989. For imple-
mentation details of Least Mean Square FIR filters, please refer to T. Hsia’s “Conver-
gence Analysis of LMS and NLMS Adaptive Algorithms”, Proc. ICASSP, pp. 667-670,
1983, as well as Sangil Park and Garth Hillman’s “On Acoustic-Echo Cancellation
Implementation with Multiple Cascadable Adaptive FIR Filter Chips”, Proc. ICASSP,
1989.

FIRStruct

Structure: FIRStruct describes the filter structure for any of the FIR filters.
Include: dsp.h
Declaration: typedef struct {
int numCoeffs;
fractional* coeffsBase;
fractional* coeffsEnd;
int coeffsPage;
fractional* delayBase;
fractional* delayEnd;
fractional* delay;
} FIRStruct;
Parameters: numCoeffs number of coefficients in filter (also M)
coeffsBase base address for filter coefficients (also h)
coeffsEnd end address for filter coefficients
coeffsPage coefficients buffer page number
delayBase base address for delay buffer
delayEnd end address for delay buffer
delay current value of delay pointer (also d)
Remarks: Number of coefficients in filter is M.
Coefficients, h[m], defined in 0 ≤ m < M, either within X-Data or program
memory.
Delay buffer d[m], defined in 0 ≤ m < M, only in Y-Data.
If coefficients are stored in X-Data space, coeffsBase points to the
actual address where coefficients are allocated. If coefficients are
stored in program memory, coeffsBase is the offset from the program
page boundary containing the coefficients to the address in the page
where coefficients are allocated. This latter value can be calculated
using the inline assembly operator psvoffset().
coeffsEnd is the address in X-Data space (or offset if in program
memory) of the last byte of the filter coefficients buffer.
If coefficients are stored in X-Data space, coeffsPage must be set to
0xFF00 (defined value COEFFS_IN_DATA). If coefficients are stored in
program memory, it is the program page number containing the coeffi-
cients. This latter value can be calculated using the inline assembly
operator psvpage().
delayBase points to the actual address where the delay buffer is allo-
cated.
delayEnd is the address of the last byte of the filter delay buffer.
When the coefficients and delay buffers are implemented as circular increasing modulo buffers, both `coeffsBase` and `delayBase` must be aligned to a `zero` power of two address (`coeffsEnd` and `delayEnd` are odd addresses). Whether these buffers are implemented as circular increasing modulo buffers or not is indicated in the remarks section of each FIR filter function description.

When the coefficients and delay buffers are not implemented as circular (increasing) modulo buffers, `coeffsBase` and `delayBase` do not need to be aligned to a `zero` power of two address, and the values of `coeffsEnd` and `delayEnd` are ignored within the particular FIR Filter function implementation.

### FIR

**Description:**  
FIR applies an FIR filter to the sequence of source samples, places the results in the sequence of destination samples, and updates the delay values.

**Include:**  
dsp.h

**Prototype:**  
```c
extern fractional* FIR (  
    int numSamps,  
    fractional* dstSamps,  
    fractional* srcSamps,  
    FIRStruct* filter  
);
```

**Arguments:**  
- `numSamps` number of input samples to filter (also N)
- `dstSamps` pointer to destination samples (also y)
- `srcSamps` pointer to source samples (also x)
- `filter` pointer to FIRStruct filter structure

**Return Value:**  
Pointer to base address of destination samples.

**Remarks:**  
Number of coefficients in filter is M.  
Coefficients, $h[m]$, defined in $0 \leq m < M$, implemented as a circular increasing modulo buffer.  
Delay, $d[m]$, defined in $0 \leq m < M$, implemented as a circular increasing modulo buffer.  
Source samples, $x[n]$, defined in $0 \leq n < N$.  
Destination samples, $y[n]$, defined in $0 \leq n < N$.  
(See also FIRStruct, FIRStructInit and FIRDelayInit.)

**Source File:**  
`fir.s`
FIR (Continued)

Function Profile:

System resources usage:

- W0..W6 used, not restored
- W8, W10 saved, used, restored
- ACCA used, not restored
- CORCON saved, used, restored
- MODCON saved, used, restored
- XMODSTRT saved, used, restored
- XMODEND saved, used, restored
- YMODSTRT saved, used, restored
- PSVPAG saved, used, restored (only if coefficients in P memory)

DO and REPEAT instruction usage:
- 1 level DO instructions
- 1 level REPEAT instructions

Program words (24-bit instructions):
- 55

Cycles (including C-function call and return overheads):
- 53 + N(4+M), or
- 56 + N(8+M) if coefficients in P memory.

Example
Please refer to the MPLAB C30 installation folder for a sample project demonstrating the use of this function.

FIRDecimate

Description:
FIRDecimate decimates the sequence of source samples at a rate of R to 1; or equivalently, it downsamples the signal by a factor of R. Effectively, $y[n] = x[Rn]$. To diminish the effect of aliasing, the source samples are first filtered and then downsampled. The decimated results are stored in the sequence of destination samples, and the delay values updated.

Include:
dsp.h

Prototype:

```c
extern fractional* FIRDecimate(
    int numSamps,
    fractional* dstSamps,
    fractional* srcSamps,
    FIRStruct* filter,
    int rate
);
```

Arguments:
- numSamps number of output samples (also $N = Rp$, $p$ integer)
- dstSamp pointer to destination samples (also $y$)
- srcSamps pointer to source samples (also $x$)
- filter pointer to FIRStruct filter structure
- rate rate of decimation (downsampling factor, also $R$)

Return Value:
Pointer to base address of destination samples.
**Remarks:**
Number of coefficients in filter is M, with M an integer multiple of R. Coefficients, h[m], defined in 0 ≤ m < M, not implemented as a circular modulo buffer.
Delay, d[m], defined in 0 ≤ m < M, not implemented as a circular modulo buffer.
Source samples, x[n], defined in 0 ≤ n < NR.
Destination samples, y[n], defined in 0 ≤ n < N.

(See also FIRStruct, FIRStructInit, and FIRDelayInit.)

**Source File:**
firdecim.s

**Function Profile:**
System resources usage:
- W0..W7 used, not restored
- W8..W12 saved, used, restored
- ACCA used, not restored
- CORCON saved, used, restored
- PSVPAG saved, used, restored (only if coefficients in P memory)

**DO and REPEAT instruction usage:**
- 1 level DO instructions
- 1 level REPEAT instructions

**Program words (24-bit instructions):**
48

**Cycles (including C-function call and return overheads):**
45 + N(10 + 2M), or
48 + N(13 + 2M) if coefficients in P memory.

---

**FIRDelayInit**

**Description:**
FIRDelayInit initializes to zero the delay values in an FIRStruct filter structure.

**Include:**
dsp.h

**Prototype:**
extern void FIRDelayInit (
   FIRStruct* filter
);

**Arguments:**
- filter pointer to FIRStruct filter structure.

**Remarks:**
See description of FIRStruct structure above.

**Note:** FIR interpolator's delay is initialized by function FIRInterpDelayInit.

**Source File:**
firdelay.s

**Function Profile:**
System resources usage:
- W0..W2 used, not restored

**DO and REPEAT instruction usage:**
- no DO instructions
- 1 level REPEAT instructions

**Program words (24-bit instructions):**
7

**Cycles (including C-function call and return overheads):**
11 + M
### FIRInterpolate

**Description:**
FIRInterpolate interpolates the sequence of source samples at a rate of 1 to R; or equivalently, it upsamples the signal by a factor of R. Effectively, \( y[n] = x[n/R] \).

To diminish the effect of aliasing, the source samples are first upsampled and then filtered. The interpolated results are stored in the sequence of destination samples, and the delay values updated.

**Include:**
dsp.h

**Prototype:**

```c
extern fractional* FIRInterpolate (  
    int numSamps,  
    fractional* dstSamps,  
    fractional* srcSamps,  
    FIRStruct* filter,  
    int rate  
);
```

**Arguments:**
- `numSamps` number of input samples (also \( N = Rp \), \( p \) integer)
- `dstSamps` pointer to destination samples (also \( y \))
- `srcSamps` pointer to source samples (also \( x \))
- `filter` pointer to `FIRStruct` filter structure
- `rate` rate of interpolation (upsampling factor, also \( R \))

**Return Value:**
Pointer to base address of destination samples.

**Remarks:**
Number of coefficients in filter is \( M \), with \( M \) an integer multiple of \( R \).

Coefficients, \( h[m] \), defined in \( 0 \leq m < M \), not implemented as a circular modulo buffer.

Delay, \( d[m] \), defined in \( 0 \leq m < M/R \), not implemented as a circular modulo buffer.

Source samples, \( x[n] \), defined in \( 0 \leq n < N \).

Destination samples, \( y[n] \), defined in \( 0 \leq n < NR \).

(See also `FIRStruct`, `FIRStructInit`, and `FIRInterpDelayInit`.)

**Source File:**
firinter.s

**Function Profile:**
System resources usage:
- `W0..W7` used, not restored
- `W8..W13` saved, used, restored
- `ACCA` used, not restored
- `CORCON` saved, used, restored
- `PSVPAG` saved, used, restored (only if coefficients in P memory)

\( \text{DO} \) and \( \text{REPEAT} \) instruction usage:
- 2 level \( \text{DO} \) instructions
- 1 level \( \text{REPEAT} \) instructions

Program words (24-bit instructions):
- 63

Cycles (including C-function call and return overheads):
- \( 45 + 6(M/R) + N(14 + M/R + 3M + 5R) \), or
- \( 48 + 6(M/R) + N(14 + M/R + 4M + 5R) \) if coefficients in P memory.
**FIRInterpDelayInit**

**Description:**
FIRInterpDelayInit initializes to zero the delay values in an FIRStruct filter structure, optimized for use with an FIR interpolating filter.

**Include:**
dsp.h

**Prototype:**
extern void FIRDelayInit (
    FIRStruct* filter,
    int rate
);  

**Arguments:**
filter: pointer to FIRStruct filter structure
rate: rate of interpolation (upsampling factor, also R)

**Remarks:**
Delay, \( d[m] \), defined in \( 0 \leq m < M/R \), with \( M \) the number of filter coefficients in the interpolator.
See description of FIRStruct structure above.

**Source File:**
firintdl.s

**Function Profile:**
System resources usage:
- W0..W4 used, not restored

- DO and REPEAT instruction usage:
  - no DO instructions
  - 1 level REPEAT instructions

Program words (24-bit instructions):
- 13

- Cycles (including C-function call and return overheads):
  - \( 10 + 7M/R \)

---

**FIRLattice**

**Description:**
FIRLattice uses a lattice structure implementation to apply an FIR filter to the sequence of source samples. It then places the results in the sequence of destination samples, and updates the delay values.

**Include:**
dsp.h

**Prototype:**
extern fractional* FIRLattice (  
    int numSamps,
    fractional* dstSamps,
    fractional* srcSamps,
    FIRStruct* filter
);  

**Arguments:**
numSamps: number of input samples to filter (also N)
dstSamps: pointer to destination samples (also y)
srcSamps: pointer to source samples (also x)
filter: pointer to FIRStruct filter structure

**Return Value:**
Pointer to base address of destination samples.

**Remarks:**
Number of coefficients in filter is M.
Lattice coefficients, \( k[m] \), defined in \( 0 \leq m < M \), not implemented as a circular modulo buffer.
Delay, \( d[m] \), defined in \( 0 \leq m < M \), not implemented as a circular modulo buffer.
Source samples, \( x[n] \), defined in \( 0 \leq n < N \).
Destination samples, \( y[n] \), defined in \( 0 \leq n < N \).
(See also FIRStruct, FIRStructInit and FIRDelayInit.)

**Source File:**
firlatt.s
## FIRLattice (Continued)

### Function Profile:

<table>
<thead>
<tr>
<th>System resources usage:</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0..W7 used, not restored</td>
</tr>
<tr>
<td>W8..W12 saved, used, restored</td>
</tr>
<tr>
<td>ACCA used, not restored</td>
</tr>
<tr>
<td>ACCB used, not restored</td>
</tr>
<tr>
<td>CORCON saved, used, restored</td>
</tr>
<tr>
<td>PSVPAG saved, used, restored (only if coefficients in P memory)</td>
</tr>
</tbody>
</table>

### DO and REPEAT instruction usage:

- 2 level DO instructions
- no REPEAT instructions

### Program words (24-bit instructions):

50

### Cycles (including C-function call and return overheads):

- $41 + N(4 + 7M)$
- $44 + N(4 + 8M)$ if coefficients in P memory

---

## FIRLMS

### Description:

FIRLMS applies an adaptive FIR filter to the sequence of source samples, stores the results in the sequence of destination samples, and updates the delay values. The filter coefficients are also updated, at a sample-per-sample basis, using a Least Mean Square algorithm applied according to the values of the reference samples.

### Include:

dsp.h

### Prototype:

```c
extern fractional* FIRLMS (  
    int numSamps,  
    fractional* dstSamps,  
    fractional* srcSamps,  
    FIRStruct* filter,  
    fractional* refSamps,  
    fractional muVal  
) ;
```

### Arguments:

- `numSamps` number of input samples (also N)
- `dstSamps` pointer to destination samples (also y)
- `srcSamps` pointer to source samples (also x)
- `filter` pointer to FIRStruct filter structure
- `refSamps` pointer to reference samples (also r)
- `muVal` adapting factor (also mu)

### Return Value:

Pointer to base address of destination samples.
Remarks: Number of coefficients in filter is M. Coefficients, \( h[m] \), defined in \( 0 \leq m < M \), implemented as a circular increasing modulo buffer. Delay, \( d[m] \), defined in \( 0 \leq m < M - 1 \), implemented as a circular increasing modulo buffer. Source samples, \( x[n] \), defined in \( 0 \leq n < N \). Reference samples, \( r[n] \), defined in \( 0 \leq n < N \). Destination samples, \( y[n] \), defined in \( 0 \leq n < N \). Adaptation: 
\[
h_m[n] = h_m[n-1] + \mu (r[n] - y[n]) * x[n-m],
\]
for \( 0 \leq n < N, 0 \leq m < M \). The operation could result in saturation if the absolute value of \( (r[n] - y[n]) \) is greater than or equal to one. Filter coefficients must not be allocated in program memory, because in that case their values could not be adapted. If filter coefficients are detected as allocated in program memory the function returns NULL. (See also FIRStruct, FIRStructInit and FIRDelayInit.)

Source File:

firlms.s

Function Profile:

System resources usage:

- W0..W7 used, not restored
- W8..W12 saved, used, restored
- ACCA used, not restored
- ACCB used, not restored
- CORCON saved, used, restored
- MODCON saved, used, restored
- XMODSTRT saved, used, restored
- XMODEND saved, used, restored
- YMODSTRT saved, used, restored

DO and REPEAT instruction usage:

- 2 level DO instructions
- 1 level REPEAT instructions

Program words (24-bit instructions):

- 76

Cycles (including C-function call and return overheads):

- \( 61 + N(13 + 5M) \)

FIRLMSNorm

Description:

FIRLMSNorm applies an adaptive FIR filter to the sequence of source samples, stores the results in the sequence of destination samples, and updates the delay values. The filter coefficients are also updated, at a sample-per-sample basis, using a Normalized Least Mean Square algorithm applied according to the values of the reference samples.

Include:

dsp.h
Prototype: 
```c
extern fractional* FIRLMSNorm (
    int numSamps,
    fractional* dstSamps,
    fractional* srcSamps,
    FIRStruct* filter,
    fractional* refSamps,
    fractional muVal,
    fractional* energyEstimate
);
```

Arguments:
- `numSamps`: number of input samples (also `N`)
- `dstSamps`: pointer to destination samples (also `y`)
- `srcSamps`: pointer to source samples (also `x`)
- `filter`: pointer to FIRStruct filter structure
- `refSamps`: pointer to reference samples (also `r`)
- `muVal`: adapting factor (also `mu`)
- `energyEstimate`: estimated energy value for the last `M` input signal samples, with `M` the number of filter coefficients

Return Value: 
Pointer to base address of destination samples.

Remarks:
- Number of coefficients in filter is `M`.
- Coefficients, `h[m]`, defined in `0 ≤ m < M`, implemented as a circular increasing modulo buffer.
- delay, `d[m]`, defined in `0 ≤ m < M`, implemented as a circular increasing modulo buffer.
- Source samples, `x[n]`, defined in `0 ≤ n < N`.
- Reference samples, `r[n]`, defined in `0 ≤ n < N`.
- Destination samples, `y[n]`, defined in `0 ≤ n < N`.
- Adaptation:
  ```c
  h_m[n] = h_m[n – 1] + nu[n] * (r[n] – y[n]) * x[n – m],
  ```
  for `0 ≤ n < N, 0 ≤ m < M`, where `nu[n] = mu / (mu + E[n])`
  `with E[n]=E[n – 1] + (x[n])^2 – (x[n – M + 1])^2` an estimate of input signal energy.
- On start up, `energyEstimate` should be initialized to the value of `E[-1]` (zero the first time the filter is invoked). Upon return, `energyEstimate` is updated to the value `E[N – 1]` (which may be used as the start up value for a subsequent function call if filtering an extension of the input signal).
- The operation could result in saturation if the absolute value of `(r[n] – y[n])` is greater than or equal to one.
- **Note:** Another expression for the energy estimate is:
  ```c
  E[n] = (x[n])^2 + (x[n – 1])^2 + ... + (x[n – M + 2])^2.
  ```
  Thus, to avoid saturation while computing the estimate, the input sample values should be bound so that
  ```c
  \sum_{m=0}^{M+2} (x[n + m])^2 < 1 , \text{ for } 0 ≤ n < N.
  ```
- Filter coefficients must not be allocated in program memory, because in that case their values could not be adapted. If filter coefficients are detected as allocated in program memory the function returns `NULL`. (See also `FIRStruct`, `FIRStructInit` and `FIRDelayInit`.)

Source File: `firlmsn.s`
FIRLMSNorm (Continued)

Function Profile:          System resources usage:
  W0..W7                  used, not restored
  W8..W13                 saved, used, restored
  ACCA                    used, not restored
  ACCB                    used, not restored
  CORCON                  saved, used, restored
  MODCON                  saved, used, restored
  XMODSTRT                saved, used, restored
  XMODEND                 saved, used, restored
  YMODSTRT                saved, used, restored

DO and REPEAT instruction usage:
  2 level DO instructions
  1 level REPEAT instructions

Program words (24-bit instructions): 91

Cycles (including C-function call and return overheads): 66 + N(49 + 5M)

FIRStructInit

Description:    FIRStructInit initializes the values of the parameters in an
                FIRStruct FIR Filter structure.

Include:        dsp.h

Prototype:      extern void FIRStructInit(
                 FIRStruct* filter,
                 int numCoeffs,
                 fractional* coeffsBase,
                 int coeffsPage,
                 fractional* delayBase
                 );

Arguments:      filter         pointer to FIRStruct filter structure
                 numCoeffs     number of coefficients in filter (also M)
                 coeffsBase   base address for filter coefficients (also h)
                 coeffsPage   coefficient buffer page number
                 delayBase    base address for delay buffer

Remarks:        See description of FIRStruct structure above.
                 Upon completion, FIRStructInit initializes the coeffsEnd and
delayEnd pointers accordingly. Also, delay is set equal to
delayBase.

Source File:    firinit.s

Function Profile:  System resources usage:
  W0..W5                  used, not restored

DO and REPEAT instruction usage:
  no DO instructions
  no REPEAT instructions

Program words (24-bit instructions): 10

Cycles (including C-function call and return overheads): 19
**IIRCanonic**

**Description:**
IIRCanonic applies an IIR filter, using a cascade of canonic (direct form II) biquadratic sections, to the sequence of source samples. It places the results in the sequence of destination samples, and updates the delay values.

**Include:**
dsp.h

**Prototype:**
```c
typedef struct {
    int numSectionsLess1;
    fractional* coeffsBase;
    int coeffsPage;
    fractional* delayBase;
    int initialGain;
    int finalShift;
} IIRCanonicStruct;

extern fractional* IIRCanonic (
    int numSamps,
    fractional* dstSamps,
    fractional* srcSamps,
    IIRCanonicStruct* filter
);
```

**Arguments:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numSectionsLess1</td>
<td>1 less than number of cascaded second order (biquadratic) sections (also S-1)</td>
</tr>
<tr>
<td>coeffsBase</td>
<td>pointer to filter coefficients (also [a, b]), either within X-Data or program memory</td>
</tr>
<tr>
<td>coeffsPage</td>
<td>coefficients buffer page number, or 0xFF00 (defined value COEFFS_IN_DATA) if coefficients in data space</td>
</tr>
<tr>
<td>delayBase</td>
<td>pointer to filter delay (also d), only in Y-Data</td>
</tr>
<tr>
<td>initialGain</td>
<td>initial gain value</td>
</tr>
<tr>
<td>finalShift</td>
<td>output scaling (shift left)</td>
</tr>
</tbody>
</table>

**Filter Description:**
numSamps number of input samples to filter (also N)
dstSamps pointer to destination samples (also y)
srcSamps pointer to source samples (also x)
filter pointer to IIRCanonicStruct filter structure

**Return Value:**
Pointer to base address of destination samples.

**Remarks:**
There are 5 coefficients per second order (biquadratic) sections arranged in the ordered set \{a2[s], a1[s], b2[s], b1[s], b0[s]\}, 0 ≤ s < S. Coefficient values should be generated with dsPICFD filter design package from Momentum Data Systems, Inc., or similar tool. The delay is made up of two words of filter state per section \{d1[s], d2[s]\}, 0 ≤ s < S.

Source samples, x[n], defined in 0 ≤ n < N.
Destination samples, y[n], defined in 0 ≤ n < N.
Initial gain value is applied to each input sample prior to entering the filter structure.
The output scale is applied as a shift to the output of the filter structure prior to storing the result in the output sequence. It is used to restore the filter gain to 0 dB. Shift count may be zero; if not zero, it represents the number of bits to shift: negative indicates shift left, positive is shift right.

**Source File:**
iircan.s
**Function Profile:**  
System resources usage:
- **W0..W7** used, not restored  
- **W8..W11** saved, used, restored  
- **ACCA** used, not restored  
- **CORCON** saved, used, restored  
- **PSVPAG** saved, used, restored  

**DO and REPEAT instruction usage:**
- **2 level DO instructions**  
- **1 level REPEAT instructions**  

Program words (24-bit instructions):
- **42**  

Cycles (including C-function call and return overheads):
- **36 + N(8 + 7S), or 39 + N(9 + 12S) if coefficients in program memory.**

---

### IIRCanonicInit

**Description:**  
The IIRCanonicInit function initializes to zero the delay values in an IIRCanonicStruct filter structure.

**Include:**  
dsp.h  

**Prototype:**  
extern void IIRCanonicInit (
  IIRCanonicStruct* filter
);  

**Arguments:**  
Filter structure:  
(See description of IIRCanonic function).

**Initialization Description:**  
filter pointer to IIRCanonicStruct filter structure

**Remarks:**  
Two words of filter state per second order section (d1[s], d2[s]),  
0 ≤ s < S.

**Source File:**  
iircan.s

**Function Profile:**  
System resources usage:  
- **W0, W1** used, not restored

**DO and REPEAT instruction usage:**
- **1 level DO instructions**  
- **no REPEAT instructions**

Program words (24-bit instructions):
- **7**

Cycles (including C-function call and return overheads):
- **10 + S2.**
### IIRLattice

**Description:** IIRLattice uses a lattice structure implementation to apply an IIR filter to the sequence of source samples. It then places the results in the sequence of destination samples, and updates the delay values.

**Include:**

dsp.h

**Prototype:**

typedef struct {
    int order;
    fractional* kappaVals;
    fractional* gammaVals;
    int coeffsPage;
    fractional* delay;
} IIRLatticeStruct;

extern fractional* IIRLattice {
    int numSamps,
    fractional* dstSamps,
    fractional* srcSamps,
    IIRLatticeStruct* filter
};

**Arguments:**

Filter structure:

- **order:** filter order (also M, M ≤ N; see FIRLattice for N)
- **kappaVals:** base address for lattice coefficients (also k), either in X-Data or program memory
- **gammaVals:** base address for ladder coefficients (also g), either in X-Data or program memory. If NULL, the function will implement an all-pole filter.
- **coeffsPage:** coefficients buffer page number, or 0xFF00 (defined value COEFS_IN_DATA) if coefficients in data space
- **delay:** base address for delay (also d), only in Y-Data

Filter Description:

- **numSamps:** number of input samples to filter (also N, N ≥ M; see IIRLatticeStruct for M)
- **dstSamps:** pointer to destination samples (also y)
- **srcSamps:** pointer to source samples (also x)
- **filter:** pointer to IIRLatticeStruct filter structure

**Return Value:**

Pointer to base address of destination samples.

**Remarks:**

- Lattice coefficients, k[m], defined in 0 ≤ m ≤ M.
- Ladder coefficients, g[m], defined in 0 ≤ m ≤ M (unless if implementing an all-pole filter).
- Delay, d[m], defined in 0 ≤ m ≤ M.
- Source samples, x[n], defined in 0 ≤ n < N.
- Destination samples, y[n], defined in 0 ≤ n < N.

**Note:** The fractional implementation provided with this library is prone to saturation. Design and test the filter “off-line” using a floating-point implementation such as the OCTAVE model at the end of this section. Then, the intermediate forward and backward values should be monitored during the floating-point execution in search for levels outside the [-1, 1) range. If any one of the intermediate values spans outside of that range, the maximum absolute value should be used to scale the input signal prior to applying the fractional filter in real-time; i.e., multiply the signal by the inverse of that maximum. This scaling should prevent the fractional implementation from saturating.

**Source File:** iirlatt.s
IIRLattice (Continued)

Function Profile:  System resources usage:

| W0..W7       | used, not restored            |
| W8..W13      | saved, used, restored         |
| ACCA         | used, not restored            |
| ACCB         | used, not restored            |
| CORCON       | saved, used, restored         |

DO and REPEAT instruction usage:
- 2 level DO instructions
- no REPEAT instructions

Program words (24-bit instructions):
76

Cycles (including C-function call and return overheads):
46 + N(16 + 7M), or
49 + N(20 + 8M) if coefficients in program memory.

If implementing an all-pole filter:
46 + N(16 + 6M), or
49 + N(16 + 7M) if coefficients in program memory

IIRLatticeInit

Description: IIRLatticeInit initializes to zero the delay values in an IIRLatticeStruct filter structure.

Include: dsp.h

Prototype: extern void IIRLatticeInit (
  IIRLatticeStruct* filter
);

Arguments: Filter structure:
(See description of IIRLatticeInit function).

Initialization Description:
filter pointer to IIRLatticeStruct filter structure.

Source File: iirlattd.s

Function Profile: System resources usage:

| W0..W2       | used, not restored            |

DO and REPEAT instruction usage:
- no DO instructions
- 1 level REPEAT instructions

Program words (24-bit instructions):
6

Cycles (including C-function call and return overheads):
10 + M
## IIRTransposed

**Description:** IIRTransposed applies an IIR filter, using a cascade of transposed (direct form II) biquadratic sections, to the sequence of source samples. It places the results in the sequence of destination samples, and updates the delay values.

**Include:**

dsp.h

**Prototype:**

typedef struct {
    int numSectionsLess1;
    fractional* coeffsBase;
    int coeffsPage;
    fractional* delayBase1;
    fractional* delayBase2;
    int finalShift;
} IIRTransposedStruct;

extern fractional* IIRTransposed (
    int numSamps,
    fractional* dstSamps,
    fractional* srcSamps,
    IIRTransposedStruct* filter
);

**Arguments:**

- numSectionsLess1: 1 less than number of cascaded second order (biquadratic) sections (also S-1)
- coeffsBase: pointer to filter coefficients (also [a, b]), either in X-Data or program memory
- coeffsPage: coefficient buffer page number, or 0xFF00 (defined value COEFFS_IN_DATA) if coefficients in data space
- delayBase1: pointer to filter state 1, with one word of delay per second order section (also d1), only in Y-Data
- delayBase2: pointer to filter state 2, with one word of delay per second order section (also d2), only in Y-Data
- finalShift: output scaling (shift left)

**Filter Description:**
- numSamps: number of input samples to filter (also N)
- dstSamps: pointer to destination samples (also y)
- srcSamps: pointer to source samples (also x)
- filter: pointer to IIRTransposedStruct filter structure

**Return Value:** Pointer to base address of destination samples.

**Remarks:**

There are 5 coefficients per second order (biquadratic) section arranged in the ordered set \{b0[s], b1[s], a1[s], b2[s], a2[s]\}, 0 ≤ s < S. Coefficient values should be generated with dsPICFD filter design package from Momentum Data Systems, Inc., or similar tool. The delay is made up of two independent buffers, each buffer containing one word of filter state per section \{d2[s], d1[s]\}, 0 ≤ s < S. Source samples, x[n], defined in 0 ≤ n < N. Destination samples, y[n], defined in 0 ≤ n < N.

The output scale is applied as a shift to the output of the filter structure prior to storing the result in the output sequence. It is used to restore the filter gain to 0 dB. Shift count may be zero; if not zero, it represents the number of bits to shift: negative indicates shift left, positive is shift right.

**Source File:** iirtrans.s
IIRTransposed (Continued)

Function Profile: System resources usage:
- W0..W7 used, not restored
- W8..W11 saved, used, restored
- ACCA used, not restored
- ACCB used, not restored
- CORCON saved, used, restored
- PSVPAG saved, used, restored

DO and REPEAT instruction usage:
- 2 level DO instructions
- 1 level REPEAT instructions

Program words (24-bit instructions):
- 48

Cycles (including C-function call and return overheads):
- \(35 + N(11 + 11S)\), or
- \(38 + N(9 + 17S)\) if coefficients in P memory.
  - \(S\) is number of second order sections.

Example
Please refer to the MPLAB C30 installation folder for a sample project demonstrating the use of this function.

IIRTransposedInit

Description: IIRTransposedInit initializes to zero the delay values in an IIRTransposedStruct filter structure.

Include:
dsp.h

Prototype:
extern void IIRTransposedInit (IIRTransposedStruct* filter);

Arguments:
- Filter structure: Filter structure:
  - (See description of IIRTransposed function).

Initialization Description:
- filter pointer to IIRTransposedStruct filter structure.

Remarks:
The delay is made up of two independent buffers, each buffer containing one word of filter state per section \(\{d2[s], d1[s]\}\), \(0 \leq s < S\).

Source File:
iirtrans.s

Function Profile: System resources usage:
- W0..W2 used, not restored

DO and REPEAT instruction usage:
- 1 level DO instructions
  - no REPEAT instructions

Program words (24-bit instructions):
- 8

Cycles (including C-function call and return overheads):
- \(11 + 2S\),
  - \(S\) is number of second order sections.

Example
Please refer to the MPLAB C30 installation folder for a sample project demonstrating the use of this function.
2.6.6 OCTAVE model for analysis of IIRLattice filter

The following OCTAVE model may be used to examine the performance of an IIR Lattice Filter prior to using the fractional implementation provided by the function IIRLattice.

IIRLattice OCTAVE model

```octave
function [out, del, forward, backward] = iirlatt (in, kappas, gammas, delay)
## FUNCTION.
## IIRLATT: IIR Filter Lattice implementation.
##
## [out, del, forward, backward] = iirlatt (in, kappas, gammas, delay)
##
## forward: records intermediate forward values.
## backward: records intermediate backward values.
#
## Get implicit parameters.
numSamps = length(in); numKapps = length(kappas);
if (gammas != 0)
    numGamms = length(gammas);
else
    numGamms = 0;
endif
numDels = length(delay); filtOrder = numDels-1;

## Error check.
if (numGamms != 0)
    if (numGamms != numKapps)
        fprintf("ERROR! %d should be equal to %d.\n", numGamms, numKapps);
        return;
    endif
endif
if (numDels != numKapps)
    fprintf("ERROR! %d should equal to %d.\n", numDels, numKapps);
    return;
endif

## Initialize.
M = filtOrder; out = zeros(numSamps,1); del = delay;
forward = zeros(numSamps*M,1); backward = forward; i = 0;

## Filter samples.
for n = 1:numSamps
    ## Get new sample.
    current = in(n);
```

## Lattice structure.

```matlab
for m = 1:M
    after = current - kappas(M+1-m) * del(m+1);
    del(m) = del(m+1) + kappas(M+1-m) * after;
    i = i+1;
    forward(i) = current;
    backward(i) = after;
    current = after;
end

del(M+1) = after;
```

## Ladder structure (computes output).

```matlab
if (gammas == 0)
    out(n) = del(M+1);
else
    for m = 1:M+1
        out(n) = out(n) + gammas(M+2-m)*del(m);
    endfor
endif
endfor
```

## Return.

```matlab
return;
```

#endfunction
2.7 TRANSFORM FUNCTIONS

This section presents the concept of a fractional transform, as considered by the DSP Library, and describes the individual functions which perform transform operations. The user may refer to example projects that utilize the DSP library Transform functions, in order to ascertain proper usage of functions. Example MPLAB IDE-based projects/workspaces have been provided in the installation folder of the MPLAB C30 toolsuite.

2.7.1 Fractional Transform Operations

A fractional transform is a linear, time invariant, discrete operation that when applied to a fractional time domain sample sequence, results in a fractional frequency in the frequency domain. Conversely, inverse fractional transform operation, when applied to frequency domain data, results in its time domain representation.

A set of transforms (and a subset of inverse transforms) are provided by the DSP Library. The first set applies a Discrete Fourier transform (or its inverse) to a complex data set (see below for a description of fractional complex values). The second set applies a Type II Discrete Cosine Transform (DCT) to a real valued sequence. These transforms have been designed to either operate out-of-place, or in-place. The former type populates an output sequence with the results of the transformation. In the latter, the input sequence is (physically) replaced by the transformed sequence. For out-of-place operations, enough memory to accept the results of the computation must be provided.

The transforms make use of transform factors (or constants) which must be supplied to the transforming function during its invocation. These factors, which are complex data sets, are computed in floating-point arithmetic, and then transformed into fractionals for use by the operations. To avoid excessive computational overhead when applying a transformation, a particular set of transform factors could be generated once and used many times during the execution of the program. Thus, it is advisable to store the factors returned by any of the initialization operations in a permanent (static) complex vector. It is also advantageous to generate the factors “off-line”, and place them in program memory, and use them when the program is later executing. This way, not only cycles, but also RAM memory is saved when designing an application which involves transformations.

2.7.2 Fractional Complex Vectors

A complex data vector is represented by a data set in which every pair of values represents an element of the vector. The first value in the pair is the real part of the element, and the second its imaginary part. Both the real and imaginary parts are stored in memory using one word (two bytes) for each, and must be interpreted as 1.15 fractionals. As with the fractional vector, the fractional complex vector stores its elements consecutively in memory.

The organization of data in a fractional complex vector may be addressed by the following data structure:

```c
#ifdef fractional
ifndef fractcomplex
typedef struct {
    fractional real;
    fractional imag;
} fractcomplex;
#else
endif
#endif
```
2.7.3 User Considerations

a) No boundary checking is performed by these functions. Out of range sizes (including zero length vectors) as well as nonconforming use of source complex vectors and factor sets may produce unexpected results.

b) It is recommended that the STATUS Register (SR) is examined after completion of each function call. In particular, users can inspect the SA, SB and SAB flags after the function returns to determine if saturation occurred.

c) The input and output complex vectors involved in the family of transformations must be allocated in Y-Data memory. Transforms factors may be allocated either in X-Data or program memory.

d) Because bit reverse addressing requires the vector set to be modulo aligned, the input and output complex vectors in operations using either explicitly or implicitly the BitReverseComplex function must be properly allocated.

e) Operations which return a destination complex vector can be nested, so that for instance if:

\[ \text{a} = \text{Op1 (b, c)}, \text{with b} = \text{Op2 (d)}, \text{and c} = \text{Op3 (e, f)}, \text{then} \]

\[ \text{a} = \text{Op1 (Op2 (d), Op3 (e, f))}. \]

In what follows, the individual functions implementing transform and inverse transform operations are described.

BitReverseComplex

**Description:**

BitReverseComplex reorganizes the elements of a complex vector in bit reverse order.

**Include:**
dsp.h

**Prototype:**

```c
extern fractcomplex* BitReverseComplex ( 
    int log2N, 
    fractcomplex* srcCV 
);
```

**Arguments:**

- `log2N` based 2 logarithm of N (number of complex elements in source vector)
- `srcCV` pointer to source complex vector

**Return Value:**

Pointer to base address of source complex vector.

**Remarks:**

- N must be an integer power of 2.
- The `srcCV` vector must be allocated at a modulo alignment of N.
- This function operates in place.

**Source File:**

bitrev.s

**Function Profile:**

System resources usage:

- W0..W7 used, not restored
- MODCON saved, used, restored
- XBREV saved, used, restored

DO and REPEAT instruction usage:

- 1 level DO instructions
- no REPEAT instructions

Program words (24-bit instructions):

27

Cycles (including C-function call and return overheads):

See below:

**Example**

Please refer to the MPLAB C30 installation folder for a sample project demonstrating the use of this function.
## 16-Bit Language Tools Libraries

<table>
<thead>
<tr>
<th>Transform Size</th>
<th># Complex Elements</th>
<th># Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 point</td>
<td>32</td>
<td>245</td>
</tr>
<tr>
<td>64 point</td>
<td>64</td>
<td>485</td>
</tr>
<tr>
<td>128 point</td>
<td>128</td>
<td>945</td>
</tr>
<tr>
<td>256 point</td>
<td>256</td>
<td>1905</td>
</tr>
</tbody>
</table>
CosFactorInit

Description: CosFactorInit generates the first half of the set of cosine factors required by a Type II Discrete Cosine Transform, and places the result in the complex destination vector. Effectively, the set contains the values:

\[ CN(k) = e^{\frac{j nk}{2N}}, \text{ where } 0 \leq k < N/2. \]

Include: dsp.h

Prototype: extern fractcomplex* CosFactorInit (int log2N, fractcomplex* cosFactors);

Arguments: log2N based 2 logarithm of N (number of complex factors needed by a DCT)

Return Value: Pointer to base address of cosine factors.

Remarks: N must be an integer power of 2.

A complex vector of size N/2 must have already been allocated and assigned to cosFactors prior to invoking the function. The complex vector should reside in X-Data memory.

Factors are computed in floating-point arithmetic and converted to 1.15 complex fractionals.

Source File: initcosf.c

Function Profile: System resources usage:

\[ \begin{align*}
\text{W0..W7} & \text{ used, not restored} \\
\text{W8..W14} & \text{ saved, used, restored}
\end{align*} \]

DO and REPEAT instruction usage:

None

Program words (24-bit instructions):

See the file “readme.txt” in pic30_tools\src\dsp for this information.

Cycles (including C-function call and return overheads):

See the file “readme.txt” in pic30_tools\src\dsp for this information.

DCT

Description: DCT computes the Discrete Cosine Transform of a source vector, and stores the results in the destination vector.

Include: dsp.h

Prototype: extern fractional* DCT (int log2N, fractional* dstV, fractional* srcV, fractcomplex* cosFactors, fractcomplex* twidFactors, int factPage);

C
## DCT (Continued)

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log2N</td>
<td>log2N based 2 logarithm of N (number of complex elements in source vector)</td>
</tr>
<tr>
<td>dstCV</td>
<td>pointer to destination vector</td>
</tr>
<tr>
<td>srcCV</td>
<td>pointer to source vector</td>
</tr>
<tr>
<td>cosFactors</td>
<td>pointer to cosine factors</td>
</tr>
<tr>
<td>twidFactors</td>
<td>pointer to twiddle factors</td>
</tr>
<tr>
<td>factPage</td>
<td>memory page for transform factors</td>
</tr>
</tbody>
</table>

**Return Value:** Pointer to base address of destination vector.

**Remarks:**
- N must be an integer power of 2.
- This function operates out of place. A vector of size 2N elements, must already been allocated and assigned to dstV.
- The dstV vector must be allocated at a modulo alignment of N.
- The results of computation are stored in the first N elements of the destination vector.
- To avoid saturation (overflow) during computation, the values of the source vector should be in the range [-0.5, 0.5].
- Only the first N/2 cosine factors are needed.
- Only the first N/2 twiddle factors are needed.
- If the transform factors are stored in X-Data space, cosFactors and twidFactors point to the actual address where the factors are allocated. If the transform factors are stored in program memory, cosFactors and twidFactors are the offset from the program page boundary where the factors are allocated. This latter value can be calculated using the inline assembly operator psvoffset().
- If the transform factors are stored in X-Data space, factPage must be set to 0xFF00 (defined value COEFFS_IN_DATA). If they are stored in program memory, factPage is the program page number containing the factors. This latter value can be calculated using the inline assembly operator psvpage().
- The twiddle factors must be initialized with conjFlag set to a value different than zero.
- Only the first N/2 cosine factors are needed.
- Output is scaled by the factor \( \frac{1}{\sqrt{2N}} \)

**Source File:** dctoop.s

**Function Profile:**
- **System resources usage:**
  - W0..W5 used, not restored
  - plus system resources from VectorZeroPad, and DCTIP
- **DO and REPEAT instruction usage:**
  - no DO instructions
  - no REPEAT instructions
  - plus DO/REPEAT instructions from VectorZeroPad, and DCTIP
- **Program words (24-bit instructions):**
  - 16
  - plus program words from VectorZeroPad, and DCTIP
- **Cycles (including C-function call and return overheads):**
  - 22
  - plus cycles from VectorZeroPad, and DCTIP

**Note:** In the description of VectorZeroPad the number of cycles reported includes 4 cycles of C-function call overhead. Thus, the number of actual cycles from VectorZeroPad to add to DCT is 4 less than whatever number is reported for a stand-alone VectorZeroPad.
In the same way, the number of actual cycles from DCTIP to add to DCT is 3 less than whatever number is reported for a stand-alone DCTIP.
## DCTIP

**Description:** DCTIP computes the Discrete Cosine Transform of a source vector in place.

**Include:** dsp.h

**Prototype:**

```c
extern fractional* DCTIP(
    int log2N,
    fractional* srcV,
    fractcomplex* cosFactors,
    fractcomplex* twidFactors,
    int factPage
);
```

**Arguments:**

- `log2N`: based 2 logarithm of N (number of complex elements in source vector)
- `srcV`: pointer to source vector
- `cosFactors`: pointer to cosine factors
- `twidFactors`: pointer to twiddle factors
- `factPage`: memory page for transform factors

**Return Value:** Pointer to base address of destination vector.

**Remarks:**

- N must be an integer power of 2.
- This function expects that the source vector has been zero padded to length 2N.
- The `srcV` vector must be allocated at a modulo alignment of N.
- The results of computation are stored in the first N elements of source vector.
- To avoid saturation (overflow) during computation, the values of the source vector should be in the range \([-0.5, 0.5]\).
- Only the first N / 2 cosine factors are needed.
- Only the first N / 2 twiddle factors are needed.
- If the transform factors are stored in X-Data space, `cosFactors` and `twidFactors` point to the actual address where the factors are allocated. If the transform factors are stored in program memory, `cosFactors` and `twidFactors` are the offset from the program page boundary where the factors are allocated. This latter value can be calculated using the inline assembly operator `psvoffset()`.
- If the transform factors are stored in X-Data space, `factPage` must be set to 0xFF00 (defined value `COEFFS_IN_DATA`). If they are stored in program memory, `factPage` is the program page number containing the factors. This latter value can be calculated using the inline assembly operator `psvpage()`.
- The twiddle factors must be initialized with `conjFlag` set to a value different than zero.
- Output is scaled by the factor $1/(\sqrt{2N})$.

**Source File:** dctoop.s
DCTIP (Continued)

Function Profile:

System resources usage:

- W0..W7 used, not restored
- W8..W13 saved, used, restored
- ACCA used, not restored
- CORCON saved, used, restored
- PSVPAG saved, used, restored (only if coefficients in P memory)

DO and REPEAT instruction usage:

- 1 level DO instructions
- 1 level REPEAT instructions
- plus DO/REPEAT instructions from IFFTComplexIP

Program words (24-bit instructions):

- 92
- plus program words from IFFTComplexIP

Cycles (including C-function call and return overheads):

- 71 + 10N, or
- 73 + 11N if factors in program memory,
- plus cycles from IFFTComplexIP

Note: In the description of IFFTComplexIP the number of cycles reported includes 4 cycles of C-function call overhead. Thus, the number of actual cycles from IFFTComplexIP to add to DCTIP is 4 less than whatever number is reported for a stand-alone IFFTComplexIP.

FFTComplex

Description:

FFTComplex computes the Discrete Fourier Transform of a source complex vector, and stores the results in the destination complex vector.

Include:

dsp.h

Prototype:

extern fractcomplex* FFTComplex (int log2N,
fractcomplex* dstCV,
fractcomplex* srcCV,
fractcomplex* twidFactors,
int factPage
);

Arguments:

- log2N based 2 logarithm of N (number of complex elements in source vector)
- dstCV pointer to destination complex vector
- srcCV pointer to source complex vector
- twidFactors base address of twiddle factors
- factPage memory page for transform factors

Return Value:

Pointer to base address of destination complex vector.
FFTComplex (Continued)

**Remarks:**
- **N must** be an integer power of 2.
- This function operates out of place. A complex vector, large enough to receive the results of the operation, *must* already have been allocated and assigned to `dstCV`.
- The `dstCV` vector must be allocated at a modulo alignment of `N`.
- The elements in source complex vector are expected in natural order.
- The elements in destination complex vector are generated in natural order.
- To avoid saturation (overflow) during computation, the magnitude of the values of the source complex vector should be in the range [-0.5, 0.5]. Only the first N/2 twiddle factors are needed.
- If the twiddle factors are stored in X-Data space, `twidFactors` points to the actual address where the factors are allocated. If the twiddle factors are stored in program memory, `twidFactors` is the offset from the program page boundary where the factors are allocated. This latter value can be calculated using the inline assembly operator `psvoffset()`.
- If the twiddle factors are stored in X-Data space, `factPage` must be set to 0xFF00 (defined value `COEFFS_IN_DATA`). If they are stored in program memory, `factPage` is the program page number containing the factors. This latter value can be calculated using the inline assembly operator `psvpage()`.
- The twiddle factors must be initialized with `conjFlag` set to zero.
- Output is scaled by the factor 1/N.

**Source File:** fftcoo.s

**Function Profile:**
- **System resources usage:**
  - `W0..W4` used, not restored
  - plus system resources from `VectorCopy`, `FFTComplexIP`, and `BitReverseComplex`.
- **DO and REPEAT instruction usage:**
  - no DO instructions
  - no REPEAT instructions
  - plus DO/REPEAT instructions from `VectorCopy`, `FFTComplexIP`, and `BitReverseComplex`.
- **Program words (24-bit instructions):**
  - 17
  - plus program words from `VectorCopy`, `FFTComplexIP`, and `BitReverseComplex`.
- **Cycles (including C-function call and return overheads):**
  - 23
  - plus cycles from `VectorCopy`, `FFTComplexIP`, and `BitReverseComplex`.

**Note:** In the description of `VectorCopy` the number of cycles reported includes 3 cycles of C-function call overhead. Thus, the number of actual cycles from `VectorCopy` to add to `FFTComplex` is 3 less than whatever number is reported for a stand-alone `VectorCopy`. In the same way, the number of actual cycles from `FFTComplexIP` to add to `FFTComplex` is 4 less than whatever number is reported for a stand-alone `FFTComplexIP`. And those from `BitReverseComplex` are 2 less than whatever number is reported for a stand-alone `FFTComplex`. 
### FFTComplexIP

**Description:** FFTComplexIP computes the Discrete Fourier Transform of a source complex vector in place.

**Include:**

dsp.h

**Prototype:**

```c
extern fractcomplex* FFTComplexIP (int log2N, fractcomplex* srcCV, fractcomplex* twidFactors, int factPage);
```

**Arguments:**

- `log2N` based 2 logarithm of N (number of complex elements in source vector)
- `srcCV` pointer to source complex vector
- `twidFactors` base address of twiddle factors
- `factPage` memory page for transform factors

**Return Value:** Pointer to base address of source complex vector.

**Remarks:**

- N must be an integer power of 2.
- The elements in source complex vector are expected in natural order.
- The resulting transform is stored in bit reverse order.
- To avoid saturation (overflow) during computation, the magnitude of the values of the source complex vector should be in the range [-0.5, 0.5].
- Only the first N/2 twiddle factors are needed.
- If the twiddle factors are stored in X-Data space, `twidFactors` points to the actual address where the factors are allocated. If the twiddle factors are stored in program memory, `twidFactors` is the offset from the program page boundary where the factors are allocated. This latter value can be calculated using the inline assembly operator `psvoffset()`.
- If the twiddle factors are stored in X-Data space, `factPage` must be set to 0xFF00 (defined value `COEFFS_IN_DATA`). If they are stored in program memory, `factPage` is the program page number containing the factors. This latter value can be calculated using the inline assembly operator `psvpage()`.
- The twiddle factors must be initialized with `conjFlag` set to zero.
- Output is scaled by the factor 1/N.

**Source File:** fft.s
FFTComplexIP (Continued)

Function Profile:

System resources usage:
- W0..W7 used, not restored
- W8..W13 saved, used, restored
- ACCA used, not restored
- ACCB used, not restored
- CORCON saved, used, restored
- PSVPAG saved, used, restored (only if coefficients in P memory)

DO and REPEAT instruction usage:
- 2 level DO instructions
- no REPEAT instructions

Program words (24-bit instructions):
- 59

Cycles (including C-function call and return overheads):
- See table below

Example:
Please refer to the MPLAB C30 installation folder for a sample project demonstrating the use of this function.

<table>
<thead>
<tr>
<th>Transform Size</th>
<th># Cycles if Twiddle Factors in X-mem</th>
<th># Cycles if Twiddle Factors in P-mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 point</td>
<td>1,633</td>
<td>1,795</td>
</tr>
<tr>
<td>64 point</td>
<td>3,739</td>
<td>4,125</td>
</tr>
<tr>
<td>128 point</td>
<td>8,485</td>
<td>9,383</td>
</tr>
<tr>
<td>256 point</td>
<td>19,055</td>
<td>21,105</td>
</tr>
</tbody>
</table>

IFFTComplex

Description: IFFTComplex computes the Inverse Discrete Fourier Transform of a source complex vector, and stores the results in the destination complex vector.

Include:
dsp.h

Prototype:
extern fractcomplex* IFFTComplex (int log2N, fractcomplex* dstCV, fractcomplex* srcCV, fractcomplex* twidFactors, int factPage);

Arguments:
- log2N based 2 logarithm of N (number of complex elements in source vector)
- dstCV pointer to destination complex vector
- srcCV pointer to source complex vector
- twidFactors base address of twiddle factors
- factPage memory page for transform factors

Return Value: Pointer to base address of destination complex vector.
Remarks:  
N must be an integer power of 2.  
This function operates out of place. A complex vector, large enough to  
receive the results of the operation, must already have been allocated and assigned to dstCV.  
The dstCV vector must be allocated at a modulo alignment of N.  
The elements in source complex vector are expected in natural order.  
The elements in destination complex vector are generated in natural  
order.  
To avoid saturation (overflow) during computation, the magnitude of the  
values of the source complex vector should be in the range [-0.5, 0.5].  
If the twiddle factors are stored in X-Data space, twidFactors points  
to the actual address where the factors are allocated. If the twiddle  
factors are stored in program memory, twidFactors is the offset from  
the program page boundary where the factors are allocated. This latter  
value can be calculated using the inline assembly operator  
psvoffset().  
If the twiddle factors are stored in X-Data space, factPage must be  
set to 0xFF00 (defined value COEFFS_IN_DATA). If they are stored in  
program memory, factPage is the program page number containing  
the factors. This latter value can be calculated using the inline  
assembly operator psvpage().  
The twiddle factors must be initialized with conjFlag set to a value  
other than zero.  
Only the first N/2 twiddle factors are needed.

Source File:  
ifftoop.s

Function Profile:  
System resources usage:  
W0..W4 used, not restored  
plus system resources from VectorCopy, and IFFTComplexIP.  
DO and REPEAT instruction usage:  
no DO instructions  
no REPEAT instructions  
plus DO/REPEAT instructions from VectorCopy, and  
IFFTComplexIP.  
Program words (24-bit instructions):  
12  
plus program words from VectorCopy, and IFFTComplexIP.  
Cycles (including C-function call and return overheads):  
15  
plus cycles from VectorCopy, and IFFTComplexIP.

Note: In the description of VectorCopy the number of cycles reported  
includes 3 cycles of C-function call overhead. Thus, the number of  
actual cycles from VectorCopy to add to IFFTComplex is 3 less than  
whatever number is reported for a stand-alone VectorCopy. In the  
same way, the number of actual cycles from IFFTComplexIP to add to  
IFFTComplex is 4 less than whatever number is reported for a  
stand-alone IFFTComplexIP.
IFFTComplexIP

**Description:** IFFTComplexIP computes the Inverse Discrete Fourier Transform of a source complex vector in place.

**Include:**
dsp.h

**Prototype:**
extern fractcomplex* IFFTComplexIP (
    int log2N,
    fractcomplex* srcCV,
    fractcomplex* twidFactors,
    int factPage
);

**Arguments:**
- *log2N* based 2 logarithm of N (number of complex elements in source vector)
- *srcCV* pointer to source complex vector
- *twidFactors* base address of twiddle factors
- *factPage* memory page for transform factors

**Return Value:** Pointer to base address of source complex vector.

**Remarks:**
- N must be an integer power of 2.
- The elements in source complex vector are expected in bit reverse order. The resulting transform is stored in natural order.
- The srcCV vector must be allocated at a modulo alignment of N.
- To avoid saturation (overflow) during computation, the magnitude of the values of the source complex vector should be in the range [-0.5, 0.5].
- If the twiddle factors are stored in X-Data space, twidFactors points to the actual address where the factors are allocated. If the twiddle factors are stored in program memory, *twidFactors* is the offset from the program page boundary where the factors are allocated. This latter value can be calculated using the inline assembly operator psvoffset().
- If the twiddle factors are stored in X-Data space, *factPage* must be set to 0xFF00 (defined value COEFFS_IN_DATA). If they are stored in program memory, *factPage* is the program page number containing the factors. This latter value can be calculated using the inline assembly operator psvpage().
- The twiddle factors must be initialized with *conjFlag* set to a value other than zero.
- Only the first N/2 twiddle factors are needed.

**Source File:** ifft.s
IFFTComplexIP (Continued)

Function Profile:

System resources usage:
- W0..W3 used, not restored
- plus system resources from FFTComplexIP, and BitReverseComplex.

DO and REPEAT instruction usage:
- no DO instructions
- no REPEAT instructions
- plus DO/REPEAT instructions from FFTComplexIP, and BitReverseComplex.

Program words (24-bit instructions):
- 11
- plus program words from FFTComplexIP, and BitReverseComplex.

Cycles (including C-function call and return overheads):
- 15
- plus cycles from FFTComplexIP, and BitReverseComplex.

Note: In the description of FFTComplexIP the number of cycles reported includes 3 cycles of C-function call overhead. Thus, the number of actual cycles from FFTComplexIP to add to IFFTComplexIP is 3 less than whatever number is reported for a stand-alone FFTComplexIP. In the same way, the number of actual cycles from BitReverseComplex to add to IFFTComplexIP is 2 less than whatever number is reported for a stand-alone BitReverseComplex.

SquareMagnitudeCplx

Description:
SquareMagnitudeCplx computes the squared magnitude of each element in a complex source vector.

Include:
- dsp.h

Prototype:
- extern fractional* SquareMagnitudeCplx (int numelems, fractcomplex* srcV, fractional* dstV); 

Arguments:
- numElems number of elements in the complex source vector
- srcV pointer to complex source vector
- dstV pointer to real destination vector

Return Value:
- Pointer to base address of destination vector.

Remarks:
- If the sum of squares of the real and imaginary parts of a complex element in the source vector is larger than $1 - 2^{-15}$, this operation results in saturation.
- This function can be used to operate in-place on a source data set.

Source File:
- cplxsqrmag.s
SquareMagnitudeCplx (Continued)

| Function Profile: System resources usage: |
|-----------------------------------------|-----------------------------------------|
| W0..W2 used, not restored                | W4, W5, W10 saved, used, restored       |
| ACCA used, not restored                 | CORCON saved, used, restored            |
| DO and REPEAT instruction usage:        | 1 level DO instructions                 |
|                                        | no REPEAT instructions                  |
| Program words (24-bit instructions):    | 19                                       |
| Cycles (including C-function call and return overheads): | 20 + 3(numElements)                     |

Example: Please refer to the MPLAB C30 installation folder for a sample project demonstrating the use of this function.

TwidFactorInit

Description: TwidFactorInit generates the first half of the set of twiddle factors required by a Discrete Fourier Transform or Discrete Cosine Transform, and places the result in the complex destination vector. Effectively, the set contains the values:

\[ W_N(k) = e^{-j \frac{2\pi k}{N}}, \text{ where } 0 \leq k \leq N/2, \text{ for } \text{conjFlag} = 0 \]

\[ W_N(k) = e^{-j \frac{2\pi k}{N}}, \text{ where } 0 \leq k \leq N/2, \text{ for } \text{conjFlag} \neq 0 \]

Include: dsp.h

Prototype:

```c
extern fractcomplex* TwidFactorInit (int log2N, fractcomplex* twidFactors, int conjFlag);
```

Arguments:

- `log2N` based 2 logarithm of N (number of complex factors needed by a DFT)
- `twidFactors` pointer to complex twiddle factors
- `conjFlag` flag to indicate whether or not conjugate values are to be generated

Return Value: Pointer to base address of twiddle factors.

Remarks:

N must be an integer power of 2.

Only the first N/2 twiddle factors are generated.

The value of `conjFlag` determines the sign in the argument of the exponential function. For forward Fourier Transforms, `conjFlag` should be set to '0'. For inverse Fourier Transforms and Discrete Cosine Transforms, `conjFlag` should be set to '1'.

A complex vector of size N/2 must have already been allocated and assigned to `twidFactors` prior to invoking the function. The complex vector should be allocated in X-Data memory.

Factors computed in floating-point arithmetic and converted to 1.15 complex fractionals.

Source File: inittwid.c
**TwidFactorInit (Continued)**

<table>
<thead>
<tr>
<th>Function Profile:</th>
<th>System resources usage:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>W0..W7 used, not restored</td>
</tr>
<tr>
<td></td>
<td>W8..W14 saved, used, restored</td>
</tr>
</tbody>
</table>

**DO and REPEAT instruction usage:**

None

**Program words (24-bit instructions):**

See the file “readme.txt” in pic30_tools\src\dsp for this information.

**Cycles (including C-function call and return overheads):**

See the file “readme.txt” in pic30_tools\src\dsp for this information.

**Example:**

Please refer to the MPLAB C30 installation folder for a sample project demonstrating the use of this function.
2.8 CONTROL FUNCTIONS

This section describes functions provided in the DSP library that aid the implementation of closed-loop control systems.

2.8.1 Proportional Integral Derivative (PID) Control

A complete discussion of Proportional Integral Derivative (PID) controllers is beyond the scope of this discussion, but this section will try to provide you with some guidelines for tuning PID controllers.

2.8.1.1 PID CONTROLLER BACKGROUND

A PID controller responds to an error signal in a closed control loop and attempts to adjust the controlled quantity in order to achieve the desired system response. The controlled parameter can be any measurable system quantity, such as speed, voltage or current. The output of the PID controller can control one or more system parameters that will affect the controlled system quantity. For example, a speed control loop in a Sensorless Brushless DC motor application can control the PWM duty cycle directly or it can set the current demand for an inner control loop that regulates the motor currents. The benefit of the PID controller is that it can be adjusted empirically by adjusting one or more gain values and observing the change in system response.

A digital PID controller is executed at a periodic sampling interval and it is assumed that the controller is executed frequently enough so that the system can be properly controlled. For example, the current controller in the Sensorless Brushless DC motor application is executed every PWM cycle, since the motor can change very rapidly. The speed controller in such an application is executed at the medium event rate (100 Hz), because motor speed changes will occur relatively slowly due to mechanical time constants.

The error signal is formed by subtracting the desired setting of the parameter to be controlled from the actual measured value of that parameter. This sign of the error indicates the direction of change required by the control input.

The Proportional (P) term of the controller is formed by multiplying the error signal by a P gain. This will cause the PID controller to produce a control response that is a function of the error magnitude. As the error signal becomes larger, the P term of the controller becomes larger to provide more correction.

The effect of the P term will tend to reduce the overall error as time elapses. However, the effect of the P term will reduce as the error approaches zero. In most systems, the error of the controlled parameter will get very close to zero, but will not converge. The result is a small remaining steady state error. The Integral (I) term of the controller is used to fix small steady state errors. The I term takes a continuous running total of the error signal. Therefore, a small steady state error will accumulate into a large error value over time. This accumulated error signal is multiplied by an I gain factor and becomes the I output term of the PID controller.

The Differential (D) term of the PID controller is used to enhance the speed of the controller and responds to the rate of change of the error signal. The D term input is calculated by subtracting the present error value from a prior value. This delta error value is multiplied by a D gain factor that becomes the D output term of the PID controller. The D term of the controller produces more control output the faster the system error is changing.

It should be noted that not all PID controllers will implement the D or, less commonly, the I terms. For example, the speed controller in a Brushless DC motor application described by Microchip Application Note AN901 does not have a D term due to the rel-
atively slow response time of motor speed changes. In this case, the D term could cause excessive changes in PWM duty cycle that could affect the operation of the sensorless algorithm and produce over current trips.

2.8.1.2 ADJUSTING PID GAINS

The P gain of a PID controller will set the overall system response. When first tuning a controller, the I and D gains should be set to zero. The P gain can then be increased until the system responds well to set-point changes without excessive overshoot or oscillations. Using lower values of P gain will 'loosely' control the system, while higher values will give 'tighter' control. At this point, the system will probably not converge to the set-point.

After a reasonable P gain is selected, the I gain can be slowly increased to force the system error to zero. Only a small amount of I gain is required in most systems. Note that the effect of the I gain, if large enough, can overcome the action of the P term, slow the overall control response, and cause the system to oscillate around the set-point. If this occurs, reducing the I gain and increasing the P gain will usually solve the problem.

After the P and I gains are set, the D gain can be set. The D term will speed up the response of control changes, but it should be used sparingly because it can cause very rapid changes in the controller output. This behavior is called 'set-point kick'. The set-point kick occurs because the difference in system error becomes instantaneously very large when the control set-point is changed. In some cases, damage to system hardware can occur. If the system response is acceptable with the D gain set to zero, you can probably omit the D term.

**FIGURE 2-1: PID CONTROL SYSTEM**
2.8.1.3 PID LIBRARY FUNCTIONS AND DATA STRUCTURES

The DSP library provides a PID Controller function, \texttt{PID ( tPID* )}, to perform a PID operation. The function uses a data structure defined in the header file \texttt{dsp.h}, which has the following form:

\begin{verbatim}
typedef struct {
  fractional* abcCoefficients;
  fractional* controlHistory;
  fractional controlOutput;
  fractional measuredOutput;
  fractional controlReference;
} tPID;
\end{verbatim}

Prior to invoking the \texttt{PID()} function, the application should initialize the data structure of type \texttt{tPID}. This is done in the following steps:

1. Calculate Coefficients from PID Gain values

   The element \texttt{abcCoefficients} in the data structure of type \texttt{tPID} is a pointer to A, B & C coefficients located in X-data space. These coefficients are derived from the PID gain values, Kp, Ki and Kd, shown in Figure 2-1, as follows:

   \begin{align*}
   A &= Kp + Ki + Kd \\
   B &= -(Kp + 2*Kd) \\
   C &= Kd
   \end{align*}

   To derive the A, B and C coefficients, the DSP library provides a function, \texttt{PIDCoeffCalc}.

2. Clear the PID State Variables

   The structural element \texttt{controlHistory} is a pointer to a history of 3 samples located in Y-space, with the first sample being the most recent (current). These samples constitute a history of current and past differences between the Reference Input and the Measured Output of the plant function. The \texttt{PIDInit} function clears the elements pointed to by \texttt{controlHistory}. It also clears the \texttt{controlOutput} element in the \texttt{tPID} data structure.

2.8.2 Individual Functions

\begin{center}
\underline{PIDInit}
\end{center}

**Description:** This routine clears the delay line elements in the 3-element array located in Y-space and pointed to by \texttt{controlHistory}. It also clears the current PID output element, \texttt{controlOutput}.

**Include:** \texttt{dsp.h}

**Prototype:**

\begin{verbatim}
void PIDInit ( tPID *fooPIDStruct );
\end{verbatim}

**Arguments:** \texttt{fooPIDStruct} is a pointer to a PID data structure of type \texttt{tPID}

**Return Value:** \texttt{void}

**Remarks:**

**Source File:** \texttt{pid.s}
### PIDCoeffCalc

**Description:** PIDInit computes the PID coefficients based on values of Kp, Ki and Kd provided by the user.

\[
\text{abcCoefficients}[0] = K_p + K_i + K_d \\
\text{abcCoefficients}[1] = -(K_p + 2*K_d) \\
\text{abcCoefficients}[2] = K_d
\]

This routine also clears the delay line elements in the array ControlDifference, as well as clears the current PID output element, ControlOutput.

**Include:** dsp.h

**Prototype:**

```c
void PIDCoeffCalc ( fractional *fooPIDGainCoeff, tPID *fooPIDStruct )
```

**Arguments:**

- `fooPIDGainCoeff` is a pointer to input array containing Kp, Ki, Kd coefficients in order [Kp, Ki, Kd]
- `fooPIDStruct` is a pointer to a PID data structure of type `tPID`

**Return Value:** Void.

**Remarks:** PIDCoefficient array elements may be subject to saturation depending on values of Kp, Ki and Kd.

**Source File:** pid.s

**Function Profile:**

<table>
<thead>
<tr>
<th>System resources usage:</th>
<th>System resources usage:</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0..W4 used, not restored</td>
<td>W0..W4 used, not restored</td>
</tr>
<tr>
<td>ACCA, ACCB used, not restored</td>
<td>ACCA, ACCB used, not restored</td>
</tr>
<tr>
<td>CORCON saved, used, restored</td>
<td>CORCON saved, used, restored</td>
</tr>
</tbody>
</table>

**DO and REPEAT instruction usage:**

- 0 level DO instructions
- 0 REPEAT instructions

**Program words (24-bit instructions):**

- 11

**Cycles (including C-function call and return overheads):**

- 13
PID

Description: PID computes the controlOutput element of the data structure tPID:

\[
\text{controlOutput}[n] = \text{controlOutput}[n-1] + \text{controlHistory}[n] \times \text{abcCoefficient}[0] \\
+ \text{controlHistory}[n-1] \times \text{abcCoefficient}[1] \\
+ \text{controlHistory}[n-2] \times \text{abcCoefficient}[2]
\]

where,

\[
\text{abcCoefficient}[0] = Kp + Ki + Kd \\
\text{abcCoefficient}[1] = -(Kp + 2Kd) \\
\text{abcCoefficient}[2] = Kd
\]

\[
\text{ControlHistory}[n] = \text{MeasuredOutput}[n] - \text{ReferenceInput}[n]
\]

Include: dsp.h

Prototype: extern void PID ( tPID* ooPIDStruct );

Arguments: fooPIDStruct is a pointer to a PID data structure of type tPID

Return Value: Pointer to fooPIDStruct

Remarks: controlOutput element is updated by the PID() routine. The controlOutput will be subject to saturation.

Source File: pid.s

Function Profile:

System resources usage:

- W0..W5 used, not restored
- W8,W10 saved, used, restored
- ACCA used, not restored
- CORCON saved, used, restored

DO and REPEAT instruction usage:

- 0 level DO instructions
- 0 REPEAT instructions

Program words (24-bit instructions): 28

Cycles (including C-function call and return overheads): 30
2.9 MISCELLANEOUS FUNCTIONS

This section describes other helpful functions provided in the DSP library.

2.9.1 Individual Functions

**Fract2Float**

Description: Fract2Float converts a 1.15 fractional value to an IEEE floating-point value.

Include: dsp.h

Prototype: extern float Fract2Float (fractional aVal);

Arguments: aVal 1.15 fractional number in the implicit range [-1, (+ 1 – 2\(^{-15}\)]

Return Value: IEEE floating-point value in the range [-1, (+ 1 – 2\(^{-15}\)]

Remarks: None

Source File: flt2frct.c

Function Profile:

System resources usage:
- W0..W7 used, not restored
- W8..W14 saved, used, restored

DO and REPEAT instruction usage: None

Program words (24-bit instructions):
- See the file “readme.txt” in pic30_tools\src\dsp for this information.

Cycles (including C-function call and return overheads):
- See the file “readme.txt” in pic30_tools\src\dsp for this information.

**Float2Fract**

Description: Float2Fract converts an IEEE floating-point value to a 1.15 fractional number.

Include: dsp.h

Prototype: extern fractional Float2Fract (float aVal);

Arguments: aVal Floating-point number in the range [-1, (+ 1 – 2\(^{-15}\)]

Return Value: 1.15 Fractional value in the range [-1, (+ 1 – 2\(^{-15}\)]

Remarks: The conversion is performed using convergent rounding and saturation mechanisms.

Source File: flt2frct.c
Float2Fract (Continued)

Function Profile: System resources usage:
  W0..W7 used, not restored
  W8..W14 saved, used, restored

DO and REPEAT instruction usage:
  None

Program words (24-bit instructions):
  See the file “readme.txt” in pic30_tools\src\dsp for this information.

Cycles (including C-function call and return overheads):
  See the file “readme.txt” in pic30_tools\src\dsp for this information.
Chapter 3. 16-Bit Peripheral Libraries

3.1 INTRODUCTION

This chapter documents the functions and macros contained in the 16-bit peripheral libraries. Examples of use are also provided.

Code size for each library function or macro may be found in the file readme.txt in Program Files\Microchip\MPLAB C30\src\peripheral.

3.1.1 Assembly Code Applications

Free versions of these libraries and associated header files are available from the Microchip web site. Source code is included.

3.1.2 C Code Applications

The MPLAB C30 C compiler install directory (c:\Program Files\Microchip\MPLAB C30) contains the following subdirectories with library-related files:

- lib – 16-bit peripheral library files
- src\peripheral – source code for library functions and a batch file to rebuild the library
- support\h – header files for libraries

3.1.3 Chapter Organization

This chapter is organized as follows:

- Using the 16-Bit Peripheral Libraries

Software Functions
- External LCD Functions

Hardware Functions
- CAN Functions
- ADC12 Functions
- ADC10 Functions
- Timer Functions
- Reset/Control Functions
- I/O Port Functions
- Input Capture Functions
- Output Compare Functions
- UART Functions
- DCI Functions
- SPI Functions
- QE1 Functions
- PWM Functions
- I2C™ Functions
3.2 USING THE 16-BIT PERIPHERAL LIBRARIES

Building an application which utilizes the 16-bit Peripheral Libraries requires a processor-specific library file and a header file for each peripheral module.

For each peripheral, the corresponding header file provides all the function prototypes, #defines and typedefs used by the library. The archived library file contains all the individual object files for each library function.

The header files are of the form peripheral.h, where peripheral = name of the particular peripheral being used (e.g., can.h for CAN).

The library files are of the form libpDevice-omf.a, where Device = 16-bit device number (e.g., libp30F6014-coff.a for the dsPIC30F6014 device). See Section 1.2 “OMF-Specific Libraries/Start-up Modules” for more on OMF-specific libraries.

When compiling an application, header file must be referenced (using #include) by all source files which call a function in the library or use its symbols or typedefs. When linking an application, the library file must be provided as an input to the linker (using the --library or -l linker switch) such that the functions used by the application may be linked into the application.

The batch file makeplib.bat may be used to remake the libraries. The default behavior is to build peripheral libraries for all supported target processors; however, you may select a particular processor to build by naming it on the command line. For example:

makeplib.bat 30f6014

or

makeplib.bat 30F6014

will rebuild the library for the dsPIC30F6014 device.

3.3 EXTERNAL LCD FUNCTIONS

This section contains a list of individual functions for interfacing with P-tec PCOG1602B LCD controller and an example of use of the functions in this section. Functions may be implemented as macros.

The external LCD functions are only supported for the following devices:

- dsPIC30F5011
- dsPIC30F5013
- dsPIC30F6010
- dsPIC30F6011
- dsPIC30F6012
- dsPIC30F6013
- dsPIC30F6014
### 3.3.1 Individual Functions

#### BusyXLCD

<table>
<thead>
<tr>
<th>Description</th>
<th>This function checks for the busy flag of the P-tec PCOG1602B LCD controller.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td>xlcd.h</td>
</tr>
<tr>
<td>Prototype</td>
<td>char BusyXLCD(void);</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>If ‘1’ is returned, it indicates that the LCD controller is busy and can not take any command. If ‘0’ is returned, it indicates that the LCD is ready for next command.</td>
</tr>
<tr>
<td>Remarks</td>
<td>This function returns the status of the busy flag of the P-tec PCOG1602B LCD controller.</td>
</tr>
<tr>
<td>Source File</td>
<td>BusyXLCD.c</td>
</tr>
<tr>
<td>Code Example</td>
<td>while(BusyXLCD());</td>
</tr>
</tbody>
</table>

#### OpenXLCD

<table>
<thead>
<tr>
<th>Description</th>
<th>This function configures the I/O pins and initializes the P-tec PCOG1602B LCD controller.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td>xlcd.h</td>
</tr>
<tr>
<td>Prototype</td>
<td>void OpenXLCD (unsigned charlcdtype);</td>
</tr>
<tr>
<td>Arguments</td>
<td>lcdtype</td>
</tr>
<tr>
<td>This contains the LCD controller parameters to be configured as defined below:</td>
<td>Type of interface</td>
</tr>
<tr>
<td></td>
<td>FOUR_BIT</td>
</tr>
<tr>
<td></td>
<td>EIGHT_BIT</td>
</tr>
<tr>
<td>Number of lines</td>
<td>SINGLE_LINE</td>
</tr>
<tr>
<td></td>
<td>TWO_LINE</td>
</tr>
<tr>
<td>Segment data transfer direction</td>
<td>SEG1_50_SEG51_100</td>
</tr>
<tr>
<td></td>
<td>SEG1_50_SEG100_51</td>
</tr>
<tr>
<td></td>
<td>SEG100_51_SEG50_100</td>
</tr>
<tr>
<td></td>
<td>SEG100_51_SEG1_50</td>
</tr>
<tr>
<td>COM data transfer direction</td>
<td>COM1_COM16</td>
</tr>
<tr>
<td></td>
<td>COM16_COM1</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
<tr>
<td>Remarks</td>
<td>This function configures the I/O pins used to control the P-tec PCOG1602B LCD controller. It also initializes the LCD controller. The I/O pin definitions that must be made to ensure that the external LCD operates correctly are:</td>
</tr>
</tbody>
</table>
OpenXLCD (Continued)

Control I/O pin definitions

<table>
<thead>
<tr>
<th>PIN</th>
<th>Port Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>RW_PIN</td>
<td>PORTxbits.Rx?</td>
</tr>
<tr>
<td>TRIS_RW</td>
<td>TRISxbits.Rx?</td>
</tr>
<tr>
<td>RS_PIN</td>
<td>PORTxbits.Rx?</td>
</tr>
<tr>
<td>TRIS_RS</td>
<td>TRISxbits.Rx?</td>
</tr>
<tr>
<td>E_PIN</td>
<td>PORTxbits.Rx?</td>
</tr>
<tr>
<td>TRIS_E</td>
<td>TRISxbits.Rx?</td>
</tr>
</tbody>
</table>

where \( x \) is the PORT, \( ? \) is the pin number

Data pin definitions

<table>
<thead>
<tr>
<th>PIN</th>
<th>Port Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA_PIN</td>
<td>PORTxbits.RD?</td>
</tr>
<tr>
<td>TRIS_DATA_PIN</td>
<td>TRISxbits.TRISD?</td>
</tr>
</tbody>
</table>

where \( x \) is the PORT, \( ? \) is the pin number

The Data pins can be from either one port or from multiple ports.

The control pins can be on any port and are not required to be on the same port. The data interface must be defined as either 4-bit or 8-bit. The 8-bit interface is defined when a `#define EIGHT_BIT_INTERFACE` is included in the header file `xlcd.h`. If no define is included, then the 4-bit interface is included.

After these definitions have been made, the user must compile the application code into an object to be linked.

This function also requires three external routines for specific delays:

- `DelayFor18TCY()` - 18 Tcy delay
- `DelayPORXLCD()` - 15ms delay
- `DelayXLC()` - 5ms delay
- `Delay100XLC()` - 100Tcy delay

Source File: `openXLCD.c`

Code Example: `OpenXLCD(EIGHT_BIT & TWO_LINE & SEG1_50_SEG51_100 &COM1_COM16);`

**putsXLCD**

**putrsXLCD**

**Description:** This function writes a string of characters to the P-tec PCOG1602B LCD controller.

**Include:** `xlcd.h`

**Prototype:**

```c
void putsXLCD (char *buffer);
void putrsXLCD (const rom char *buffer);
```

**Arguments:**

- `buffer` - Pointer to the characters to be written to the LCD controller.

**Return Value:** None

**Remarks:** These functions write a string of characters located in `buffer` to the P-tec PCOG1602B LCD controller until a NULL character is encountered in the string.

For continuous display of data written to the P-tec PCOG1602B LCD controller, you could set up the display in a Shift mode.

Source File: `PutsXLCD.c`

Code Example: `char display_char[13]; putsXLCD(display_char);`
### ReadAddrXLCD

**Description:** This function reads the address byte from the P-tec PCOG1602B LCD controller.

**Include:** xlcd.h

**Prototype:**

```c
unsigned char ReadAddrXLCD (void);
```

**Arguments:** None

**Return Value:** This function returns an 8-bit which is the 7-bit address in the lower 7 bits of the byte and the BUSY status flag in the 8th bit.

**Remarks:** This function reads the address byte from the P-tec PCOG1602B LCD controller. The user must first check to see if the LCD controller is busy by calling the `BusyXLCD()` function. The address read from the controller is for the character generator RAM or the display data RAM depending on the previous `Set??RamAddr()` function that was called where ?? can be CG or DD.

**Source File:** ReadAddrXLCD.c

**Code Example:**

```c
char address;
while(BusyXLCD());
address = ReadAddrXLCD();
```

### ReadDataXLCD

**Description:** This function reads a data byte from the P-tec PCOG1602B LCD controller.

**Include:** xlcd.h

**Prototype:**

```c
char ReadDataXLCD (void);
```

**Arguments:** None

**Remarks:** This function reads a data byte from the P-tec PCOG1602B LCD controller. The user must first check to see if the LCD controller is busy by calling the `BusyXLCD()` function. The data read from the controller is for the character generator RAM or the display data RAM depending on the previous `Set??RamAddr()` function that was called where ?? is either CG or DD.

**Return Value:** This function returns the 8-bit data value pointed by the address.

**Source File:** ReadDataXLCD.c

**Code Example:**

```c
char data;
while (BusyXLCD());
data = ReadDataXLCD();
```
### SetCGRamAddr

**Description:**
This function sets the character generator address.

**Include:**
xlcd.h

**Prototype:**
```c
void SetCGRamAddr (unsigned char CGaddr);
```

**Arguments:**
- `CGaddr` Character generator address.

**Return Value:**
None

**Remarks:**
This function sets the character generator address of the P-tec PCOG1602B LCD controller. The user must first check to see if the controller is busy by calling the `BusyXLCD()` function.

**Source File:**
SetCGRamAddr.c

**Code Example:**
```c
char cgaddr = 0x1F;
while (BusyXLCD());
SetCGRamAddr(cgaddr);
```

### SetDDRamAddr

**Description:**
This function sets the display data address.

**Include:**
xlcd.h

**Prototype:**
```c
void SetDDRamAddr (unsigned char DDaddr);
```

**Arguments:**
- `DDaddr` Display data address.

**Return Value:**
None

**Remarks:**
This function sets the display data address of the P-tec PCOG1602B LCD controller. The user must first check to see if the controller is busy by calling the `BusyXLCD()` function.

**Source File:**
SetDDRamAddr.c

**Code Example:**
```c
char ddaddr = 0x10;
while (BusyXLCD());
SetDDRamAddr(ddaddr);
```

### WriteDataXLCD

**Description:**
This function writes a data byte (one character) from the P-tec PCOG1602B LCD controller.

**Include:**
xlcd.h

**Prototype:**
```c
void WriteDataXLCD (char data);
```

**Arguments:**
- `data` The value of `data` can be any 8-bit value, but should correspond to the character RAM table of the P-tec PCOG1602B LCD controller.

**Return Value:**
None

**Remarks:**
This function writes a data byte to the P-tec PCOG1602B LCD controller. The user must first check to see if the LCD controller is busy by calling the `BusyXLCD()` function.

The data read from the controller is for the character generator RAM or the display data RAM depending on the previous `Set??RamAddr()` function that was called where ?? refers to either CG or DD.

**Source File:**
WriteDataXLCD.c

**Code Example:**
```c
WriteDataXLCD(0x30);
```
WriteCmdXLCD

Description: This function writes a command to the P-tec PCOG1602B LCD controller.

Include: xlcd.h

Prototype: void WriteCmdXLCD (unsigned char cmd);

Arguments: cmd This contains the LCD controller parameters to be configured as defined below:

Type of interface
FOUR_BIT
EIGHT_BIT

Number of lines
SINGLE_LINE
TWO_LINE

Segment data transfer direction
SEG1_50_SEG1_100
SEG1_50_SEG100_51
SEG100_51_SEG50_1
SEG100_51_SEG1_50

COM data transfer direction
COM1_COM16
COM16_COM1

Display On/Off control
DON
DOFF
CURSOR_ON
CURSOR_OFF
BLINK_ON
BLINK_OFF

Cursor or Display Shift defines
SHIFT_CUR_LEFT
SHIFT_CUR_RIGHT
SHIFT_DISP_LEFT
SHIFT_DISP_RIGHT

Return Value: None

Remarks: This function writes the command byte to the P-tec PCOG1602B LCD controller. The user must first check to see if the LCD controller is busy by calling the BusyXLCD() function.

Source File: WriteCmdXLCD.c

Code Example:
while(BusyXLCD());
WriteCmdXLCD(EIGHT_BIT & TWO_LINE);
WriteCmdXLCD(DON);
WriteCmdXLCD(SHIFT_DISP_LEFT);
3.3.2 Example of Use

```c
#define __dsPIC30F6014__
#include <p30fxxxx.h>
#include<xlcd.h>
/* holds the address of message */
char * buffer;
char data;
char mesg1[] = {'H','A','R','W','A','R','E',\0};
char mesg2[] = {'P','E','R','I','P','H','E','R','A','L',' ', 'L','I','B',\0};

int main(void)
{
    /* Set 8bit interface and two line display */
    OpenXLCD(EIGHT_BIT & TWO_LINE & SEG1_50_SEGS1_100
             & COM1_COM16);
    /* Wait till LCD controller is busy */
    while(BusyXLCD());
    /* Turn on the display */
    WriteCmdXLCD(DON & CURSOR_ON & BLINK_OFF);
    buffer = mesg1;
    PutsXLCD(buffer);
    while(BusyXLCD());
    /* Set DDRam address to 0x40 to dispaly data in the second line */
    SetDDRamAddr(0x40);
    while(BusyXLCD());
    buffer = mesg2;
    PutsXLCD(buffer);
    while(BusyXLCD());
    return 0;
}
```
### 3.4 CAN FUNCTIONS

This section contains a list of individual functions for CAN and an example of use of the functions. Functions may be implemented as macros.

#### 3.4.1 Individual Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Include</th>
<th>Prototype</th>
<th>Arguments</th>
<th>Return Value</th>
<th>Remarks</th>
<th>Source File</th>
<th>Code Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN1AbortAll</td>
<td>initiates abort of all the pending transmissions.</td>
<td>can.h</td>
<td>void CAN1AbortAll(void);</td>
<td>None</td>
<td>None</td>
<td>This function sets the ABAT bit in CiCTRL register thus initiating the abort of all pending transmission. However, the transmission which is already in progress will not be aborted. This bit gets cleared by hardware when the message transmission has been successfully aborted.</td>
<td>CAN1AbortAll.c</td>
<td>CAN1AbortAll();</td>
</tr>
<tr>
<td>CAN2AbortAll</td>
<td></td>
<td></td>
<td>void CAN2AbortAll(void);</td>
<td>None</td>
<td>None</td>
<td></td>
<td>CAN2AbortAll.c</td>
<td></td>
</tr>
</tbody>
</table>
| CAN1GetRXErrorCount | returns the receive error count value.                                      | can.h    | unsigned char CAN1GetRXErrorCount(void);                                | None      | contents of CiERRCNT, which is 8 bits.                                                                                                                      | This function returns the contents of CiERRCNT (lower byte of CiEC register) which indicates the receive error count. | CAN1GetRXErrorCount.c | unsigned char rx_error_count;  
| CAN2GetRXErrorCount |                                                                              |          | unsigned char CAN2GetRXErrorCount(void);                                | None      | contents of CiERRCNT, which is 8 bits.                                                                                                                      | This function returns the contents of CiERRCNT (lower byte of CiEC register) which indicates the receive error count. | CAN2GetRXErrorCount.c | unsigned char rx_error_count = CAN1GetRXErrorCount();  

### CAN1GetTXErrorCount
#### Description:
This function returns the transmit error count value.

#### Include:
```c
#include "can.h"
```

#### Prototype:
```c
unsigned char CAN1GetTXErrorCount(void);
unsigned char CAN2GetTXErrorCount(void);
```

#### Arguments:
None

#### Return Value:
Contents of CiTERRCNT, which is 8 bits.

#### Remarks:
This function returns the contents of CiTERRCNT (upper byte of CiEC register) which indicates the transmit error count.

#### Source File:
- CAN1GetTXErrorCount.c
- CAN2GetTXErrorCount.c

#### Code Example:
```c
unsigned char tx_error_count;
tx_error_count = CAN1GetTXErrorCount();
```

### CAN1IsBusOff
#### Description:
This function determines whether the CAN node is in BusOff mode.

#### Include:
```c
#include "can.h"
```

#### Prototype:
```c
char CAN1IsBusOff(void);
char CAN2IsBusOff(void);
```

#### Arguments:
None

#### Return Value:
If the value of TXBO is '1', then '1' is returned, indicating that the bus has been turned off due to error in transmission.
If the value of TXBO is '0', then '0' is returned, indicating that the bus not been turned off.

#### Remarks:
This function returns the status of the TXBO bit of CiINTF register.

#### Source File:
- CAN1IsBusOff.c
- CAN2IsBusOff.c

#### Code Example:
```c
while(CAN1IsBusOff());
```

### CAN1IsRXReady
#### Description:
This function returns the receive buffer full status.

#### Include:
```c
#include "can.h"
```

#### Prototype:
```c
char CAN1IsRXReady(char);
char CAN2IsRXReady(char);
```

#### Arguments:
- `buffno` The value of buffno indicates the receive buffer whose status is required.

#### Return Value:
If RXFUL is 1, it indicates that the receive buffer contains a received message.
If RXFUL is 0, it indicates that the receive buffer is open to receive a new message.

#### Remarks:
This function returns the status of the RXFUL bit of Receive Control register.

#### Source File:
- CAN1IsRXReady.c
- CAN2IsRXReady.c

#### Code Example:
```c
char rx_1_status;
rx_1_status = CAN1IsRXReady(1);
```
### CAN1IsRXPassive

**Description:** This function determines if the receiver is in Error Passive state.

**Include:** `can.h`

**Prototype:**
```c
char CAN1IsRXPassive(void);
char CAN2IsRXPassive(void);
```

**Arguments:** None

**Return Value:**
- If the value of RXEP is ‘1’, then ‘1’ is returned, indicating the node going passive due to error in reception.
- If the value of RXEP is ‘0’, then ‘0’ is returned, indicating no error on bus.

**Remarks:** This function returns the status of the RXEP bit of CiINTF register.

**Source File:**
- `CAN1IsRXPassive.c`
- `CAN2IsRXPassive.c`

**Code Example:**
```c
char rx_bus_status;
rx_bus_status = CAN1IsRXPassive();
```

### CAN1IsTXPassive

**Description:** This function determines if the transmitter is in Error Passive state.

**Include:** `can.h`

**Prototype:**
```c
char CAN1IsTXPassive(void);
char CAN2IsTXPassive(void);
```

**Arguments:** None

**Return Value:**
- If the value of TXEP is ‘1’, then ‘1’ is returned, indicating error on transmit bus and the bus going passive.
- If the value of TXEP is ‘0’, then ‘0’ is returned, indicating no error on transmit bus.

**Remarks:** This function returns the status of the TXEP bit of CiINTF register.

**Source File:**
- `CAN1IsTXPassive.c`
- `CAN2IsTXPassive.c`

**Code Example:**
```c
char tx_bus_status;
tx_bus_status = CAN1IsTXPassive();
```
### CAN1IsTXReady
**Description:** This function returns the transmitter status indicating if the CAN node is ready for next transmission.

**Include:** `can.h`

**Prototype:**
```c
char CAN1IsTXReady(char);
char CAN2IsTXReady(char);
```

**Arguments:**
- `buffno` The value of `buffno` indicates the transmit buffer whose status is required.

**Return Value:**
- If `TXREQ` is `'1'`, it returns `'0'` indicating that the transmit buffer is not empty.
- If `TXREQ` is `'0'`, it returns `'1'` indicating that the transmit buffer is empty and the transmitter is ready for next transmission.

**Remarks:**
This function returns the compliment of the TXREQ Status bit in the Transmit Control register.

**Source File:**
- CAN1IsTXReady.c
- CAN2IsTXReady.c

**Code Example:**
```c
char tx_2_status;
tx_2_status = CAN1IsTXReady(2);
```

### CAN1ReceiveMessage
**Description:** This function read the data from the receive buffer.

**Include:** `can.h`

**Prototype:**
```c
void CAN1ReceiveMessage(unsigned char * data, unsigned char datalen, char MsgFlag);
void CAN2ReceiveMessage(unsigned char * data, unsigned char datalen, char MsgFlag);
```

**Arguments:**
- `data` The pointer to the location where received data is to be stored from.
- `datalen` The number of bytes of data expected.
- `MsgFlag` The buffer number where data is received.
  - If `'1'`, the data from CiRX1B1 to CiRX1B4 is read.
  - If `'0'` or otherwise, the data from CiRX0B1 to CiRX0B4 is read.

**Remarks:**
This function reads the received data into the locations pointed by input parameter `data`.

**Return Value:** None.

**Source File:**
- CAN1ReceiveMessage.c
- CAN2ReceiveMessage.c

**Code Example:**
```c
unsigned char*rx_data;
CAN1ReceiveMessage(rx_data, 5, 0);
```
**CAN1SendMessage**  
**CAN2SendMessage**

**Description:** This function writes data to be transmitted to TX registers, sets the data length and initiates the transmission.

**Include:**
```
#include "can.h"
```

**Prototype:**
```c
void CAN1SendMessage(unsigned int sid, unsigned long eid, unsigned char *data, unsigned char datalen, char MsgFlag);
void CAN2SendMessage(unsigned int sid, unsigned long eid, unsigned char *data, unsigned char datalen, char MsgFlag);
```

**Arguments:**
- **sid**: The 16-bit value to be written into CiTXnSID registers.
  - CAN_TX_SID(x) x is the required SID value.
  - Substitute Remote Request  
    - CAN_SUB_REM_TX_REQ  
    - CAN_SUB_NOR_TX_REQ
- **eid**: The 32-bit value to be written into CiTXnEID and CiTXnDLC registers.
  - CAN_TX_EID(x) x is the required EID value.
  - Substitute Remote Request  
    - CAN_REM_TX_REQ  
    - CAN_NOR_TX_REQ
- **data**: The pointer to the location where data to be transmitted is stored.
- **datalen**: The number of bytes of data to be transmitted.
- **MsgFlag**: The buffer number (‘0’, ‘1’ or ‘2’) from where data is transmitted.
  - If ‘1’, the data is written into CiTX1B1 to CiTX1B4.
  - If ‘2’, the data is written into CiTX2B1 to CiTX2B4.
  - If ‘0’ or otherwise, the data is written into CiTX0B1 to CiTX0B4.

**Return Value:** None

**Remarks:** This function writes the identifier values into SID and EID registers, data to be transmitted into TX reg, sets the data length and initiates transmission by setting TXREQ bit.

**Source File:**
- CAN1SendMessage.c
- CAN2SendMessage.c

**Code Example:**
```c
CAN1SendMessage((CAN_TX_SID(1920)) & (CAN_TX_EID_EN) & (CAN_SUB_NOR_TX_REQ),  
(CAN_TX_EID(12344)) & (CAN_NOR_TX_REQ),  
Txdata, datalen, tx_rx_no);
```
### CAN1SetFilter
### CAN2SetFilter

**Description:**
This function sets the acceptance filter values (SID and EID) for the specified filter.

**Include:**
can.h

**Prototype:**
```c
void CAN1SetFilter(char filter_no, unsigned int sid, unsigned long eid);
void CAN2SetFilter(char filter_no, unsigned int sid, unsigned long eid);
```

**Arguments:**
- **filter_no**
The filter (0, 1, 2, 3, 4 or 5) for which new filter values have to be configured.
- **sid**
The 16-bit value to be written into CiRXFnSID registers. `CAN_FILTER_SID(x)` `x` is the required SID value.
- **eid**
The 32-bit value to be written into CiRXFnEIDH and CiRXFnEIDL registers. `CAN_FILTER_EID(x)` `x` is the required EID value.

**Return Value:**
None

**Remarks:**
This function writes the 16-bit value of `sid` into the CiRXFnSID register and or the 32-bit value of `eid` into the CiRXFnEIDH and CiRXFnEIDL registers corresponding to the filter specified by `filter_no`. Filter 0 is taken as default.

**Source File:**
CAN1SetFilter.c  
CAN2SetFilter.c

**Code Example:**
```c
CAN1SetFilter(1, CAN_FILTER_SID(7) & CAN_RX_EID_EN, CAN_FILTER_EID(3));
```

---

### CAN1SetMask
### CAN2SetMask

**Description:**
This function sets the acceptance mask values (SID and EID) for the specified mask.

**Include:**
can.h

**Prototype:**
```c
void CAN1SetMask(char mask_no, unsigned int sid, unsigned long eid);
void CAN2SetMask(char mask_no, unsigned int sid, unsigned long eid);
```

**Arguments:**
- **mask_no**
The mask (‘0’ or ‘1’) for which mask values have to be configured.
- **sid**
The 16-bit value to be written into CiRXMnSID registers. `CAN_MASK_SID(x)` `x` is the required SID value.
- **eid**
The 32-bit value to be written into CiRXMnEIDH and CiRXMnEIDL registers. `CAN_MASK_EID(x)` `x` is the required EID value.

**Return Value:**
None
CAN1SetMask (Continued)
CAN2SetMask

Remarks: This function writes the 16-bit value of sid into the CiRXFnSID register and or the 32-bit value of eid into the CiRXFnEIDH and CiRXFnEIDL registers corresponding to the mask specified by mask_no. Filter 0 is taken as default.

Source File: CAN1SetMask.c
CAN2SetMask.c

Code Example: CAN1SetMask(1, CAN_MASK_SID(7) & CAN_MATCH_FILTER_TYPE, CAN_MASK_EID(3));

CAN1SetOperationMode
CAN2SetOperationMode

Description: This function configures the CAN module

Include: can.h

Prototype: void CAN1SetOperationMode(unsigned int config);
void CAN2SetOperationMode(unsigned int config);

Arguments: config The 16-bit value to be loaded into CiCTRL register, the combination of the following defines.

- CAN_IDLE_CON CAN On in Idle mode
- CAN_IDLE_STOP CAN Stop in Idle mode
- CAN_MASTERCLOCK_1 FCAN is FCY
- CAN_MASTERCLOCK_0 FCAN is 4 FCY

CAN modes of operation
- CAN_REQ_OPERMODE_NOR
- CAN_REQ_OPERMODE_DIS
- CAN_REQ_OPERMODE_LOOPBK
- CAN_REQ_OPERMODE_LISTENONLY
- CAN_REQ_OPERMODE_CONFIG
- CAN_REQ_OPERMODE_LISTENALL

CAN Capture Enable/Disable
- CAN_CAPTURE_EN
- CAN_CAPTURE_DIS

Return Value: None

Remarks: This function configures the following bits of CiCTRL::CSIDL, REQOP<2:0> and CANCKS

Source File: CAN1SetOperationMode.c
CAN2SetOperationMode.c

Code Example: CAN1SetOperationMode(CAN_IDLE_STOP & CAN_MASTERCLOCK_0 & CAN_REQ_OPERMODE_DIS & CAN_CAPTURE_DIS);
CAN1SetOperationModeNoWait
CAN2SetOperationModeNoWait

Description: This function aborts the pending transmissions and configures the CAN module.

Include: can.h

Prototype: void CAN1SetOperationModeNoWait(unsigned int config);
void CAN2SetOperationModeNoWait(unsigned int config);

Arguments: config The 16-bit value to be loaded into CiCTRL register, the combination of the following defines.
- CAN_IDLE_CONF_NO_WAIT: CAN On in Idle mode
- CAN_IDLE_STOP_NO_WAIT: CAN Stop in Idle mode
- CAN_MASTER_CLOCK_1_NO_WAIT: Fcan is Fcy
- CAN_MASTER_CLOCK_0_NO_WAIT: Fcan is 4 Fcy
- CAN modes of operation
  - CAN_REQ_OPERMODE_NOR_NO_WAIT
  - CAN_REQ_OPERMODE_DIS_NO_WAIT
  - CAN_REQ_OPERMODE_LOOPBK_NO_WAIT
  - CAN_REQ_OPERMODE_LISTENONLY_NO_WAIT
  - CAN_REQ_OPERMODE_CONFIG_NO_WAIT
  - CAN_REQ_OPERMODE_LISTENALL_NO_WAIT
- CAN Capture Enable/Disable
  - CAN_CAPTURE_EN_NO_WAIT
  - CAN_CAPTURE_DIS_NO_WAIT

Return Value: None

Remarks: This function sets the Abort bit thus initiating abort of all pending transmissions and configures the following bits of CiCTRL: - CSIDL, REQOP<2:0> and CANCKS

Source File: CAN1SetOperationModeNoWait.c
CAN2SetOperationModeNoWait.c

Code Example:
CAN1SetOperationModeNoWait(CAN_IDLE_CONF &
CAN_MASTER_CLOCK_1 & CAN_REQ_OPERMODE_LISTEN &
CAN_CAPTURE_DIS_NO_WAIT);

CAN1SetRXMode
CAN2SetRXMode

Description: This function configures the CAN receiver.

Include: can.h

Prototype: void CAN1SetRXMode(char buffno, unsigned int config);
void CAN2SetRXMode(char buffno, unsigned int config);

Arguments: buffno config
- buffno indicates the control reg to be configured.
- config The value to be written into CIRXnCON reg, the combination of the following defines.
  - Clear RXFUL bit
    - CAN_RXFUL_CLEAR
  - Double buffer enable/disable
    - CAN_BUF0_DBLBUFFER_EN
    - CAN_BUF0_DBLBUFFER_DIS
16-Bit Peripheral Libraries

### CAN1SetRXMode (Continued)
**CAN2SetRXMode**

| Return Value: | None |
| Remarks:      | This function configures the following bits of CIRXnCON register: RXRTR, RXFUL (only 0), RXM<1:0> and DBEN |
| Source File:  | CAN1SetRXMode.c |
| Source File:  | CAN2SetRXMode.c |
| Code Example: | CAN1SetRXMode(0,CAN_RXFUL_CLEAR & CAN_BUF0_DBLBUFFER_EN); |

### CAN1SetTXMode (function)
**CAN2SetTXMode**

| Description: | This function configures the CAN transmitter module |
| Include:     | can.h |
| Prototype:   | void CAN1SetTXMode(char buffno, unsigned int config); |
| Prototype:   | void CAN2SetTXMode(char buffno, unsigned int config); |
| Arguments:   | buffno  buffno indicates the control reg to be configured. |
|             | config  The value to be written into CiTXnCON reg, the combination of the following define. |
|             | Message send request |
|             | CAN_TX_REQ |
|             | CAN_TX_STOP_REQ |
|             | Message transmission priority |
|             | CAN_TX_PRIORITY_HIGH |
|             | CAN_TX_PRIORITY_HIGH_INTER |
|             | CAN_TX_PRIORITY_LOW_INTER |
|             | CAN_TX_PRIORITY_LOW |
| Return Value: | None |
| Remarks:     | This function configures the following bits of CiTXnCON register: TXRTR, TXREQ, DLC, TXPRI<1:0> |
| Source File: | CAN1SetTXMode.c |
| Source File: | CAN2SetTXMode.c |
| Code Example: | CAN1SetTXMode(1, CAN_TX_STOP_REQ & CAN_TX_PRIORITY_HIGH); |
CAN1Initialize
CAN2Initialize

Description: This function configures the CAN module

Include: can.h

Prototype:
void CAN1Initialize(unsigned int config1,
unsigned int config2);
void CAN2Initialize(unsigned int config1,
unsigned int config2);

Arguments:
config1 The value to be written into CiCFG1 register, the combination of the following defines.

 .Sync jump width
  CAN_SYNC_JUMP_WIDTH1
  CAN_SYNC_JUMP_WIDTH2
  CAN_SYNC_JUMP_WIDTH3
  CAN_SYNC_JUMP_WIDTH4

  Baud Rate prescaler
  CAN_BAUD_PRE_SCALE(x) (((x – 1) & 0x3f) | 0xC0)

config2 The value to be written into CiCFG2 register, the combination of the following defines.

  CAN bus line filter selection for wake-up
  CAN_WAKEUP_BY_FILTER_EN
  CAN_WAKEUP_BY_FILTER_DIS

  CAN propagation segment length
  CAN_PROPAGATIONTIME_SEG_TQ(x) (((x – 1) & 0x7) | 0xC7F8)

  CAN phase segment 1 length
  CAN_PHASE_SEG1_TQ(x) (((x – 1) & 0x7) * 0x8) | 0xC7C7)

  CAN phase segment 2 length
  CAN_PHASE_SEG2_TQ(x) (((x-1) & 0x7) * 0x100) | 0xC0FF)

  CAN phase segment 2 mode
  CAN_SEG2_FREE_PROG
  CAN_SEG2_TIME_LIMIT_SET

  Sample of the CAN bus line
  CAN_SAMPLE3TIMES
  CAN_SAMPLE1TIME

Return Value: None

Remarks: This function configures the following bits of CiCFG1 and CiCFG2 registers:

  SJW<1:0>, BRP<5:0>, CANCAP, WAKEFIL, SEG2PH<2:0>, SEGPHTS, SAM, SEG1PH<2:0>, PRSEG<2:0>

Source File:
CAN1Initialize.c
CAN2Initialize.c

Code Example:
CAN1Initialize(CAN_SYNC_JUMP_WIDTH2 & CAN_BAUD_PRE_SCALE(2),
CAN_WAKEUP_BY_FILTER_DIS & CAN_PHASE_SEG2_TQ(5) &
CAN_PHASE_SEG1_TQ(4) &
CAN_PROPAGATIONTIME_SEG_TQ(4) &
CAN_SEG2_FREE_PROG &
CAN_SAMPLE1TIME);
## ConfigIntCAN1

### Description:
This function configures the CAN interrupts.

### Include:
```
can.h
```

### Prototype:
```
void ConfigIntCAN1(unsigned int config1, unsigned int config2);
void ConfigIntCAN2(unsigned int config1, unsigned int config2);
```

### Arguments:
- `config1` individual interrupt enable/disable information as defined below:
  - **Interrupt enable**
    - CAN_INDI_INVMESS_EN
    - CAN_INDI_WAK_EN
    - CAN_INDI_ERR_EN
    - CAN_INDI_TXB2_EN
    - CAN_INDI_TXB1_EN
    - CAN_INDI_TXB0_EN
    - CAN_INDI_RXB1_EN
    - CAN_INDI_RXB0_EN
  - **Interrupt disable**
    - CAN_INDI_INVMESS_DIS
    - CAN_INDI_WAK_DIS
    - CAN_INDI_ERR_DIS
    - CAN_INDI_TXB2_DIS
    - CAN_INDI_TXB1_DIS
    - CAN_INDI_TXB0_DIS
    - CAN_INDI_RXB1_DIS
    - CAN_INDI_RXB0_DIS

- `config2` CAN interrupt priority and enable/disable information as defined below:
  - **CAN Interrupt enable/disable**
    - CAN_INT_ENABLE
    - CAN_INT_DISABLE
  - **CAN Interrupt priority**
    - CAN_INT_PRI_0
    - CAN_INT_PRI_1
    - CAN_INT_PRI_2
    - CAN_INT_PRI_3
    - CAN_INT_PRI_4
    - CAN_INT_PRI_5
    - CAN_INT_PRI_6
    - CAN_INT_PRI_7

### Return Value:
None

### Remarks:
This function configures the CAN interrupts. It enables/disables the individual CAN interrupts. It also enables/disables the CAN interrupt and sets priority.

### Source File:
- ConfigIntCAN1.c
- ConfigIntCAN2.c
3.4.2 Individual Macros

EnableIntCAN1
EnableIntCAN2

Description: This macro enables the CAN interrupt.
Include: can.h
Arguments: None
Remarks: This macro sets CAN Interrupt Enable bit of Interrupt Enable Control register.
Code Example: EnableIntCAN1;

DisableIntCAN1
DisableIntCAN2

Description: This macro disables the CAN interrupt.
Include: can.h
Arguments: None
Remarks: This macro clears CAN Interrupt Enable bit of Interrupt Enable Control register.
Code Example: DisableIntCAN2;

SetPriorityIntCAN1
SetPriorityIntCAN2

Description: This macro sets priority for CAN interrupt.
Include: can.h
Arguments: priority
Remarks: This macro sets CAN Interrupt Priority bits of Interrupt Priority Control register.
Code Example: SetPriorityIntCAN1(2);
3.4.3 Example of Use

```c
#define __dsPIC30F6014__
#include<p30fxxxx.h>
#include<can.h>
#define dataarray 0x1820
int main(void)
{
    /* Length of data to be transmitted/read */
    unsigned char datalen;
    unsigned char Txdata[] = {'M','I','C','R','O','C','H','I','P','\0'};
    unsigned int TXConfig, RXConfig;
    unsigned long MaskID, MessageID;
    char FilterNo, tx_rx_no;
    unsigned char * datareceived = (unsigned char *)
    dataarray;  /* Holds the data received */
    /* Set request for configuration mode */
    CAN1SetOperationMode(CAN_IDLE_CON &
        CAN_MASTER_CLOCK_1 &
        CAN_REQ_OPERMODE_CONFIG &
        CAN_CAPTURE_DIS);
    while(C1CTRLbits.OPMODE <= 3);
    /* Load configuration register */
    CAN1Initialize(CAN_SYNC_JUMP_WIDTH2 &
        CAN_BAUD_PRE_SCALE(2),
        CAN_WAKEUP_BY_FILTER_DIS &
        CAN_PHASE_SEG2_TQ(5) &
        CAN_PHASE_SEG1_TQ(4) &
        CAN_PROPAGATIONTIME_SEG_TQ(4) &
        CAN_SEG2_FREE_PROG &
        CAN_SAMPLEBITTIME);
    /* Load Acceptance filter register */
    FilterNo = 0;
    CAN1SetFilter(FilterNo, CAN_FILTER_SID(1920) &
        CAN_RX_EID_EN, CAN_FILTER_EID(12345));
    /* Load mask filter register */
    CAN1SetMask(FilterNo, CAN_MASK_SID(1920) &
        CAN_MATCH_FILTER_TYPE, CAN_MASK_EID(12344));
    /* Set transmitter and receiver mode */
    tx_rx_no = 0;
    CAN1SetTXMode(tx_rx_no,
        CAN_TX_STOP_REQ &
        CAN_TX_PRIORITY_HIGH );
    CAN1SetRXMode(tx_rx_no,
        CAN_RXFUL_CLEAR &
        CAN_BUF0_DBLBUFFER_EN);
    /* Load message ID, Data into transmit buffer and set transmit request bit */
    datalen = 8;
    CAN1SendMessage((CAN_TX_SID(1920)) & CAN_TX_EID_EN &
        CAN_SUB_NOR_TX_REQ,
        (CAN_TX_EID(12344)) & CAN_NOR_TX_REQ,
        Txdata, datalen, tx_rx_no);
```
/* Set request for Loopback mode */
CAN1SetOperationMode(CAN_IDLE_CON & CAN_CAPTURE_DIS & 
    CAN_MASTERCLOCK_1 & 
    CAN_REQ_OPERMODE_LOOPBK);

while(C1CTRLbits.OPMODE !=2);
/* Wait till message is transmitted completely */
while(!CAN1IsTXReady(0))
/* Wait till receive buffer contain valid message */
while(!CAN1IsRXReady(0));
/* Read received data from receive buffer and store it into 
    user defined dataarray */
CAN1ReceiveMessage(datareceived, datalen, tx_rx_no);
while(1);
return 0;
3.5 ADC12 FUNCTIONS

This section contains a list of individual functions for the 12-bit ADC and an example of use of the functions. Functions may be implemented as macros.

### 3.5.1 Individual Functions

#### BusyADC12

**Description:** This function returns the ADC conversion status.

**Include:** adc12.h

**Prototype:**

```c
char BusyADC12(void);
```

**Arguments:** None

**Return Value:** If the value of DONE is '0', then '1' is returned, indicating that the ADC is busy in conversion. If the value of DONE is '1', then '0' is returned, indicating that the ADC has completed conversion.

**Remarks:** This function returns the complement of the ADCON1 <DONE> bit status which indicates whether the ADC is busy in conversion.

**Source File:** BusyADC12.c

**Code Example:**

```c
while(BusyADC12());
```

#### CloseADC12

**Description:** This function turns off the ADC module and disables the ADC interrupts.

**Include:** adc12.h

**Prototype:**

```c
void CloseADC12(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:** This function first disables the ADC interrupt and then turns off the ADC module. The Interrupt Flag bit (ADIF) is also cleared.

**Source File:** CloseADC12.c

**Code Example:**

```c
CloseADC12();
```

#### ConfigIntADC12

**Description:** This function configures the ADC interrupt.

**Include:** adc12.h

**Prototype:**

```c
void ConfigIntADC12(unsigned int config);
```

**Arguments:** `config` ADC interrupt priority and enable/disable information as defined below:

```
ADC_INTERRUPT_ENABLE
ADC_INT_ENABLE
ADC_INT_DISABLE
```

**Source File:** CloseADC12.c

**Code Example:**

```c
CloseADC12();
```
### ConfigIntADC12 (Continued)

**ADC Interrupt priority**
- ADC_INT_PRI_0
- ADC_INT_PRI_1
- ADC_INT_PRI_2
- ADC_INT_PRI_3
- ADC_INT_PRI_4
- ADC_INT_PRI_5
- ADC_INT_PRI_6
- ADC_INT_PRI_7

**Return Value:** None  
**Remarks:** This function clears the Interrupt Flag (ADIF) bit and then sets the interrupt priority and enables/disables the interrupt.  
**Source File:** ConfigIntADC12.c  
**Code Example:** ConfigIntADC12(ADC_INT_PRI_6 & ADC_INT_ENABLE);

### ConvertADC12

**Description:** This function starts A/D conversion.  
**Include:** adc12.h  
**Prototype:** void ConvertADC12(void);  
**Arguments:** None  
**Return Value:** None  
**Remarks:** This function clears the ADCON1<SAMP> bit and thus stops sampling and starts conversion. This happens only when trigger source for the A/D conversion is selected as Manual, by clearing the ADCON1 <SSRC> bits.  
**Source File:** ConvertADC12.c  
**Code Example:** ConvertADC12();

### OpenADC12

**Description:** This function configures the ADC.  
**Include:** adc12.h  
**Prototype:** void OpenADC12(unsigned int config1,  
unsigned int config2,  
unsigned int config3,  
unsigned int configport,  
unsigned int configscan)  
**Arguments:** config1 This contains the parameters to be configured in the ADCON1 register as defined below:  
- **Module On/Off**  
  ADC_MODULE_ON  
  ADC_MODULE_OFF  
- **Idle mode operation**  
  ADC_IDLE_CONTINUE  
  ADC_IDLE_STOP
### OpenADC12 (Continued)

<table>
<thead>
<tr>
<th>Result output format</th>
<th>ADC_FORMAT_SIGN_FRACT</th>
<th>ADC_FORMAT_FRACT</th>
<th>ADC_FORMAT_SIGN_INT</th>
<th>ADC_FORMAT_INTG</th>
</tr>
</thead>
</table>

**Conversion trigger source**
- ADC_CLK_AUTO
- ADC_CLK_TMR
- ADC_CLK_INT0
- ADC_CLK_MANUAL

**Auto sampling select**
- ADC_AUTO_SAMPLING_ON
- ADC_AUTO_SAMPLING_OFF

**Sample enable**
- ADC_SAMP_ON
- ADC_SAMP_OFF

---

`config2` This contains the parameters to be configured in the ADCON2 register as defined below:

**Voltage Reference**
- ADC_VREF_AVDD_AVSS
- ADC_VREF_EXT_AVSS
- ADC_VREF_AVDD_EXT
- ADC_VREF_EXT_EXT

**Scan selection**
- ADC_SCAN_ON
- ADC_SCAN_OFF

**Number of samples between interrupts**
- ADC_SAMPLES_PER_INT_1
- ADC_SAMPLES_PER_INT_2
- ....
- ADC_SAMPLES_PER_INT_15
- ADC_SAMPLES_PER_INT_16

**Buffer mode select**
- ADC_ALT_BUF_ON
- ADC_ALT_BUF_OFF

Alternate Input Sample mode select
- ADC_ALT_INPUT_ON
- ADC_ALT_INPUT_OFF

---

`config3` This contains the parameters to be configured in the ADCON3 register as defined below:

**Auto Sample Time bits**
- ADC_SAMPLE_TIME_0
- ADC_SAMPLE_TIME_1
- ....
- ADC_SAMPLE_TIME_30
- ADC_SAMPLE_TIME_31

**Conversion Clock Source select**
- ADC_CONV_CLK_INTERNAL_RC
- ADC_CONV_CLK_SYSTEM
OpenADC12 (Continued)

Conversion clock select
ADC_CONV_CLK_Tcy2
ADC_CONV_CLK_Tcy
ADC_CONV_CLK_3Tcy2
....
ADC_CONV_CLK_32Tcy

configport This contains the pin select to be configured into the ADPCFG register as defined below:
ENABLE_ALL_ANA
ENABLE_ALL_DIG
ENABLE_AN0_ANA
ENABLE_AN1_ANA
ENABLE_AN2_ANA
....
ENABLE_AN15_ANA

configscan This contains the scan select parameter to be configured into the ADCSSL register as defined below:
SCAN_NONE
SCAN_ALL
SKIP_SCAN_AN0
SKIP_SCAN_AN1
....
SKIP_SCAN_AN15

Return Value: None

Remarks: This function configures the ADC for the following parameters:
Operating mode, Sleep mode behavior, Data o/p format, Sample Clk Source, VREF source, No of samples/int, Buffer Fill mode, Alternate i/p sample mod, Auto sample time, Conv clock source, Conv Clock Select bits, Port Config Control bits.

Source File: OpenADC12.c

Code Example:
OpenADC12(ADC_MODULE_OFF &
ADC_IDLE_CONTINUE &
ADC_FORMAT_INTG &
ADC_AUTO_SAMPLING_ON,
ADC_VREF_AVDD_AVSS &
ADC_SCAN_OFF &
ADC_BUF_MODE_OFF &
ADC_ALT_INPUT_ON &
ADC_SAMPLES_PER_INT_15,
ADC_SAMPLE_TIME_4 &
ADC_CONV_CLK_SYSTEM &
ADC_CONV_CLK_Tcy,
ENABLE_AN0_ANA,
SKIP_SCAN_AN1 &
SKIP_SCAN_AN2 &
SKIP_SCAN_AN5 &
SKIP_SCAN_AN7);
ReadADC12

Description: This function reads the ADC Buffer register which contains the conversion value.

Include: adc12.h

Prototype: unsigned int ReadADC12(unsigned char bufIndex);

Arguments: bufIndex This is the ADC buffer number which is to be read.

Return Value: None

Remarks: This function returns the contents of the ADC Buffer register. User should provide bufIndex value between 0 to 15 to ensure correct read of the ADCBUF0 to ADCBUFF register.

Source File: ReadADC12.c

Code Example:

```
unsigned int result;
result = ReadADC12(5);
```

StopSampADC12

Description: This function is identical to ConvertADC12.

Source File:
```
#define to ConvertADC12 in adc12.h
```

SetChanADC12

Description: This function sets the positive and negative inputs for sample multiplexers A and B.

Include: adc12.h

Prototype: void SetChanADC12(unsigned int channel);

Arguments: channel This contains the input select parameter to be configured into ADCHS register as defined below:

- A/D Channel 0 positive i/p select for SAMPLE A
  - ADC_CH0_POS_SAMPLEA_AN0
  - ADC_CH0_POS_SAMPLEA_AN1
  - ...
  - ADC_CH0_POS_SAMPLEA_AN15

- A/D Channel 0 negative i/p select for SAMPLE A
  - ADC_CH0_NEG_SAMPLEA_AN1
  - ADC_CH0_NEG_SAMPLEA_NVREF

- A/D Channel 0 positive i/p select for SAMPLE B
  - ADC_CH0_POS_SAMPLEB_AN0
  - ADC_CH0_POS_SAMPLEB_AN1
  - ...
  - ADC_CH0_POS_SAMPLEB_AN15

- A/D Channel 0 negative i/p select for SAMPLE B
  - ADC_CH0_NEG_SAMPLEB_AN1
  - ADC_CH0_NEG_SAMPLEB_NVREF

Return Value: None

Remarks: This function configures the inputs for the sample multiplexers A and B by writing to the ADCHS register.

Source File: SetChanADC12.c

Code Example:
```
SetChanADC12(ADC_CH0_POS_SAMPLEA_AN4 &
        ADC_CH0_NEG_SAMPLEA_NVREF);
```
3.5.2 Individual Macros

EnableIntADC

| Description: | This macro enables the ADC interrupt. |
| Include:     | adc12.h |
| Arguments:   | None |
| Remarks:     | This macro sets ADC Interrupt Enable bit of Interrupt Enable Control register. |
| Code Example:| EnableIntADC; |

DisableIntADC

| Description: | This macro disables the ADC interrupt. |
| Include:     | adc12.h |
| Arguments:   | None |
| Remarks:     | This macro clears ADC Interrupt Enable bit of Interrupt Enable Control register. |
| Code Example:| DisableIntADC; |

SetPriorityIntADC

| Description: | This macro sets priority for ADC interrupt. |
| Include:     | adc12.h |
| Arguments:   | priority |
| Remarks:     | This macro sets ADC Interrupt Priority bits of Interrupt Priority Control register. |
| Code Example:| SetPriorityIntADC(6); |
3.5.3 Example of Use

```c
#define __dsPIC30F6014__
#include <p30fxxxx.h>
#include<adc12.h>

unsigned int Channel, PinConfig, Scanselect, Adcon3_reg, Adcon2_reg,
Adcon1_reg;
int main(void)
{
    unsigned int result[20], i;
    ADCON1bits.ADON = 0; /* turn off ADC */
    Channel = ADC_CH0_POS_SAMPLEA_AN4 &
               ADC_CH0_NEG_SAMPLEA_NVREF &
               ADC_CH0_POS_SAMPLEB_AN2 &
               ADC_CH0_NEG_SAMPLEB_AN1;
    SetChanADC12(Channel);
    ConfigIntADC12(ADC_INT_DISABLE);
    PinConfig = ENABLE_AN4_ANA;
    Scanselect = SKIP_SCAN_AN2 & SKIP_SCAN_AN5 &
                 SKIP_SCAN_AN9 & SKIP_SCAN_AN10 &
                 SKIP_SCAN_AN14 & SKIP_SCAN_AN15;
    Adcon3_reg = ADC_SAMPLE_TIME_10 &
                 ADC_CONV_CLK_SYSTEM &
                 ADC_CONV_CLK_13Tcy;
    Adcon2_reg = ADC_VREF_AVDD_AVSS &
                 ADC_SCAN_OFF &
                 ADC_ALT_BUF_OFF &
                 ADC_ALT_INPUT_OFF &
                 ADC_SAMPLES_PER_INT_16;
    Adcon1_reg = ADC_MODULE_ON &
                 ADC_IDLE_CONTINUE &
                 ADC_FORMAT_INTG &
                 ADC_CLK_MANUAL &
                 ADC_AUTO_SAMPLING_OFF;
    OpenADC12(Adcon1_reg, Adcon2_reg,
               Adcon3_reg,PinConfig, Scanselect);
    i = 0;
    while( i < 16 )
    {
        ADCON1bits.SAMP = 1;
        while(!ADCON1bits.SAMP);
        ConvertADC12();
        while(ADCON1bits.SAMP);
        while(!BusyADC12());
        while(BusyADC12());
        result[i] = ReadADC12(i);
        i++;
    }
}
```

3.6 ADC10 FUNCTIONS

This section contains a list of individual functions for the 10-bit ADC and an example of use of the functions. Functions may be implemented as macros.

3.6.1 Individual Functions

BusyADC10

Description: This function returns the ADC conversion status.
Include: adc10.h
Prototype: char BusyADC10(void);
Arguments: None
Return Value: If the value of DONE is '0', then '1' is returned, indicating that the ADC is busy in conversion.
If the value of DONE is '1', then '0' is returned, indicating that the ADC has completed conversion.
Remarks: This function returns the complement of the ADCON1 <DONE> bit status which indicates whether the ADC is busy in conversion.
Source File: BusyADC10.c
Code Example: while(BusyADC10());

CloseADC10

Description: This function turns off the ADC module and disables the ADC interrupts.
Include: adc10.h
Prototype: void CloseADC10(void);
Arguments: None
Return Value: None
Remarks: This function first disables the ADC interrupt and then turns off the ADC module. The Interrupt Flag bit (ADIF) is also cleared.
Source File: CloseADC10.c
Code Example: CloseADC10();

ConfigIntADC10

Description: This function configures the ADC interrupt.
Include: adc10.h
Prototype: void ConfigIntADC10(unsigned int config);
Arguments: config ADC interrupt priority and enable/disable information as defined below:
ADC Interrupt enable/disable
ADC_INT_ENABLE
ADC_INT_DISABLE
ADC Interrupt priority
ADC_INT_PRI_0
ADC_INT_PRI_1
ADC_INT_PRI_2
ADC_INT_PRI_3
ADC_INT_PRI_4
ADC_INT_PRI_5
ADC_INT_PRI_6
ADC_INT_PRI_7

CloseADC10

Description: This function turns off the ADC module and disables the ADC interrupts.
Include: adc10.h
Prototype: void CloseADC10(void);
Arguments: None
Return Value: None
Remarks: This function first disables the ADC interrupt and then turns off the ADC module. The Interrupt Flag bit (ADIF) is also cleared.
Source File: CloseADC10.c
Code Example: CloseADC10();

ConfigIntADC10

Description: This function configures the ADC interrupt.
Include: adc10.h
Prototype: void ConfigIntADC10(unsigned int config);
Arguments: config ADC interrupt priority and enable/disable information as defined below:
ADC Interrupt enable/disable
ADC_INT_ENABLE
ADC_INT_DISABLE
ADC Interrupt priority
ADC_INT_PRI_0
ADC_INT_PRI_1
ADC_INT_PRI_2
ADC_INT_PRI_3
ADC_INT_PRI_4
ADC_INT_PRI_5
ADC_INT_PRI_6
ADC_INT_PRI_7
### ConfigIntADC10 (Continued)

<table>
<thead>
<tr>
<th>Return Value:</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remarks:</td>
<td>This function clears the Interrupt Flag (ADIF) bit and then sets the interrupt priority and enables/disables the interrupt.</td>
</tr>
<tr>
<td>Source File:</td>
<td>ConfigIntADC10.c</td>
</tr>
<tr>
<td>Code Example:</td>
<td>ConfigIntADC10(ADC_INT_PRI_3 &amp; ADC_INT_DISABLE);</td>
</tr>
</tbody>
</table>

### ConvertADC10

<table>
<thead>
<tr>
<th>Description:</th>
<th>This function starts the A/D conversion.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>adc10.h</td>
</tr>
<tr>
<td>Prototype:</td>
<td>void ConvertADC10(void);</td>
</tr>
<tr>
<td>Arguments:</td>
<td>None</td>
</tr>
<tr>
<td>Return Value:</td>
<td>None</td>
</tr>
<tr>
<td>Remarks:</td>
<td>This function clears the ADCON1&lt; Samp &gt; bit and thus stops sampling and starts conversion. This happens only when trigger source for the A/D conversion is selected as Manual, by clearing the ADCON1&lt; SSRC &gt; bits.</td>
</tr>
<tr>
<td>Source File:</td>
<td>ConvertADC10.c</td>
</tr>
<tr>
<td>Code Example:</td>
<td>ConvertADC10();</td>
</tr>
</tbody>
</table>

### OpenADC10

<table>
<thead>
<tr>
<th>Description:</th>
<th>This function configures the ADC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>adc10.h</td>
</tr>
<tr>
<td>Prototype:</td>
<td>void OpenADC10(unsigned int config1, unsigned int config2, unsigned int config3, unsigned int configport, unsigned int configscan)</td>
</tr>
<tr>
<td>Arguments:</td>
<td>config1 This contains the parameters to be configured in the ADCON1 register as defined below: Module On/Off AD(CON MODULE ON AD(CON MODULE OFF</td>
</tr>
</tbody>
</table>
OpenADC10 (Continued)

**Auto sampling select**
ADC_AUTO_SAMPLING_ON
ADC_AUTO_SAMPLING_OFF

**Simultaneous Sampling**
ADC_SAMPLE_SIMULTANEOUS
ADC_SAMPLE_INDIVIDUAL

**Sample enable**
ADC_SAMP_ON
ADC_SAMP_OFF

This contains the parameters to be configured in the ADCON2 register as defined below:

**Voltage Reference**
ADC_VREF_AVDD_AVSS
ADC_VREF_EXT_AVSS
ADC_VREF_AVDD_EXT
ADC_VREF_EXT_EXT

**Scan selection**
ADC_SCAN_ON
ADC_SCAN_OFF

**A/D channels utilized**
ADC_CONVERT_CH0123
ADC_CONVERT_CH01
ADC_CONVERT_CH0

**Number of samples between interrupts**
ADC_SAMPLES_PER_INT_1
ADC_SAMPLES_PER_INT_2
....
ADC_SAMPLES_PER_INT_15
ADC_SAMPLES_PER_INT_16

**Buffer mode select**
ADC_ALT_BUF_ON
ADC_ALT_BUF_OFF

**Alternate Input Sample mode select**
ADC_ALT_INPUT_ON
ADC_ALT_INPUT_OFF

This contains the parameters to be configured in the ADCON3 register as defined below:

**Auto Sample Time bits**
ADC_SAMPLE_TIME_0
ADC_SAMPLE_TIME_1
....
ADC_SAMPLE_TIME_30
ADC_SAMPLE_TIME_31

**Conversion Clock Source select**
ADC_CONV_CLK_INTERNAL_RC
ADC_CONV_CLK_SYSTEM

**Conversion clock select**
ADC_CONV_CLK_Tcy2
ADC_CONV_CLK_Tcy
ADC_CONV_CLK_3Tcy2
....
ADC_CONV_CLK_32Tcy
OpenADC10 (Continued)

<table>
<thead>
<tr>
<th>configport</th>
<th>This contains the pin select to be configured into the ADPCFG register as defined below:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ENABLE_ALL_ANA</td>
</tr>
<tr>
<td></td>
<td>ENABLE_ALL_DIG</td>
</tr>
<tr>
<td></td>
<td>ENABLE_AN0_ANA</td>
</tr>
<tr>
<td></td>
<td>ENABLE_AN1_ANA</td>
</tr>
<tr>
<td></td>
<td>ENABLE_AN2_ANA</td>
</tr>
<tr>
<td></td>
<td>.....</td>
</tr>
<tr>
<td></td>
<td>ENABLE_AN15_ANA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>configscan</th>
<th>This contains the scan select parameter to be configured into the ADCSSL register as defined below:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SCAN_NONE</td>
</tr>
<tr>
<td></td>
<td>SCAN_ALL</td>
</tr>
<tr>
<td></td>
<td>SKIP_SCAN_AN0</td>
</tr>
<tr>
<td></td>
<td>SKIP_SCAN_AN1</td>
</tr>
<tr>
<td></td>
<td>......</td>
</tr>
<tr>
<td></td>
<td>SKIP_SCAN_AN15</td>
</tr>
</tbody>
</table>

Return Value: None

Remarks: This function configures the ADC for the following parameters:
Operating mode, Sleep mode behavior, Data o/p format, Sample Clk Source, VREF source, No of samples/int, Buffer Fill mode, Alternate i/p sample mod, Auto sample time, Conv clock source, Conv Clock Select bits, Port Config Control bits.

Source File: OpenADC10.c

Code Example: OpenADC10(ADC_MODULE_OFF &
ADC_IDLE_STOP &
ADC_FORMAT_SIGN_FRACT &
ADC_CLK_INT0 &
ADC_SAMPLE_INDIVIDUAL &
ADC_AUTO_SAMPLING_ON,
ADC_VREF_AVDD_AVSS &
ADC_SCAN_OFF &
ADC_BUF_MODE_OFF &
ADC_ALT_INPUT_ON &
ADC_CONVERT_CH0 &
ADC_SAMPLES_PER_INT_10,
ADC_SAMPLE_TIME_4 &
ADC_CONV_CLK_SYSTEM &
ADC_CONV_CLK_Tcy,
ENABLE_AN1_ANA,
SKIP_SCAN_AN0 &
SKIP_SCAN_AN3 &
SKIP_SCAN_AN4 &
SKIP_SCAN_AN5);
## ReadADC10

**Description:** This function reads the ADC Buffer register which contains the conversion value.

**Include:**
adc10.h

**Prototype:**
unsigned int ReadADC10(unsigned char bufIndex);

**Arguments:**
bufIndex This is the ADC buffer number which is to be read.

**Return Value:** None

**Remarks:** This function returns the contents of the ADC Buffer register. User should provide bufIndex value between 0 to 15 to ensure correct read of the ADCBUF0 to ADCBUFF.

**Source File:** ReadADC10.c

### Code Example

```c
unsigned int result;
result = ReadADC10(3);
```

## StopSampADC10

**Description:** This function is identical to ConvertADC10.

**Source File:** #define to ConvertADC10 in adc10.h

## SetChanADC10

**Description:** This function sets the positive and negative inputs for the sample multiplexers A and B.

**Include:**
adc10.h

**Prototype:**
void SetChanADC10(unsigned int channel);

**Arguments:** channel This contains the input select parameter to be configured into the ADCHS register as defined below:

- **A/D Channel 1, 2, 3 Negative input for Sample A**
  - ADC_CHX_NEG_SAMPLEA_AN9AN10AN11
  - ADC_CHX_NEG_SAMPLEA_AN6AN7AN8
  - ADC_CHX_NEG_SAMPLEA_NVREF

- **A/D Channel 1, 2, 3 Negative input for Sample B**
  - ADC_CHX_NEG_SAMPLEB_AN9AN10AN11
  - ADC_CHX_NEG_SAMPLEB_AN6AN7AN8
  - ADC_CHX_NEG_SAMPLEB_NVREF

- **A/D Channel 1, 2, 3 Positive input for Sample A**
  - ADC_CHX_POS_SAMPLEA_AN3AN4AN5
  - ADC_CHX_POS_SAMPLEA_AN0AN1AN2

- **A/D Channel 1, 2, 3 Positive input for Sample B**
  - ADC_CHX_POS_SAMPLEB_AN3AN4AN5
  - ADC_CHX_POS_SAMPLEB_AN0AN1AN2

- **A/D Channel 0 positive i/p select for Sample A**
  - ADC_CH0_POS_SAMPLEA_AN0
  - ADC_CH0_POS_SAMPLEA_AN1
  - ...
  - ADC_CH0_POS_SAMPLEA_AN15

- **A/D Channel 0 negative i/p select for Sample A**
  - ADC_CH0_NEG_SAMPLEA_AN1
  - ADC_CH0_NEG_SAMPLEA_NVREF
### 3.6.2 Individual Macros

#### EnableIntADC

**Description:** This macro enables the ADC interrupt.

**Include:** adc10.h

**Arguments:** None

**Remarks:** This macro sets ADC Interrupt Enable bit of Interrupt Enable Control register.

**Code Example:**

```
EnableIntADC();
```

#### DisableIntADC

**Description:** This macro disables the ADC interrupt.

**Include:** adc10.h

**Arguments:** None

**Remarks:** This macro clears ADC Interrupt Enable bit of Interrupt Enable Control register.

**Code Example:**

```
DisableIntADC();
```

#### SetPriorityIntADC

**Description:** This macro sets priority for ADC interrupt.

**Include:** adc10.h

**Arguments:** priority

**Remarks:** This macro sets ADC Interrupt Priority bits of Interrupt Priority Control register.

**Code Example:**

```
SetPriorityIntADC(2);
```
3.6.3 Example of Use

#define __dsPIC30F6010__
#include <p30fxxxx.h>
#include<adc10.h>
unsigned int Channel, PinConfig, Scanselect, Adcon3_reg, Adcon2_reg,
Adcon1_reg;
int  main(void)
{
    unsigned int result[20], i;
    ADCON1bits.ADON = 0;         /* turn off ADC */
    Channel = ADC_CH0_POS_SAMPLEA_AN4 &
              ADC_CH0_NEG_SAMPLEA_NVREF &
              ADC_CH0_POS_SAMPLEB_AN2 &
              ADC_CH0_NEG_SAMPLEB_AN1;
    SetChanADC1(Channel);
    ConfigIntADC10(ADC_INT_DISABLE);
    PinConfig  = ENABLE_AN4_ANA;
    Scanselect = SKIP_SCAN_AN2 & SKIP_SCAN_AN5 &
                 SKIP_SCAN_AN9 & SKIP_SCAN_AN10 &
                 SKIP_SCAN_AN14 & SKIP_SCAN_AN15;
    Adcon3_reg = ADC_SAMPLE_TIME_10 &
                 ADC_CONV_CLK_SYSTEM &
                 ADC_CONV_CLK_13Tcy;
    Adcon2_reg = ADC_VREF_AVDD_AVSS &
                 ADC_SCAN_OFF &
                 ADC_ALT_BUF_OFF &
                 ADC_ALT_INPUT_OFF &
                 ADC_CONVERT_CH0123 &
                 ADC_SAMPLES_PER_INT_16;
    Adcon1_reg = ADC_MODULE_ON &
                 ADC_IDLE_CONTINUE &
                 ADC_FORMAT_INTG &
                 ADC_CLK_MANUAL &
                 ADC_SAMPLE_SIMULTANEOUS &
                 ADC_AUTO_SAMPLING_OFF;
    OpenADC10(Adcon1_reg, Adcon2_reg,
              Adcon3_reg,PinConfig, Scanselect);
    i = 0;
    while(i <16 )
    {
        ADCON1bits.SAMP = 1;
        while(!ADCON1bits.SAMP);
        ConvertADC10();
        while(ADCON1bits.SAMP);
        while(!BusyADC10());
        while(BusyADC10());
        result[i] = ReadADC10(i);
        i++;}
}
### 3.7 TIMER FUNCTIONS

This section contains a list of individual functions for Timer and an example of use of the functions. Functions may be implemented as macros.

#### 3.7.1 Individual Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Include</th>
<th>Prototype</th>
<th>Arguments</th>
<th>Return Value</th>
<th>Remarks</th>
<th>Source File</th>
<th>Code Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloseTimer1</td>
<td>This function turns off the 16-bit timer module.</td>
<td>timer.h</td>
<td>void CloseTimer1(void);</td>
<td>None</td>
<td>None</td>
<td>This function first disables the 16-bit timer interrupt and then turns off the timer module. The Interrupt Flag bit (TxIF) is also cleared.</td>
<td>CloseTimer1.c</td>
<td>CloseTimer1();</td>
</tr>
<tr>
<td>CloseTimer2</td>
<td></td>
<td></td>
<td>void CloseTimer2(void);</td>
<td>None</td>
<td>None</td>
<td></td>
<td>CloseTimer2.c</td>
<td></td>
</tr>
<tr>
<td>CloseTimer3</td>
<td></td>
<td></td>
<td>void CloseTimer3(void);</td>
<td>None</td>
<td>None</td>
<td></td>
<td>CloseTimer3.c</td>
<td></td>
</tr>
<tr>
<td>CloseTimer4</td>
<td></td>
<td></td>
<td>void CloseTimer4(void);</td>
<td>None</td>
<td>None</td>
<td></td>
<td>CloseTimer4.c</td>
<td></td>
</tr>
<tr>
<td>CloseTimer5</td>
<td></td>
<td></td>
<td>void CloseTimer5(void);</td>
<td>None</td>
<td>None</td>
<td></td>
<td>CloseTimer5.c</td>
<td></td>
</tr>
</tbody>
</table>

| CloseTimer23 | This function turns off the 32-bit timer module. | timer.h    | void CloseTimer23(void)                        | None      | None         | This function disables the 32-bit timer interrupt and then turns off the timer module. The Interrupt Flag bit (TxIF) is also cleared. | CloseTimer23.c        | CloseTimer23();            |
| CloseTimer45 |                                                |            | void CloseTimer45(void)                        | None      | None         | CloseTimer23 turns off Timer2 and enables Timer3 interrupt.             | CloseTimer45.c        |

### Code Example:

```c
CloseTimer1();
CloseTimer23();
CloseTimer45();
```
ConfigIntTimer1
ConfigIntTimer2
ConfigIntTimer3
ConfigIntTimer4
ConfigIntTimer5

Description: This function configures the 16-bit timer interrupt.

Include: timer.h

Prototype:
void ConfigIntTimer1(unsigned int config);
void ConfigIntTimer2(unsigned int config);
void ConfigIntTimer3(unsigned int config);
void ConfigIntTimer4(unsigned int config);
void ConfigIntTimer5(unsigned int config);

Arguments: config Timer interrupt priority and enable/disable information as defined below:

- Tx_INT_PRIOR_7
- Tx_INT_PRIOR_6
- Tx_INT_PRIOR_5
- Tx_INT_PRIOR_4
- Tx_INT_PRIOR_3
- Tx_INT_PRIOR_2
- Tx_INT_PRIOR_1
- Tx_INT_PRIOR_0
- Tx_INT_ON
- Tx_INT_OFF

Return Value: None

Remarks: This function clears the 16-bit Interrupt Flag (TxIF) bit and then sets the interrupt priority and enables/disables the interrupt.

Source File: ConfigIntTimer1.c
ConfigIntTimer2.c
ConfigIntTimer3.c
ConfigIntTimer4.c
ConfigIntTimer5.c

Code Example: ConfigIntTimer1(T1_INT_PRIOR_3 & T1_INT_ON);
**ConfigIntTimer23**  
**ConfigIntTimer45**

**Description:**  
This function configures the 32-bit timer interrupt.

**Include:**  
timer.h

**Prototype:**  
void ConfigIntTimer23(unsigned int config);
void ConfigIntTimer45(unsigned int config);

**Arguments:**  
`config` Timer interrupt priority and enable/disable information as defined below:

* Tx_INT_PRIOR_7
* Tx_INT_PRIOR_6
* Tx_INT_PRIOR_5
* Tx_INT_PRIOR_4
* Tx_INT_PRIOR_3
* Tx_INT_PRIOR_2
* Tx_INT_PRIOR_1
* Tx_INT_PRIOR_0

* Tx_INT_ON
* Tx_INT_OFF

**Return Value:**  
None

**Remarks:**  
This function clears the 32-bit Interrupt Flag (TxIF) bit and then sets the interrupt priority and enables/disables the interrupt.

**Source File:**  
ConfigIntTimer23.c  
ConfigIntTimer45.c

**Code Example:**  
ConfigIntTimer23(T3_INT_PRIOR_5 & T3_INT_ON);

---

**OpenTimer1**  
**OpenTimer2**  
**OpenTimer3**  
**OpenTimer4**  
**OpenTimer5**

**Description:**  
This function configures the 16-bit timer module.

**Include:**  
timer.h

**Prototype:**  
void OpenTimer1(unsigned int config,  
unsigned int period);
void OpenTimer2(unsigned int config,  
unsigned int period);
void OpenTimer3(unsigned int config,  
unsigned int period);
void OpenTimer4(unsigned int config,  
unsigned int period);
void OpenTimer5(unsigned int config,  
unsigned int period);

**Arguments:**  
`config` This contains the parameters to be configured in the TxCON register as defined below:

* Timer Module On/Off
  * Tx_ON
  * Tx_OFF

* Timer Module Idle mode On/Off
  * Tx_IDLE_CON
  * Tx_IDLE_STOP
### OpenTimer1 (Continued)

**Description:**
This function configures the 16-bit Timer Control register and sets the period match value into the PR register.

**Prototype:**
```c
void OpenTimer1(T1_ON & T1_GATE_OFF & T1_PS_1_8 & T1_SYNC_EXT_OFF & T1_SOURCE_INT, 0xFF);
```

**Arguments:**
- `config`: This contains the parameters to be configured in the TxCON register as defined below:
  - **Timer module On/Off**
    - `Tx.ON`
    - `Tx.OFF`
  - **Timer Module Idle mode On/Off**
    - `Tx_IDLE_CON`
    - `Tx_IDLE_STOP`
  - **Timer Gate time accumulation enable**
    - `Tx_GATE_ON`
    - `Tx_GATE_OFF`
  - **Timer prescaler**
    - `Tx_PS_1_1`
    - `Tx_PS_1_8`
    - `Tx_PS_1_64`
    - `Tx_PS_1_128`
  - **Timer Synchronous clock enable**
    - `Tx_SYNC_EXT_ON`
    - `Tx_SYNC_EXT_OFF`
  - **Timer clock source**
    - `Tx_SOURCE_EXT`
    - `Tx_SOURCE_INT`
  - **period**: This contains the period match value to be stored into the PR register.

**Source File:**
- `OpenTimer1.c`
- `OpenTimer2.c`
- `OpenTimer3.c`
- `OpenTimer4.c`
- `OpenTimer5.c`

**Return Value:**
None

**Remarks:**
This function configures the 16-bit Timer Control register and sets the period match value into the PR register.

---

### OpenTimer23

**Description:**
This function configures the 32-bit timer module.

**Include:**
```c
#include "timer.h"
```

**Prototype:**
```c
void OpenTimer23(unsigned int config, unsigned long period);
```

**Arguments:**
- `config`: This contains the parameters to be configured in the TxCON register as defined below:
  - **Timer module On/Off**
    - `Tx.ON`
    - `Tx.OFF`
  - **Timer Module Idle mode On/Off**
    - `Tx_IDLE_CON`
    - `Tx_IDLE_STOP`
  - **Timer Gate time accumulation enable**
    - `Tx_GATE_ON`
    - `Tx_GATE_OFF`
```

---

### OpenTimer45

**Description:**
This function configures the 32-bit timer module.

**Prototype:**
```c
void OpenTimer45(unsigned int config, unsigned long period);
```

**Arguments:**
- `config`: This contains the parameters to be configured in the TxCON register as defined below:
  - **Timer module On/Off**
    - `Tx.ON`
    - `Tx.OFF`
  - **Timer Module Idle mode On/Off**
    - `Tx_IDLE_CON`
    - `Tx_IDLE_STOP`
  - **Timer Gate time accumulation enable**
    - `Tx_GATE_ON`
    - `Tx_GATE_OFF`
```
OpenTimer23 (Continued)
OpenTimer45

Timer prescaler
Tx_PS_1_1
Tx_PS_1_8
Tx_PS_1_64
Tx_PS_1_128

Timer Synchronous clock enable
Tx_SYNC_EXT_ON
Tx_SYNC_EXT_OFF

Timer clock source
Tx_SOURCE_EXT
Tx_SOURCE_INT

period  This contains the period match value to be stored into the
32-bit PR register.

Return Value:  None
Remarks:  This function configures the 32-bit Timer Control register and sets the
period match value into the PR register

Source File:  OpenTimer23.c
OpenTimer45.c

Code Example:  OpenTimer23 (T2_ON & T2_GATE_OFF &
T2_PS_1_8 & T2_32BIT_MODE_ON &
T2_SYNC_EXT_OFF &
T2_SOURCE_INT, 0xFFFF);

ReadTimer1
ReadTimer2
ReadTimer3
ReadTimer4
ReadTimer5

Description:  This function reads the contents of the 16-bit Timer register.

Include:  timer.h

Prototype:  unsigned int ReadTimer1(void);
unsigned int ReadTimer2(void);
unsigned int ReadTimer3(void);
unsigned int ReadTimer4(void);
unsigned int ReadTimer5(void);

Arguments:  None
Return Value:  None
Remarks:  This function returns the contents of the 16-bit TMR register.

Source File:  ReadTimer1.c
ReadTimer2.c
ReadTimer3.c
ReadTimer4.c
ReadTimer5.c

Code Example:  unsigned int timer1_value;
timer1_value = ReadTimer1();
### ReadTimer23

**Description:** This function reads the contents of the 32-bit Timer register.

**Include:**
```
timer.h
```

**Prototype:**
```
unsigned long ReadTimer23(void);
unsigned long ReadTimer45(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:** This function returns the contents of the 32-bit TMR register.

**Source File:**
```
ReadTimer23.c
ReadTimer45.c
```

**Code Example:**
```
unsigned long timer23_value;
timer23_value = ReadTimer23();
```

### WriteTimer1

**Description:** This function writes the 16-bit value into the Timer register.

**Include:**
```
timer.h
```

**Prototype:**
```
void WriteTimer1(unsigned int timer);
void WriteTimer2(unsigned int timer);
void WriteTimer3(unsigned int timer);
void WriteTimer4(unsigned int timer);
void WriteTimer5(unsigned int timer);
```

**Arguments:**
```
timer    This is the 16-bit value to be stored into TMR register.
```

**Return Value:** None

**Remarks:** None

**Source File:**
```
WriteTimer1.c
WriteTimer2.c
WriteTimer3.c
WriteTimer4.c
WriteTimer5.c
```

**Code Example:**
```
unsigned int timer_init = 0xAB;
WriteTimer1(timer_init);
```
### WriteTimer23

**Description:** This function writes the 32-bit value into the Timer register.

**Include:**

```c
#include "timer.h"
```

**Prototype:**

```c
void WriteTimer23(unsigned long timer);
void WriteTimer45(unsigned long timer);
```

**Arguments:**

- **timer**  
  This is the 32-bit value to be stored into TMR register.

**Return Value:** None

**Remarks:** None

**Source File:**

- `WriteTimer23.c`
- `WriteTimer45.c`

**Code Example:**

```c
unsigned long timer23_init = 0xABCD;
WriteTimer23(timer23_init);
```

### 3.7.2 Individual Macros

#### EnableIntT1

**Description:** This macro enables the timer interrupt.

**Include:**

```c
#include "timer.h"
```

**Arguments:** None

**Remarks:** This macro sets Timer Interrupt Enable bit of Interrupt Enable Control register.

**Code Example:**

```c
EnableIntT1;
```

#### DisableIntT1

**Description:** This macro disables the timer interrupt.

**Include:**

```c
#include "timer.h"
```

**Arguments:** None

**Remarks:** This macro clears Timer Interrupt Enable bit of Interrupt Enable Control register.

**Code Example:**

```c
DisableIntT1;
```
3.7.3 Example of Use

```c
#define __dsPIC30F6014__
#include <p30fxxxx.h>
#include<timer.h>

unsigned int timer_value;

void __attribute__((__interrupt__)) _T1Interrupt(void)
{
    PORTDbits.RD1 = 1;    /* turn off LED on RD1 */
    WriteTimer1(0);
    IFS0bits.T1IF = 0;    /* Clear Timer interrupt flag */
}

int main(void)
{
    unsigned int match_value;
    TRISDbits.TRISD1 = 0;
    PORTDbits.RD1 = 1;    /* turn off LED on RD1 */
    /* Enable Timer1 Interrupt and Priority to "1" */
    ConfigIntTimer1(T1_INT_PRIOR_1 & T1_INT_ON);
    WriteTimer1(0);
    match_value = 0xFFF;
    OpenTimer1(T1_ON & T1_GATE_OFF & T1_IDLE_STOP &
                T1_PS_1_1 & T1_SYNC_EXT_OFF &
                T1_SOURCE_INT, match_value);
    /* Wait till the timer matches with the period value */
    while(1)
    {
        timer_value = ReadTimer1();
        if(timer_value >= 0x7FF)
        {
            PORTDbits.RD1 = 0; /* turn on LED on RD1 */
        }
    }
    CloseTimer1();
}
```

---

**SetPriorityIntT1, SetPriorityIntT2, SetPriorityIntT3, SetPriorityIntT4, SetPriorityIntT5**

**Description:** This macro sets priority for timer interrupt.

**Include:** timer.h

**Arguments:** priority

**Remarks:** This macro sets Timer Interrupt Priority bits of Interrupt Priority Control register.

**Code Example:** SetPriorityIntT4(7);
3.8  RESET/CONTROL FUNCTIONS

This section contains a list of individual functions for Reset/Control. Functions may be implemented as macros.

3.8.1  Individual Functions

**isBOR**

| Description:  | This function checks if Reset is due to Brown-out Reset. |
| Include:      | reset.h                                                  |
| Prototype:    | char isBOR(void);                                       |
| Arguments:    | None                                                     |
| Return Value: | This function returns the RCON<BOR> bit status.          |
| Remarks:      | None                                                     |
| Source File:  | isBOR.c                                                  |
| Code Example: | char reset_state;                                        |
|              | reset_state = isBOR();                                  |

**isPOR**

| Description:  | This function checks if Reset is due to Power-on Reset. |
| Include:      | reset.h                                                  |
| Prototype:    | char isPOR(void);                                       |
| Arguments:    | None                                                     |
| Return Value: | This function returns the RCON<POR> bit status.          |
| Remarks:      | None                                                     |
| Source File:  | isPOR.c                                                  |
| Code Example: | char reset_state;                                        |
|              | reset_state = isPOR();                                  |

**isLVD**

| Description:  | This function checks if low-voltage detect interrupt flag is set. |
| Include:      | reset.h                                                  |
| Prototype:    | char isLVD(void);                                       |
| Arguments:    | None                                                     |
| Return Value: | This function returns the IFS2<LVDIF> bit status.         |
| Remarks:      | None                                                     |
| Source File:  | isLVD.c                                                  |
| Code Example: | char lvd;                                                |
|              | lvd = isLVD();                                          |
### isMCLR

**Description:**
This function checks if Reset condition is due to MCLR pin going low.

**Include:**
reset.h

**Prototype:**
```c
char isMCLR(void);
```

**Arguments:**
None

**Return Value:**
This function returns the RCON<EXTR> bit status.
- If return value is ‘1’, then Reset occurred due to MCLR pin going low.
- If return value is ‘0’, then Reset is not due to MCLR going low.

**Remarks:**
None

**Source File:**
isMCLR.c

**Code Example:**
```c
char reset_state;
reset_state = isMCLR();
```

### isWDTTO

**Description:**
This function checks if Reset condition is due to WDT time-out.

**Include:**
reset.h

**Prototype:**
```c
char isWDTTO(void);
```

**Arguments:**
None

**Return Value:**
This function returns the RCON<WDTO> bit status.
- If return value is ‘1’, then reset occurred due to WDT time-out.
- If return value is ‘0’, then reset is not due to WDT time-out.

**Remarks:**
None

**Source File:**
isWDTTO.c

**Code Example:**
```c
char reset_state;
reset_state = isWDTTO();
```

### isWDTWU

**Description:**
This function checks if Wake-up from Sleep is due to WDT time-out.

**Include:**
reset.h

**Prototype:**
```c
char isWDTWU(void);
```

**Arguments:**
None

**Return Value:**
This function returns the status of RCON<WDTO> and RCON<SLEEP>bits
- If return value is ‘1’, then Wake-up from Sleep occurred due to WDT time-out.
- If return value is ‘0’, then Wake-up from Sleep is not due to WDT time-out.

**Remarks:**
None

**Source File:**
isWDTWU.c

**Code Example:**
```c
char reset_state;
reset_state = isWDTWU();
```
### isWU

**Description:** This function checks if Wake-up from Sleep is due to MCLR, POR, BOR or any interrupt.

**Include:** reset.h

**Prototype:**

```c
char isWU(void);
```

**Arguments:** None

**Return Value:** This function checks if Wake-up from Sleep has occurred. If yes, it checks for the cause for wake-up. If ‘1’, wake-up is due to the occurrence of interrupt. If ‘2’, wake-up is due to MCLR. If ‘3’, wake-up is due to POR. If ‘4’, wake-up is due to BOR. If Wake-up from Sleep has not occurred, then a value of ‘0’ is returned.

**Remarks:** None

**Source File:** isWU.c

**Code Example:**

```c
char reset_state;
reset_state = isWU();
```

### 3.8.2 Individual Macros

#### DisableInterrupts

**Description:** This macro disables all the peripheral interrupts for specified number of instruction cycles.

**Include:** reset.h

**Arguments:** cycles

**Remarks:** This macro executes DISI instruction to disable all the peripheral interrupts for specified number of instruction cycles.

**Code Example:** DisableInterrupts(15);

#### PORStatReset

**Description:** This macro sets POR bit of RCON register to Reset state.

**Include:** reset.h

**Arguments:** None

**Remarks:** None

**Code Example:** PORStatReset;

#### BORStatReset

**Description:** This macro sets BOR bit of RCON register to Reset state.

**Include:** reset.h

**Arguments:** None

**Remarks:** None

**Code Example:** BORStatReset;
3.9 I/O PORT FUNCTIONS

This section contains a list of individual functions for I/O ports. Functions may be implemented as macros.

3.9.1 Individual Functions

CloseINT0
CloseINT1
CloseINT2
CloseINT3
CloseINT4

Description: This function disables the external interrupt on INT pin.
Include: ports.h
Prototype:
void CloseINT0(void);
void CloseINT1(void);
void CloseINT2(void);
void CloseINT3(void);
void CloseINT4(void);
Arguments: None
Return Value: None
Remarks: This function disables the interrupt on INT pin and clears the corresponding Interrupt flag.
Source File:
CloseInt0.c
CloseInt1.c
CloseInt2.c
CloseInt3.c
CloseInt4.c
Code Example: CloseINT0();
ConfigINT0
ConfigINT1
ConfigINT2
ConfigINT3
ConfigINT4

Description:  This function configures the interrupt on INT pin.
Include:  ports.h
Prototype:  void ConfigINT0(unsigned int config);
          void ConfigINT1(unsigned int config);
          void ConfigINT2(unsigned int config);
          void ConfigINT3(unsigned int config);
          void ConfigINT4(unsigned int config);

Arguments:  config  Interrupt edge, priority and enable/disable information as
defined below:
  Interrupt edge selection
  RISING_EDGE_INT
  FALLING_EDGE_INT
  Interrupt enable
  INT_ENABLE
  INT_DISABLE
  Interrupt priority
  INT_PRI_0
  INT_PRI_1
  INT_PRI_2
  INT_PRI_3
  INT_PRI_4
  INT_PRI_5
  INT_PRI_6
  INT_PRI_7

Return Value:  None
Remarks:  This function clears the interrupt flag corresponding to the INTx pin and
then selects the edge detect polarity.
It then sets the interrupt priority and enables/disables the interrupt.
Source File:  ConfigInt0.c
          ConfigInt1.c
          ConfigInt2.c
          ConfigInt3.c
          ConfigInt4.c

Code Example:  ConfigINT0(RISING_EDGE_INT & EXT_INT_PRI_5 &
EXT_INT_ENABLE);
### ConfigCNPullups

| Description: | This function configures the pull-up resistors for CN pins. |
| Include: | ports.h |
| Prototype: | `void ConfigCNPullups(long int config);` |
| Arguments: | `config` This is the 32-bit value for configuring pull-ups. The lower word is stored into CNPU1 register and next upper word is stored into CNPU2 register. The upper 8 bits of CNPU2 register are unimplemented. |
| Return Value: | None |
| Remarks: | None |
| Source File: | ConfigCNPullups.c |
| Code Example: | `ConfigCNPullups(0xFFF);` |

### ConfigIntCN

| Description: | This function configures the CN interrupts. |
| Include: | ports.h |
| Prototype: | `void ConfigIntCN(long int config);` |
| Arguments: | `config` This is the 32-bit value for configuring the CN interrupts. The lower 24 bits contain the individual enable/disable information for the CN interrupts. Setting bit $x$ ($x = 0, 1, ..., 23$) would enable the CN$x$ interrupt. The upper most byte of config contains the Interrupt Priority and Enable/Disable bits. The lower word is stored into the CNEN1 register and next upper byte is stored into the CNEN2 register and the upper most byte is used for setting priority and enable/disable the CN interrupts. |
| Return Value: | None |
| Remarks: | This function clears the CN interrupt flag and enables/disables the individual interrupts on CN pins. This also configures the interrupt priority and enables/disables the CN Interrupt Enable bit. |
| Source File: | ConfigIntCN.c |
| Code Example: | // This would enable CN0, CN1, CN2 and CN7 only. ConfigIntCN(CHANG_INT_OFF & CHANGE_INT_PRI_4 & 0xFFFF000087); |
3.9.2 Individual Macros

EnableCN0
EnableCN1
EnableCN2

....................
EnableCN23

Description: This macro enables the individual change notification interrupt.
Include: ports.h
Arguments: None
Remarks: None
Code Example: EnableCN6;

DisableCN0
DisableCN1
DisableCN2

....................
DisableCN23

Description: This macro disables individual change notification interrupt.
Include: ports.h
Arguments: None
Remarks: None
Code Example: DisableCN14;

EnableINT0
EnableINT1
EnableINT2
EnableINT3
EnableINT4

Description: This macro enables the individual external interrupt.
Include: ports.h
Arguments: None
Remarks: None
Code Example: EnableINT2;
DisableINT0
DisableINT1
DisableINT2
DisableINT3
DisableINT4

**Description:** This macro disables the individual external interrupt.
**Include:** ports.h
**Arguments:** None
**Remarks:** None
**Code Example:** DisableINT2;

SetPriorityInt0
SetPriorityInt1
SetPriorityInt2
SetPriorityInt3
SetPriorityInt4

**Description:** This macro sets priority for external interrupts.
**Include:** ports.h
**Arguments:** priority
**Remarks:** This macro sets External Interrupt Priority bits of Interrupt Priority Control register.
**Code Example:** SetPriorityInt4(6);
### 3.10 INPUT CAPTURE FUNCTIONS

This section contains a list of individual functions for Input Capture module and an example of use of the functions. Functions may be implemented as macros.

#### 3.10.1 Individual Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Include</th>
<th>Prototype</th>
<th>Arguments</th>
<th>Return Value</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloseCapture1</td>
<td>This function turns off the Input Capture module.</td>
<td>InCap.h</td>
<td>void CloseCapture1(void);</td>
<td>None</td>
<td>None</td>
<td>This function disables the Input Capture interrupt and then turns off the module. The Interrupt Flag bit is also cleared.</td>
</tr>
<tr>
<td>CloseCapture2</td>
<td></td>
<td></td>
<td>void CloseCapture2(void);</td>
<td>None</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>CloseCapture3</td>
<td></td>
<td></td>
<td>void CloseCapture3(void);</td>
<td>None</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>CloseCapture4</td>
<td></td>
<td></td>
<td>void CloseCapture4(void);</td>
<td>None</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>CloseCapture5</td>
<td></td>
<td></td>
<td>void CloseCapture5(void);</td>
<td>None</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>CloseCapture6</td>
<td></td>
<td></td>
<td>void CloseCapture6(void);</td>
<td>None</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>CloseCapture7</td>
<td></td>
<td></td>
<td>void CloseCapture7(void);</td>
<td>None</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>CloseCapture8</td>
<td></td>
<td></td>
<td>void CloseCapture8(void);</td>
<td>None</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

**Source File:**
- CloseCapture1.c
- CloseCapture2.c
- CloseCapture3.c
- CloseCapture4.c
- CloseCapture5.c
- CloseCapture6.c
- CloseCapture7.c
- CloseCapture8.c

**Code Example:**

```c
CloseCapture1();
```
## ConfigIntCapture1

**Description:** This function configures the Input Capture interrupt.

**Include:**

InCap.h

**Prototype:**

```c
void ConfigIntCapture1(unsigned int config);
void ConfigIntCapture2(unsigned int config);
void ConfigIntCapture3(unsigned int config);
void ConfigIntCapture4(unsigned int config);
void ConfigIntCapture5(unsigned int config);
void ConfigIntCapture6(unsigned int config);
void ConfigIntCapture7(unsigned int config);
void ConfigIntCapture8(unsigned int config);
```

**Arguments:**

`config` Input Capture interrupt priority and enable/disable information as defined below:

- **Interrupt enable/disable**
  - IC_INT_ON
  - IC_INT_OFF

- **Interrupt Priority**
  - IC_INT_PRIOR_0
  - IC_INT_PRIOR_1
  - IC_INT_PRIOR_2
  - IC_INT_PRIOR_3
  - IC_INT_PRIOR_4
  - IC_INT_PRIOR_5
  - IC_INT_PRIOR_6
  - IC_INT_PRIOR_7

**Return Value:** None

**Remarks:** This function clears the Interrupt Flag bit and then sets the interrupt priority and enables/disables the interrupt.

**Source File:**

- ConfigIntCapture1.c
- ConfigIntCapture2.c
- ConfigIntCapture3.c
- ConfigIntCapture4.c
- ConfigIntCapture5.c
- ConfigIntCapture6.c
- ConfigIntCapture7.c
- ConfigIntCapture8.c

**Code Example:**

```c
ConfigIntCapture1(IC_INT_ON & IC_INT_PRIOR_1);
```
OpenCapture1
OpenCapture2
OpenCapture3
OpenCapture4
OpenCapture5
OpenCapture6
OpenCapture7
OpenCapture8

**Description:** This function configures the Input Capture module.

**Include:**

InCap.h

**Prototype:**

void OpenCapture1(unsigned int config);
void OpenCapture2(unsigned int config);
void OpenCapture3(unsigned int config);
void OpenCapture4(unsigned int config);
void OpenCapture5(unsigned int config);
void OpenCapture6(unsigned int config);
void OpenCapture7(unsigned int config);
void OpenCapture8(unsigned int config);

**Arguments:**

*config* This contains the parameters to be configured in the ICxCON register as defined below:

- **Idle mode operation**
  - IC_IDLE_CON
  - IC_IDLE_STOP

- **Clock select**
  - IC_TIMER2_SRC
  - IC_TIMER3_SRC

- **Captures per interrupt**
  - IC_INT_4CAPTURE
  - IC_INT_3CAPTURE
  - IC_INT_2CAPTURE
  - IC_INT_1CAPTURE
  - IC_INTERRUPT

- **IC mode select**
  - IC_EVERY_EDGE
  - IC_EVERY_16_RISE_EDGE
  - IC_EVERY_4_RISE_EDGE
  - IC_EVERY_RISE_EDGE
  - IC_EVERY_FALL_EDGE
  - IC_INPUTCAP_OFF

**Return Value:** None

**Remarks:** This function configures the Input Capture Module Control register (ICxCON) with the following parameters: Clock select, Captures per interrupt, Capture mode of operation.

**Source File:**

OpenCapture1.c
OpenCapture2.c
OpenCapture3.c
OpenCapture4.c
OpenCapture5.c
OpenCapture6.c
OpenCapture7.c
OpenCapture8.c

**Code Example:**

OpenCapture1(IC_IDLE_CON & IC_TIMER2_SRC & IC_INT_1CAPTURE & IC_EVERY_RISE_EDGE);
### ReadCapture1
**Description:** This function reads all the pending Input Capture buffers.
**Include:** InCap.h
**Prototype:**
```c
void ReadCapture1(unsigned int *buffer);
void ReadCapture2(unsigned int *buffer);
void ReadCapture3(unsigned int *buffer);
void ReadCapture4(unsigned int *buffer);
void ReadCapture5(unsigned int *buffer);
void ReadCapture6(unsigned int *buffer);
void ReadCapture7(unsigned int *buffer);
void ReadCapture8(unsigned int *buffer);
```
**Arguments:**
- `buffer` This is the pointer to the locations where the data read from the Input Capture buffers have to be stored.
**Return Value:** None
**Remarks:** This function reads all the pending Input Capture buffers until the buffers are empty indicated by the ICxCON<ICBNE> bit getting cleared.
**Source File:**
- ReadCapture1.c
- ReadCapture2.c
- ReadCapture3.c
- ReadCapture4.c
- ReadCapture5.c
- ReadCapture6.c
- ReadCapture7.c
- ReadCapture8.c
**Code Example:**
```c
unsigned int *buffer = 0x1900;
ReadCapture1(buffer);
```
3.10.2 Individual Macros

EnableIntIC1
EnableIntIC2
EnableIntIC3
EnableIntIC4
EnableIntIC5
EnableIntIC6
EnableIntIC7
EnableIntIC8

Description: This macro enables the interrupt on capture event.
Include: InCap.h
Arguments: None
Remarks: This macro sets Input Capture Interrupt Enable bit of Interrupt Enable Control register.
Code Example: EnableIntIC7;

DisableIntIC1
DisableIntIC2
DisableIntIC3
DisableIntIC4
DisableIntIC5
DisableIntIC6
DisableIntIC7
DisableIntIC8

Description: This macro disables the interrupt on capture event.
Include: InCap.h
Arguments: None
Remarks: This macro clears Input Capture Interrupt Enable bit of Interrupt Enable Control register.
Code Example: DisableIntIC7;

SetPriorityIntIC1
SetPriorityIntIC2
SetPriorityIntIC3
SetPriorityIntIC4
SetPriorityIntIC5
SetPriorityIntIC6
SetPriorityIntIC7
SetPriorityIntIC8

Description: This macro sets priority for input capture interrupt.
Include: InCap.h
Arguments: priority
Remarks: This macro sets Input Capture Interrupt Priority bits of Interrupt Priority Control register.
Code Example: SetPriorityIntIC4(1);
3.10.3 Example of Use

```c
#define __dsPIC30F6014__
#include <p30fxxxx.h>
#include<InCap.h>
int Interrupt_Count = 0 , Int_flag, count;
unsigned int timer_first_edge, timer_second_edge;
void __attribute__((__interrupt__)) _IC1Interrupt(void)
{
    Interrupt_Count++;
    if(Interrupt_Count == 1)
        ReadCapture1(&timer_first_edge);
    else if(Interrupt_Count == 2)
        ReadCapture1(&timer_second_edge);
    Int_flag = 1;
    IFS0bits.IC1IF = 0;
}
int main(void)
{
    unsigned int period;
    Int_flag = 0;
    TRISDbits.TRISD0 = 0; /* Alarm output on RD0 */
    PORTDbits.RD0 = 1;
    /* Enable Timer1 Interrupt and Priority to '1' */
    ConfigIntCapture1(IC_INT_PRIOR_1 & IC_INT_ON);
    T3CON = 0x8000; /* Timer 3 On */
    /* Configure the InputCapture in stop in idle mode , Timer 3 as source , interrupt on capture 1, I/C on every fall edge */
    OpenCapture1(IC_IDLE_STOP & IC_TIMER3_SRC &
                 IC_INT_ICAPTURE & IC_EVERY_FALL_EDGE);
    while(1)
    {
        while(!Int_flag); /* wait here till first capture event */
        Int_flag = 0;
        while(!Int_flag); /* wait here till next capture event */
        /* calculate time count between two capture events */
        period = timer_second_edge - timer_first_edge;
        /* if the time count between two capture events is more than 0x200 counts, set alarm on RD0 */
        if(period >= 0x200)
        {
            /* set alarm and wait for sometime and clear alarm */
            PORTDbits.RD0 = 0;
            while(count <= 0x10)
            {
                count++;
            }
            PORTDbits.RD0 = 1;
            Interrupt_Count = 0;
            count = 0;
        }
        CloseCapture1();
    }
```
### 3.11 OUTPUT COMPARE FUNCTIONS

This section contains a list of individual functions for Output Compare module and an example of use of the functions. Functions may be implemented as macros.

#### 3.11.1 Individual Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Include</th>
<th>Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloseOC1</td>
<td>This function turns off the Output Compare module.</td>
<td>outcompare.h</td>
<td>void CloseOC1(void);</td>
</tr>
<tr>
<td>CloseOC2</td>
<td></td>
<td></td>
<td>void CloseOC2(void);</td>
</tr>
<tr>
<td>CloseOC3</td>
<td></td>
<td></td>
<td>void CloseOC3(void);</td>
</tr>
<tr>
<td>CloseOC4</td>
<td></td>
<td></td>
<td>void CloseOC4(void);</td>
</tr>
<tr>
<td>CloseOC5</td>
<td></td>
<td></td>
<td>void CloseOC5(void);</td>
</tr>
<tr>
<td>CloseOC6</td>
<td></td>
<td></td>
<td>void CloseOC6(void);</td>
</tr>
<tr>
<td>CloseOC7</td>
<td></td>
<td></td>
<td>void CloseOC7(void);</td>
</tr>
<tr>
<td>CloseOC8</td>
<td></td>
<td></td>
<td>void CloseOC8(void);</td>
</tr>
</tbody>
</table>

**Arguments:** None  
**Return Value:** None  
**Remarks:** This function disables the Output Compare interrupt and then turns off the module. The Interrupt Flag bit is also cleared.

**Source File:** CloseOC1.c  
CloseOC2.c  
CloseOC3.c  
CloseOC4.c  
CloseOC5.c  
CloseOC6.c  
CloseOC7.c  
CloseOC8.c

**Code Example:** CloseOC1();
ConfigIntOC1
ConfigIntOC2
ConfigIntOC3
ConfigIntOC4
ConfigIntOC5
ConfigIntOC6
ConfigIntOC7
ConfigIntOC8

Description: This function configures the Output Compare interrupt.

Include: outcompare.h

Prototype:
void ConfigIntOC1(unsigned int config);
void ConfigIntOC2(unsigned int config);
void ConfigIntOC3(unsigned int config);
void ConfigIntOC4(unsigned int config);
void ConfigIntOC5(unsigned int config);
void ConfigIntOC6(unsigned int config);
void ConfigIntOC7(unsigned int config);
void ConfigIntOC8(unsigned int config);

Arguments: config
Output Compare interrupt priority and enable/disable information as defined below:

- Interrupt enable/disable
  - OC_INT_ON
  - OC_INT_OFF

- Interrupt Priority
  - OC_INT_PRIOR_0
  - OC_INT_PRIOR_1
  - OC_INT_PRIOR_2
  - OC_INT_PRIOR_3
  - OC_INT_PRIOR_4
  - OC_INT_PRIOR_5
  - OC_INT_PRIOR_6
  - OC_INT_PRIOR_7

Return Value: None

Remarks: This function clears the Interrupt Flag bit and then sets the interrupt priority and enables/disables the interrupt.

Source File:
ConfigIntOC1.c
ConfigIntOC2.c
ConfigIntOC3.c
ConfigIntOC4.c
ConfigIntOC5.c
ConfigIntOC6.c
ConfigIntOC7.c
ConfigIntOC8.c

Code Example:
ConfigIntOC1(OC_INT_ON & OC_INT_PRIOR_2);
### Description:
This function configures the Output Compare module.

### Include:
- `outcompare.h`

### Prototype:
- `void OpenOC1(unsigned int config, unsigned int value1, unsigned int value2);`
- `void OpenOC2(unsigned int config, unsigned int value1, unsigned int value2);`
- `void OpenOC3(unsigned int config, unsigned int value1, unsigned int value2);`
- `void OpenOC4(unsigned int config, unsigned int value1, unsigned int value2);`
- `void OpenOC5(unsigned int config, unsigned int value1, unsigned int value2);`
- `void OpenOC6(unsigned int config, unsigned int value1, unsigned int value2);`
- `void OpenOC7(unsigned int config, unsigned int value1, unsigned int value2);`
- `void OpenOC8(unsigned int config, unsigned int value1, unsigned int value2);`

### Arguments:
- `config` This contains the parameters to be configured in the OCxCON register as defined below:
  - **Idle mode operation**
    - OC_IDLE_STOP
    - OC_IDLE_CON
  - **Clock select**
    - OC_TIMER2_SRC
    - OC_TIMER3_SRC
  - **Output Compare modes of operation**
    - OC_PWM_FAULT_PIN_ENABLE
    - OC_PWM_FAULT_PIN_DISABLE
    - OC_CONTINUE_PULSE
    - OC_SINGLE_PULSE
    - OC_TOGGLE_PULSE
    - OC_HIGH_LOW
    - OC_LOW_HIGH
    - OC_OFF
- `value1` This contains the value to be stored into OCxRS Secondary Register.
- `value2` This contains the value to be stored into OCxR Main Register.

### Return Value:
None
Remarks: This function configures the Output Compare Module Control register (OCxCON) with the following parameters: Clock select, mode of operation, operation in Idle mode. It also configures the OCxRS and OCxR registers.

Source File: OpenOC1.c
OpenOC2.c
OpenOC3.c
OpenOC4.c
OpenOC5.c
OpenOC6.c
OpenOC7.c
OpenOC8.c

Code Example: OpenOC1(OC_IDLE_CON & OC_TIMER2_SRC & OC_PWM_FAULT_PIN_ENABLE, 0x80, 0x60);
ReadDCOC1PWM
ReadDCOC2PWM
ReadDCOC3PWM
ReadDCOC4PWM
ReadDCOC5PWM
ReadDCOC6PWM
ReadDCOC7PWM
ReadDCOC8PWM

## Description:
This function reads the duty cycle from the Output Compare Secondary register.

## Include:
outcompare.h

## Prototype:
unsigned int ReadDCOC1PWM(void);
unsigned int ReadDCOC2PWM(void);
unsigned int ReadDCOC3PWM(void);
unsigned int ReadDCOC4PWM(void);
unsigned int ReadDCOC5PWM(void);
unsigned int ReadDCOC6PWM(void);
unsigned int ReadDCOC7PWM(void);
unsigned int ReadDCOC8PWM(void);

## Arguments:
None

## Return Value:
This function returns the content of OCxRS register when Output Compare module is in PWM mode. Else ‘-1’ is returned

## Remarks:
This function reads the duty cycle from the Output Compare Secondary register (OCxRS) when Output Compare module is in PWM mode. If not in PWM mode, the functions returns a value of ‘-1’.

## Source File:
ReadDCOC1PWM.c
ReadDCOC2PWM.c
ReadDCOC3PWM.c
ReadDCOC4PWM.c
ReadDCOC5PWM.c
ReadDCOC6PWM.c
ReadDCOC7PWM.c
ReadDCOC8PWM.c

## Code Example:
unsigned int compare_reg;
compare_reg = ReadDCOC1PWM();
ReadRegOC1
ReadRegOC2
ReadRegOC3
ReadRegOC4
ReadRegOC5
ReadRegOC6
ReadRegOC7
ReadRegOC8

Description: This function reads the duty cycle registers when Output Compare module is not in PWM mode.

Include: outcompare.h

Prototype:
unsigned int ReadRegOC1(char reg);
unsigned int ReadRegOC2(char reg);
unsigned int ReadRegOC3(char reg);
unsigned int ReadRegOC4(char reg);
unsigned int ReadRegOC5(char reg);
unsigned int ReadRegOC6(char reg);
unsigned int ReadRegOC7(char reg);
unsigned int ReadRegOC8(char reg);

Arguments: reg This indicates if the read should happen from the main or secondary duty cycle registers of Output Compare module.
If reg is '1', then the contents of Main Duty Cycle register (OCxR) is read.
If reg is '0', then the contents of Secondary Duty Cycle register (OCxRS) is read.

Return Value: If reg is '1', then the contents of Main Duty Cycle register (OCxR) is read.
If reg is '0', then the contents of Secondary Duty Cycle register (OCxRS) is read.
If Output Compare module is in PWM mode, '-1' is returned.

Remarks: The read of Duty Cycle register happens only when Output Compare module is not in PWM mode. Else, a value of '-1' is returned.

Source File:
ReadRegOC1.c
ReadRegOC2.c
ReadRegOC3.c
ReadRegOC4.c
ReadRegOC5.c
ReadRegOC6.c
ReadRegOC7.c
ReadRegOC8.c

Code Example:
unsigned int dutycycle_reg;
dutycycle_reg = ReadRegOC1(1);
SetDCOC1PWM
SetDCOC2PWM
SetDCOC3PWM
SetDCOC4PWM
SetDCOC5PWM
SetDCOC6PWM
SetDCOC7PWM
SetDCOC8PWM

<table>
<thead>
<tr>
<th>Description:</th>
<th>This function configures the Output Compare Secondary Duty Cycle register (OCxRS) when the module is in PWM mode.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>outcompare.h</td>
</tr>
<tr>
<td>Prototype:</td>
<td>void SetDCOC1PWM(unsigned int dutycycle);   void SetDCOC2PWM(unsigned int dutycycle);   void SetDCOC3PWM(unsigned int dutycycle);   void SetDCOC4PWM(unsigned int dutycycle);   void SetDCOC5PWM(unsigned int dutycycle);   void SetDCOC6PWM(unsigned int dutycycle);   void SetDCOC7PWM(unsigned int dutycycle);   void SetDCOC8PWM(unsigned int dutycycle);</td>
</tr>
<tr>
<td>Arguments:</td>
<td>dutycycle  This is the duty cycle value to be stored into Output Compare Secondary Duty Cycle register (OCxRS).</td>
</tr>
<tr>
<td>Return Value:</td>
<td>None</td>
</tr>
<tr>
<td>Remarks:</td>
<td>The Output Compare Secondary Duty Cycle register (OCxRS) will be configured with new value only if the module is in PWM mode.</td>
</tr>
<tr>
<td>Source File:</td>
<td>SetDCOC1PWM.c    SetDCOC2PWM.c    SetDCOC3PWM.c    SetDCOC4PWM.c    SetDCOC5PWM.c    SetDCOC6PWM.c    SetDCOC7PWM.c    SetDCOC8PWM.c</td>
</tr>
<tr>
<td>Code Example:</td>
<td>SetDCOC1PWM(dutycycle);</td>
</tr>
</tbody>
</table>
SetPulseOC1
SetPulseOC2
SetPulseOC3
SetPulseOC4
SetPulseOC5
SetPulseOC6
SetPulseOC7
SetPulseOC8

Description: This function configures the Output Compare main and secondary registers (OCxR and OCxRS) when the module is not in PWM mode.

Include: outcompare.h

Prototype:
void SetPulseOC1(unsigned int pulse_start, unsigned int pulse_stop);
void SetPulseOC2(unsigned int pulse_start, unsigned int pulse_stop);
void SetPulseOC3(unsigned int pulse_start, unsigned int pulse_stop);
void SetPulseOC4(unsigned int pulse_start, unsigned int pulse_stop);
void SetPulseOC5(unsigned int pulse_start, unsigned int pulse_stop);
void SetPulseOC6(unsigned int pulse_start, unsigned int pulse_stop);
void SetPulseOC7(unsigned int pulse_start, unsigned int pulse_stop);
void SetPulseOC8(unsigned int pulse_start, unsigned int pulse_stop);

Arguments:
pulse_start This is the value to be stored into Output Compare Main register (OCxR).
pulse_stop This is the value to be stored into Output Compare Secondary register (OCxRS).

Return Value: None

Remarks: The Output Compare duty cycle registers (OCxR and OCxRS) will be configured with new values only if the module is not in PWM mode.

Source File:
SetPulseOC1.c
SetPulseOC2.c
SetPulseOC3.c
SetPulseOC4.c
SetPulseOC5.c
SetPulseOC6.c
SetPulseOC7.c
SetPulseOC8.c

Code Example:
pulse_start = 0x40;
pulse_stop = 0x60;
SetPulseOC1(pulse_start, pulse_stop);
### 3.11.2 Individual Macros

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
<th>Include</th>
<th>Arguments</th>
<th>Remarks</th>
<th>Code Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnableIntOC1</td>
<td>This macro enables the interrupt on output compare match.</td>
<td><code>outcompare.h</code></td>
<td>None</td>
<td>This macro sets Output Compare (OC) Interrupt Enable bit of Interrupt Enable Control register.</td>
<td><code>EnableIntOC8;</code></td>
</tr>
<tr>
<td>EnableIntOC2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnableIntOC3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnableIntOC4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnableIntOC5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnableIntOC6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnableIntOC7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnableIntOC8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
<th>Include</th>
<th>Arguments</th>
<th>Remarks</th>
<th>Code Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisableIntOC1</td>
<td>This macro disables the interrupt on compare match.</td>
<td><code>outcompare.h</code></td>
<td>None</td>
<td>This macro clears OC Interrupt Enable bit of Interrupt Enable Control register.</td>
<td><code>DisableIntOC7;</code></td>
</tr>
<tr>
<td>DisableIntOC2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DisableIntOC3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DisableIntOC4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DisableIntOC5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DisableIntOC6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DisableIntOC7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DisableIntOC8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
<th>Include</th>
<th>Arguments</th>
<th>Remarks</th>
<th>Code Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetPriorityIntIC1</td>
<td>This macro sets priority for output compare interrupt.</td>
<td><code>outcompare.h</code></td>
<td><code>priority</code></td>
<td>This macro sets OC Interrupt Priority bits of Interrupt Priority Control register.</td>
<td><code>SetPriorityIntOC4(0);</code></td>
</tr>
<tr>
<td>SetPriorityIntIC2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SetPriorityIntIC3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SetPriorityIntIC4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SetPriorityIntIC5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SetPriorityIntIC6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SetPriorityIntIC7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SetPriorityIntIC8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.11.3 Example of Use

#define __dsPIC30F6014__
#include<p30fxxxx.h>
#include<outcompare.h>
/* This is ISR corresponding to OC1 interrupt */
void __attribute__((__interrupt__)) _OC1Interrupt(void)
{
    IFS0bits.OC1IF = 0;
}
int main(void)
{
    /* Holds the value at which OCx Pin to be driven high */
    unsigned int pulse_start ;
    /* Holds the value at which OCx Pin to be driven low */
    unsigned int pulse_stop;
    /* Turn off OC1 module */
    CloseOC1();
    /* Configure output compare1 interrupt */
    ConfigIntOC1(OC_INT_OFF & OC_INT_PRIOR_5);
    /* Configure OC1 module for required pulse width */
    pulse_start = 0x40;
    pulse_stop = 0x60;
    PR3 = 0x80 ;
    PR1 = 0xffff;
    TMR1 = 0x0000;
    T3CON = 0x8000;
    T1CON = 0x8000;
    /* Configure Output Compare module to 'initialise OCx pin
    low and generate continuous pulse'mode */
    OpenOC1(OC_IDLE_CON & OC_TIMER3_SRC &
    OC_CONTINUE_PULSE,
    pulse_stop, pulse_start);
    /* Generate continuous pulse till TMR1 reaches 0xff00 */
    while(TMR1<= 0xff00);
    asm("nop");
    CloseOC1();
    return 0;
}
3.12 UART FUNCTIONS

This section contains a list of individual functions for UART module and an example of use of the functions. Functions may be implemented as macros.

3.12.1 Individual Functions

**BusyUART1**

**BusyUART2**

| Description: | This function returns the UART transmission status. |
| Include: | uart.h |
| Prototype: | char BusyUART1(void); char BusyUART2(void); |
| Arguments: | None |
| Return Value: | If '1' is returned, it indicates that UART is busy in transmission and UxSTA<TRMT> bit is '0'. If '0' is returned, it indicates that UART is not busy and UxSTA<TRMT> bit is '1'. |
| Remarks: | This function returns the status of the UART. This indicates if the UART is busy in transmission as indicated by the UxSTA<TRMT> bit. |
| Source File: | BusyUART1.c BusyUART2.c |
| Code Example: | while(BusyUART1()); |

**CloseUART1**

**CloseUART2**

| Description: | This function turns off the UART module |
| Include: | uart.h |
| Prototype: | void CloseUART1(void); void CloseUART2(void); |
| Arguments: | None |
| Return Value: | None |
| Remarks: | This function first turns off the UART module and then disables the UART transmit and receive interrupts. The Interrupt Flag bits are also cleared. |
| Source File: | CloseUART1.c CloseUART2.c |
| Code Example: | CloseUART1(); |
ConfigIntUART1
ConfigIntUART2

Description: This function configures the UART Interrupts.
Include: uart.h
Prototype: void ConfigIntUART1(unsigned int config);
void ConfigIntUART2(unsigned int config);
Arguments: config Individual interrupt enable/disable information as defined below:

Receive Interrupt enable
UART_RX_INT_EN
UART_RX_INT_DIS

Receive Interrupt Priority
UART_RX_INT_PR0
UART_RX_INT_PR1
UART_RX_INT_PR2
UART_RX_INT_PR3
UART_RX_INT_PR4
UART_RX_INT_PR5
UART_RX_INT_PR6
UART_RX_INT_PR7

Transmit Interrupt enable
UART_TX_INT_EN
UART_TX_INT_DIS

Transmit Interrupt Priority
UART_TX_INT_PR0
UART_TX_INT_PR1
UART_TX_INT_PR2
UART_TX_INT_PR3
UART_TX_INT_PR4
UART_TX_INT_PR5
UART_TX_INT_PR6
UART_TX_INT_PR7

Return Value: None
Remarks: This function enables/disables the UART transmit and receive interrupts and sets the interrupt priorities.
Source File: ConfigIntUART1.c
ConfigIntUART2.c
Code Example: ConfigIntUART1(UART_RX_INT_EN & UART_RX_INT_PR5 & UART_TX_INT_EN & UART_TX_INT_PR3);
## DataRdyUART1

**Description:**
This function returns the UART receive buffer status.

**Include:**
`uart.h`

**Prototype:**
```c
char DataRdyUART1(void);
char DataRdyUART2(void);
```

**Arguments:**
`None`

**Return Value:**
If `1` is returned, it indicates that the receive buffer has a data to be read.
If `0` is returned, it indicates that receive buffer does not have any new data to be read.

**Remarks:**
This function returns the status of the UART receive buffer.
This indicates if the UART receive buffer contains any new data that is yet to be read as indicated by the UxSTA<URXDA> bit.

**Source File:**
- `DataRdyUART1.c`
- `DataRdyUART2.c`

**Code Example:**
```c
while(DataRdyUART1());
```

## OpenUART1

**Description:**
This function configures the UART module

**Include:**
`uart.h`

**Prototype:**
```c
void OpenUART1(unsigned int config1,
               unsigned int config2,
               unsigned int ubrg);
void OpenUART2(unsigned int config1,
               unsigned int config2,
               unsigned int ubrg);
```

**Arguments:**
`config1`
This contains the parameters to be configured in the UxMODE register as defined below:
- **UART enable/disable**
  - `UART_EN`
  - `UART_DIS`
- **UART Idle mode operation**
  - `UART_IDLE_CON`
  - `UART_IDLE_STOP`
- **UART communication with ALT pins**
  - `UART_ALTRX_ALT TX`
  - `UART_RX_TX`
  - UART communication with ALT pins is available only for certain devices and the suitable data sheet should be referred to.
- **UART Wake-up on Start**
  - `UART_EN_WAKE`
  - `UART_DIS_WAKE`
- **UART Loopback mode enable/disable**
  - `UART_EN_LOOPBACK`
  - `UART_DIS_LOOPBACK`
- **Input to Capture module**
  - `UART_EN_ABAUD`
  - `UART_DIS_ABAUD`
OpenUART1 (Continued)
OpenUART2

<table>
<thead>
<tr>
<th>Parity and data bits select</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART_NO_PAR_9BIT</td>
</tr>
<tr>
<td>UART_ODD_PAR_8BIT</td>
</tr>
<tr>
<td>UART_EVEN_PAR_8BIT</td>
</tr>
<tr>
<td>UART_NO_PAR_8BIT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of Stop bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART_2STOPBITS</td>
</tr>
<tr>
<td>UART_1STOPBIT</td>
</tr>
</tbody>
</table>

config2 This contains the parameters to be configured in the UxSTA register as defined below:

<table>
<thead>
<tr>
<th>UART Transmission mode interrupt select</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART_INT_TX_BUF_EMPTY</td>
</tr>
<tr>
<td>UART_INT_TX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UART Transmit Break bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART_TX_PIN_NORMAL</td>
</tr>
<tr>
<td>UART_TX_PIN_LOW</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UART transmit enable/disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART_TX_ENABLE</td>
</tr>
<tr>
<td>UART_TX_DISABLE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UART Receive Interrupt mode select</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART_INT_RX_BUF_FUL</td>
</tr>
<tr>
<td>UART_INT_RX_3_4_FUL</td>
</tr>
<tr>
<td>UART_INT_RX_CHAR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UART address detect enable/disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART_ADR_DETECT_EN</td>
</tr>
<tr>
<td>UART_ADR_DETECT_DIS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UART OVERRUN bit clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART_RX_OVERRUN_CLEAR</td>
</tr>
</tbody>
</table>

ubrg This is the value to be written into UxBRG register to set the baud rate.

Return Value: None

Remarks: This functions configures the UART transmit and receive sections and sets the communication baud rate.

Source File: OpenUART1.c
OpenUART2.c

Code Example:

```c
baud = 5;
UMODEvalue = UART_EN & UART_IDLE_CON &
             UART_DIS_WAKE & UART_EN_LOOPBACK &
             UART_EN_ABAUD & UART_NO_PAR_8BIT &
             UART_1STOPBIT;

U1STAvalue = UART_INT_TX_BUF_EMPTY &
             UART_TX_PIN_NORMAL &
             UART_TX_ENABLE &
             UART_INT_RX_3_4_FUL &
             UART_ADR_DETECT_DIS &
             UART_RX_OVERRUN_CLEAR;
OpenUART1(UMODEvalue, U1STAvalue, baud);
```
### ReadUART1

**Description:** This function returns the content of UART receive buffer (UxRXREG) register.

**Include:**

```c
#include "uart.h"
```

**Prototype:**

```c
unsigned int ReadUART1(void);
unsigned int ReadUART2(void);
```

**Arguments:** None

**Return Value:** This function returns the contents of Receive buffer (UxRXREG) register.

**Remarks:** This function returns the contents of the Receive Buffer register.
- If 9 bit reception is enabled, the entire register content is returned.
- If 8 bit reception is enabled, then register is read and the 9th bit is masked.

**Source File:**

```c
ReadUART1.c
ReadUART2.c
```

**Code Example:**

```c
unsigned int RX_data;
RX_data = ReadUART1();
```

### WriteUART1

**Description:** This function writes data to be transmitted into the transmit buffer (UxTXREG) register.

**Include:**

```c
#include "uart.h"
```

**Prototype:**

```c
void WriteUART1(unsigned int data);
void WriteUART2(unsigned int data);
```

**Arguments:**

- `data` This is the data to be transmitted.

**Return Value:** None

**Remarks:** This function writes the data to be transmitted into the transmit buffer.
- If 9-bit transmission is enabled, the 9-bit value is written into the transmit buffer.
- If 8-bit transmission is enabled, then upper byte is masked and then written into the transmit buffer.

**Source File:**

```c
WriteUART1.c
WriteUART2.c
```

**Code Example:**

```c
WriteUART1(0xFF);
```
### getsUART1
### getsUART2

**Description:**
This function reads a string of data of specified length and stores it into the buffer location specified.

**Include:**
uart.h

**Prototype:**
```c
unsigned int getsUART1(unsigned int length,
            unsigned int *buffer, unsigned int
            uart_data_wait);
unsigned int getsUART2(unsigned int length,
            unsigned int *buffer, unsigned int
            uart_data_wait);
```

**Arguments:**
- `length` This is the length of the string to be received.
- `buffer` This is the pointer to the location where the data received have to be stored.
- `uart_data_wait` This is the time-out count for which the module has to wait before return. If the time-out count is 'N', the actual time out would be about \((19 \times N – 1)\) instruction cycles.

**Return Value:**
This function returns the number of bytes yet to be received.
If the return value is '0', it indicates that the complete string has been received.
If the return value is non-zero, it indicates that the complete string has not been received.

**Remarks:**
None

**Source File:**
- getsUART1.c
- getsUART2.c

**Code Example:**
```c
Datarem = getsUART1(6, Rxdata_loc, 40);
```

### putsUART1
### putsUART2

**Description:**
This function writes a string of data to be transmitted into the UART transmit buffer.

**Include:**
uart.h

**Prototype:**
```c
void putsUART1(unsigned int *buffer);
void putsUART2(unsigned int *buffer);
```

**Arguments:**
- `buffer` This is the pointer to the string of data to be transmitted.

**Return Value:**
None

**Remarks:**
This function writes the data to be transmitted into the transmit buffer until NULL character is encountered. Once the transmit buffer is full, it waits until data gets transmitted and then writes the next data into the Transmit register.

**Source File:**
- putsUART1.c
- putsUART2.c

**Code Example:**
```c
putsUART1(Txdata_loc);
```
### 3.12.2 Individual Macros

#### EnableIntU1RX

<table>
<thead>
<tr>
<th>Description:</th>
<th>This macro enables the UART receive interrupt.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>uart.h</td>
</tr>
<tr>
<td>Arguments:</td>
<td>None</td>
</tr>
<tr>
<td>Remarks:</td>
<td>This macro sets UART Receive Interrupt Enable bit of Interrupt Enable Control register.</td>
</tr>
<tr>
<td>Code Example:</td>
<td>EnableIntU2RX;</td>
</tr>
</tbody>
</table>

#### EnableIntU1TX

<table>
<thead>
<tr>
<th>Description:</th>
<th>This macro enables the UART transmit interrupt.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>uart.h</td>
</tr>
<tr>
<td>Arguments:</td>
<td>None</td>
</tr>
<tr>
<td>Remarks:</td>
<td>This macro sets UART Transmit Interrupt Enable bit of Interrupt Enable Control register.</td>
</tr>
<tr>
<td>Code Example:</td>
<td>EnableIntU2TX;</td>
</tr>
</tbody>
</table>

#### DisableIntU1RX

<table>
<thead>
<tr>
<th>Description:</th>
<th>This macro disables the UART receive interrupt.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>uart.h</td>
</tr>
<tr>
<td>Arguments:</td>
<td>None</td>
</tr>
<tr>
<td>Remarks:</td>
<td>This macro clears UART Receive Interrupt Enable Control register.</td>
</tr>
<tr>
<td>Code Example:</td>
<td>DisableIntU1RX;</td>
</tr>
</tbody>
</table>
DisableIntU1TX
DisableIntU2TX

Description: This macro disables the UART transmit interrupt.
Include: uart.h
Arguments: None
Remarks: This macro clears UART Transmit Interrupt Enable bit of Interrupt Enable Control register.
Code Example: DisableIntU1TX;

SetPriorityIntU1RX
SetPriorityIntU2RX

Description: This macro sets priority for UART receive interrupt.
Include: uart.h
Arguments: priority
Remarks: This macro sets UART Receive Interrupt Priority bits of Interrupt Priority Control register.
Code Example: SetPriorityIntU1RX(6);

SetPriorityIntU1TX
SetPriorityIntU2TX

Description: This macro sets priority for UART transmit interrupt.
Include: uart.h
Arguments: priority
Remarks: This macro sets UART Transmit Interrupt Priority bits of Interrupt Priority Control register.
Code Example: SetPriorityIntU1TX(5);
3.12.3 Example of Use

#define __dsPIC30F6014__
#include<p30fxxxx.h>
#include<uart.h>

/* Received data is stored in array Buf */
char Buf[80];
char *Receiveddata = Buf;

/* This is UART1 transmit ISR */
void __attribute__((__interrupt__)) _U1TXInterrupt(void)
{
    IFS0bits.U1TXIF = 0;
}

/* This is UART1 receive ISR */
void __attribute__((__interrupt__)) _U1RXInterrupt(void)
{
    IFS0bits.U1RXIF = 0;
    /* Read the receive buffer till atleast one or more character can be
    read */
    while( DataRdyUART1())
    {
        *(Receiveddata)++ = ReadUART1();
    }
}

int main(void)
{
/* Data to be transmitted using UART communication module */
char Txdata[] = {'M','i','c','r','o','c','h','i','p','I','C','D2',

    '0'};
/* Holds the value of baud register */
unsigned int baudvalue;
/* Holds the value of uart config reg */
unsigned int U1MODEvalue;
/* Holds the information regarding uart TX & RX interrupt modes */
unsigned int U1STAvalue;
/* Turn off UART1 module */
CloseUART1();

/* Configure uart1 receive and transmit interrupt */
ConfigIntUART1(UART_RX_INT_EN & UART_RX_INT_PR6 &
    UART_TX_INT_DIS & UART_TX_INT_PR2);
/* Configure UART1 module to transmit 8 bit data with one stopbit.
Also Enable loopback mode */
baudvalue = 5;
U1MODEvalue = UART_EN & UART_IDLE_CON &
    UART_DIS_WAKE & UART_ENLOOPBACK &
    UART_EN_ABAUD & UART_NO_PAR_8BIT &
    UART_1STOPBIT;
U1STAvalue = UART_INT_TX_BUF_EMPTY &
    UART_TX_PIN_NORMAL &
    UART_TX_ENABLE & UART_INT_RX_3_4_FUL &
    UART_ADR_DETECT_DIS &
    UART_RX_OVERRUN_CLEAR;
OpenUART1(U1MODEvalue, U1STAvalue, baudvalue);
/ * Load transmit buffer and transmit the same till null character is encountered */
    putsUART1 ((unsigned int *)Txdata);
/* Wait for transmission to complete */
    while(BusyUART1());
/* Read all the data remaining in receive buffer which are unread */
    while(DataRdyUART1())
    {
        (*(Receiveddata)++) = ReadUART1();
    }
/* Turn off UART1 module */
    CloseUART1();
    return 0;
}

3.13 DCI FUNCTIONS

This section contains a list of individual functions for DCI module and an example of use of the functions. Functions may be implemented as macros.

3.13.1 Individual Functions

**CloseDCI**

| Description: | This function turns off the DCI module |
| Include:     | dci.h                     |
| Prototype:   | void CloseDCI(void);     |
| Arguments:   | None                     |
| Return Value:| None                     |
| Remarks:     | This function first turns off the DCI module and then disables the DCI interrupt. The Interrupt Flag bit is also cleared. |
| Source File: | CloseDCI.c               |
| Code Example:| CloseDCI();              |

**BufferEmptyDCI**

| Description: | This function returns the DCI Transmit Buffer Full status. |
| Include:     | dci.h                     |
| Prototype:   | char BufferEmptyDCI(void); |
| Arguments:   | None                     |
| Return Value:| If the value of TMPTY is '1', then '1' is returned, indicating that the transmit buffer is empty.
If the value of TMPTY is '0', then '0' is returned, indicating that the transmit buffer is not empty. |
| Remarks:     | This function returns the status of the DCISTAT<TMPTY> bit. This bit indicates whether the transmit buffer is empty. |
| Source File: | BufferEmptyDCI.c          |
| Code Example:| while(!BufferEmptyDCI()); |
**ConfigIntDCI**

Description: This function configures the DCI interrupt.

Include: `dci.h`

Prototype: `void ConfigIntDCI(unsigned int config);`

Arguments: `config` DCI interrupt priority and enable/disable information as defined below:

- **DCI Interrupt enable/disable**
  - DCI_INT_ON
  - DCI_INT_OFF

- **DCI Interrupt priority**
  - DCI_INT_PRI_0
  - DCI_INT_PRI_1
  - DCI_INT_PRI_2
  - DCI_INT_PRI_3
  - DCI_INT_PRI_4
  - DCI_INT_PRI_5
  - DCI_INT_PRI_6
  - DCI_INT_PRI_7

Return Value: None

Remarks: This function clears the Interrupt Flag (DCIIF) bit and then sets the interrupt priority and enables/disables the interrupt.

Source File: `ConfigIntDCI.c`

Code Example:
```
ConfigIntDCI(DCI_INT_PRI_6 & DCI_INT_ENABLE);
```

**DataRdyDCI**

Description: This function returns the status of DCI receive buffers.

Include: `dci.h`

Prototype: `char DataRdyDCI(void);`

Arguments: None

Return Value: If the value of RFUL is '1', then '1' is returned, indicating that the data is ready to be read from the receive buffers. If the value of RFUL is '0', then '0' is returned, indicating that the receive buffers are empty.

Remarks: This function returns the status of the DCISTAT<RFUL> bit. This bit indicates whether the data is available in the receive buffers.

Source File: `DataRdyDCI.c`

Code Example:
```
while(!DataRdyDCI());
```

**OpenDCI**

Description: This function configures the DCI.

Include: `dci.h`

Prototype: `void OpenDCI(unsigned int config1, unsigned int config2, unsigned int config3, unsigned int trans_mask, unsigned int recv_mask)`

Arguments: `config1`

This contains the parameters to be configured in the DCION1 register as defined below:
OpenDCI (Continued)

<table>
<thead>
<tr>
<th>Module On/Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCI_EN</td>
</tr>
<tr>
<td>DCI_DIS</td>
</tr>
</tbody>
</table>

**Idle mode operation**

<table>
<thead>
<tr>
<th>DCI_IDLE_CON</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCI_IDLE_STOP</td>
</tr>
</tbody>
</table>

**DCI Loopback mode enable**

<table>
<thead>
<tr>
<th>DCI_DIGI_LPBACK_EN</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCI_DIGI_LPBACK_DIS</td>
</tr>
</tbody>
</table>

**CSCK pin direction select**

<table>
<thead>
<tr>
<th>DCI_SCKD_INP</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCI_SCKD_OUP</td>
</tr>
</tbody>
</table>

**DCI sampling edge selection**

<table>
<thead>
<tr>
<th>DCI_SAMP_CLK_RIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCI_SAMP_CLK_FAL</td>
</tr>
</tbody>
</table>

**FS pin direction select**

<table>
<thead>
<tr>
<th>DCI_FSD_INP</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCI_FSD_OUP</td>
</tr>
</tbody>
</table>

**data to be transmitted during underflow**

<table>
<thead>
<tr>
<th>DCI_TX_LASTVAL_UNF</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCI_TX_ZERO_UNF</td>
</tr>
</tbody>
</table>

**SDO pin status during transmit disable**

<table>
<thead>
<tr>
<th>DCI_SDO_TRISTAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCI_SDO_ZERO</td>
</tr>
</tbody>
</table>

**Data justification control**

<table>
<thead>
<tr>
<th>DCI_DJST_ON</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCI_DJST_OFF</td>
</tr>
</tbody>
</table>

**Frame Sync mode select**

<table>
<thead>
<tr>
<th>DCI_FSM_ACLK_20BIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCI_FSM_ACLK_16BIT</td>
</tr>
<tr>
<td>DCI_FSM_I2S</td>
</tr>
<tr>
<td>DCI_FSM_MULTI</td>
</tr>
</tbody>
</table>

This contains the parameters to be configured in the DCICON2 register as defined below:

**Buffer length**

<table>
<thead>
<tr>
<th>DCI_BUFF_LEN_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCI_BUFF_LEN_3</td>
</tr>
<tr>
<td>DCI_BUFF_LEN_2</td>
</tr>
<tr>
<td>DCI_BUFF_LEN_1</td>
</tr>
</tbody>
</table>

**DCI Frame sync generator control**

<table>
<thead>
<tr>
<th>DCI_FRAME_LEN_16</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCI_FRAME_LEN_15</td>
</tr>
<tr>
<td>DCI_FRAME_LEN_14</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>DCI_FRAME_LEN_1</td>
</tr>
</tbody>
</table>

**DCI data word size**

<table>
<thead>
<tr>
<th>DCI_DATA_WORD_16</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCI_DATA_WORD_15</td>
</tr>
<tr>
<td>DCI_DATA_WORD_14</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>DCI_DATA_WORD_5</td>
</tr>
<tr>
<td>DCI_DATA_WORD_4</td>
</tr>
</tbody>
</table>
config3  
This contains the bit clock generator value to be configured in the DCICON3 register.

trans_mask/recv_mask  
This contains the transmit/receive slot enable bits to be configured into the TSCON/RSCON register as defined below:

- DCI_DIS_SLOT_15
- DCI_DIS_SLOT_14
- ....
- DCI_DIS_SLOT_1
- DCI_DIS_SLOT_0
- DCI_EN_SLOT_ALL
- DCI_DIS_SLOT_ALL

Return Value: None

Remarks: This routine configures the following parameters:

1. DCICON1 register:
   - Enable bit
   - Frame Sync mode
   - Data Justification
   - Sample Clock Direction
   - Sample Clock
   - Edge Control
   - Output Frame Synchronization Directions Control
   - Continuous Transmit/Receive mode
   - Underflow mode.

2. DCICON2 register:
   - Frame Sync Generator Control
   - Data Word Size bits
   - Buffer Length Control bits

3. DCICON3 register: Clock Generator Control bits

4. TSCON register: Transmit Time Slot Enable Control bits

5. RSCON register: Receive Time Slot Enable Control bits

Source File: OpenDCI.c

Code Example:

```c
DCICON1value = DCI_EN &
   DCI_IDLE_CON &
   DCI_DIGI_LPBACK_EN &
   DCI_SCKD_OUP &
   DCI_SAMP_CLK_FAL &
   DCI_FSD_OUP &
   DCI_TX_LASTVAL_UNF &
   DCI_SDO_TRISTAT &
   DCI_DJST_OFF &
   DCI_FSM_ACLINK_16BIT ;

DCICON2value = DCI_BUFF_LEN_4 &
   DCI_FRAME_LEN_2 &
   DCI_DATA_WORD_16 ;

DCICON3value = 0x02 ;

RSCONvalue = DCI_EN_SLOT_ALL &
   DCI_DIS_SLOT_15 &
   DCI_DIS_SLOT_9 &
   DCI_DIS_SLOT_2 ;

TSCONvalue = DCI_EN_SLOT_ALL &
   DCI_DIS_SLOT_14 &
   DCI_DIS_SLOT_8 &
   DCI_DIS_SLOT_1 ;

OpenDCI(DCICON1value, DCICON2value, DCICON3value,
   TSCONvalue, RSCONvalue);
```

OpenDCI (Continued)
### ReadDCI

**Description:** This function reads the contents of DCI receive buffer.

**Include:** `dci.h`

**Prototype:**
```
unsigned int ReadDCI(unsigned char buffer);
```

**Arguments:**
- `buffer` This is the DCI buffer number to be read.

**Return Value:** None

**Remarks:** This function returns the contents of DCI receive buffer pointed by the `buffer`.

**Source File:** `ReadDCI.c`

**Code Example:**
```
unsigned int DCI_buf0;
DCI_buf0 = ReadDCI(0);
```

### WriteDCI

**Description:** This function writes the data to be transmitted to the DCI transmit buffer.

**Include:** `dci.h`

**Prototype:**
```
void WriteDCI(unsigned int data_out, unsigned char buffer);
```

**Arguments:**
- `data_out` This is the data to be transmitted.
- `buffer` This is the DCI buffer number to be written.

**Return Value:** None

**Remarks:** This function loads the transmit buffer specified by the `buffer` with `data_out`.

**Source File:** `WriteDCI.c`

**Code Example:**
```
unsigned int DCI_tx0 = 0x60;
WriteDCI(DCI_tx0, 0);
```
3.13.2 Individual Macros

EnableIntDCI

| Description: | This macro enables the DCI interrupt. |
| Include:     | dci.h |
| Arguments:   | None |
| Remarks:     | This macro sets DCI Interrupt Enable bit of Interrupt Enable Control register. |
| Code Example:| EnableIntDCI; |

DisableIntDCI

| Description: | This macro disables the DCI interrupt. |
| Include:     | dci.h |
| Arguments:   | None |
| Remarks:     | This macro clears DCI Interrupt Enable bit of Interrupt Enable Control register. |
| Code Example:| DisableIntDCI; |

SetPriorityIntDCI

| Description: | This macro sets priority for DCI interrupt. |
| Include:     | dci.h |
| Arguments:   | priority |
| Remarks:     | This macro sets DCI Interrupt Priority bits of Interrupt Priority Control register. |
| Code Example:| SetPriorityIntDCI(4); |
3.13.3 Example of Use

```c
#define __dsPIC30F6014__
#include<p30fxxxx.h>
#include<dci.h>

/* Received data is stored from 0x1820 onwards. */
unsigned int * Receiveddata = ( unsigned int *)0x1820;

void __attribute__((__interrupt__)) _DCIInterrupt(void)
{
  IFS2bits.DCIIF = 0;
}

int main(void)
{
  /* Data to be transmitted using DCI module */
  unsigned int data16[] = {0xabcd, 0x1234, 0x1578,
                           0xfff0, 0xf679};
  /* Holds configuration information */
  unsigned int DCICON1value;
  /* Holds the value of framelen, wordsize and buffer length */
  unsigned int DCICON2value;
  /* Holds the information regarding bit clock generator */
  unsigned int DCICON3value;
  /* Holds the information regarding data to be received or ignored during this timeslot */
  unsigned int RSCONvalue;
  /* Holds the information regarding transmit buffer contents are sent during the timeslot */
  unsigned int TSCONvalue;
  int i;
  CloseDCI();
  /* Configure DCI receive / transmit interrupt */
  ConfigIntDCI( DCI_INT_ON & DCI_INT_PRI_6);
  /* Configure DCI module to transmit 16 bit data with multichannel mode */
  DCICON1value = DCI_EN & DCI_IDLE_CON &
                 DCI_DIGI_LPBACK_EN &
                 DCI_SCKD_OUP &
                 DCI_SAMP_CLK_PAL &
                 DCI_FSD_OUP &
                 DCI_TX_ZERO_UNF &
                 DCI_SDO_TRISTAT &
                 DCI_DJST_OFF &
                 DCI_FSM_MULTI;
  DCICON2value = DCI_BUFF_LEN_4 & DCI_FRAME_LEN_4 &
                 DCI_DATA_WORD_16 ;
  DCICON3value = 0x00;
  RSCONvalue = DCI_EN_SLOT_ALL & DCI_DIS SLOT_11 &
               DCI_DIS SLOT_4 & DCI_DIS SLOT_5;
  TSCONvalue = DCI_EN_SLOT_ALL & DCI_DIS SLOT_11 &
               DCI_DIS SLOT_4 & DCI_DIS SLOT_5;
  OpenDCI(DCICON1value, DCICON2value, DCICON3value,
          TSCONvalue, RSCONvalue);
```

/* Load transmit buffer and transmit the same */
    i = 0;
    while( i<= 3)
    {
        WriteDCI(data16[i],i);
        i++;
    }
/* Start generating serial clock by DCI module */
    DCICON3 = 0X02;
/* Wait for transmit buffer to get empty */
    while(!BufferEmptyDCI());
/* Wait till new data is available in RX buffer */
    while(!DataRdyDCI());
/* Read all the data remaining in receive buffer which are unread into user defined data buffer*/
    i = 0;
    while( i<=3)
    {
        (*( Receiveddata)++) = ReadDCI(i);
        i++;
    }
/* Turn off DCI module and clear IF bit */
    CloseDCI();
    return 0;
3.14 SPI FUNCTIONS

This section contains a list of individual functions for SPI module and an example of use of the functions. Functions may be implemented as macros.

3.14.1 Individual Functions

**ConfigIntSPI1**

**ConfigIntSPI2**

| Description: | This function configures the SPI Interrupt. |
| Include:     | spi.h                                      |
| Prototype:   | void ConfigIntSPI1(unsigned int config);   |
|             | void ConfigIntSPI2(unsigned int config);   |
| Arguments:   | config SPI interrupt priority and enable/disable information as defined below: |
|             | interrupt enable/disable                  |
|             | SPI_INT_EN                                |
|             | SPI_INT_DIS                               |
|             | Interrupt Priority                        |
|             | SPI_INT_PRI_0                             |
|             | SPI_INT_PRI_1                             |
|             | SPI_INT_PRI_2                             |
|             | SPI_INT_PRI_3                             |
|             | SPI_INT_PRI_4                             |
|             | SPI_INT_PRI_5                             |
|             | SPI_INT_PRI_6                             |
|             | SPI_INT_PRI_7                             |

| Return Value: | None                                      |
| Remarks:      | This function clears the Interrupt Flag bit, sets the interrupt priority and enables/disables the interrupt. |

| Source File:  | ConfigIntSPI1.c                           |
|              | ConfigIntSPI2.c                           |

| Code Example: | ConfigIntSPI1(SPI_INT_PRI_3 & SPI_INT_EN); |

**CloseSPI1**

**CloseSPI2**

| Description: | This function turns off the SPI module |
| Include:     | spi.h                                  |
| Prototype:   | void CloseSPI1(void);                  |
|             | void CloseSPI2(void);                  |

| Arguments:   | None                                    |
| Return Value:| None                                    |

| Remarks:     | This function disables the SPI interrupt and then turns off the module. The Interrupt Flag bit is also cleared. |

| Source File:  | CloseSPI1.c                              |
|              | CloseSPI2.c                              |

| Code Example: | CloseSPI1();                            |
### DataRdySPI1
### DataRdySPI2

**Description:** This function determines if the SPI buffer contains any data to be read.

**Include:**
spi.h

**Prototype:**
char DataRdySPI1(void);
char DataRdySPI2(void);

**Arguments:** None

**Return Value:**
- If ‘1’ is returned, it indicates that the data has been received in the receive buffer and is to be read.
- If ‘0’ is returned, it indicates that the receive is not complete and the receive buffer is empty.

**Remarks:**
This function returns the status of SPI receive buffer. This indicates if the SPI receive buffer contains any new data that is yet to be read as indicated by the SPIxSTAT<SPIRBF> bit. This bit is cleared by hardware when the data is read from the buffer.

**Source File:**
DataRdySPI1.c
DataRdySPI2.c

**Code Example:**
while(DataRdySPI1());

---

### ReadSPI1
### ReadSPI2

**Description:** This function reads the content of the SPI Receive Buffer (SPIxBUF) register.

**Include:**
spi.h

**Prototype:**
unsigned int ReadSPI1(void);
unsigned int ReadSPI2(void);

**Arguments:** None

**Return Value:**
This function returns the content of Receive Buffer (SPIxBUF) register. If a value of ‘-1’ is returned, it indicates that there is no data to be read from the SPI buffer.

**Remarks:**
This function returns the content of the Receive Buffer register. If 16-bit communication is enabled, the data in the SPIxBUF register is returned. If 8-bit communication is enabled, then the lower byte of SPIxBUF is returned. The SPIxBUF is read only if it contains any data as indicated by the SPIxSTAT<RBF> bit. Otherwise, a value of ‘-1’ is returned.

**Source File:**
ReadSPI1.c
ReadSPI2.c

**Code Example:**
unsigned int RX_data;
RX_data = ReadSPI1();
### WriteSPI1

<table>
<thead>
<tr>
<th>Description:</th>
<th>This function writes the data to be transmitted into the Transmit Buffer (SPIxBUF) register.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>spi.h</td>
</tr>
<tr>
<td>Prototype:</td>
<td><code>void WriteSPI1(unsigned int data);</code> <code>void WriteSPI2(unsigned int data);</code></td>
</tr>
<tr>
<td>Arguments:</td>
<td><code>data</code> This is the data to be transmitted which will be stored in SPI buffer.</td>
</tr>
<tr>
<td>Remarks:</td>
<td>This function writes the data (byte/word) to be transmitted into the transmit buffer.</td>
</tr>
<tr>
<td>Return Value:</td>
<td>None</td>
</tr>
<tr>
<td>Source File:</td>
<td>WriteSPI1.c</td>
</tr>
<tr>
<td></td>
<td>WriteSPI2.c</td>
</tr>
<tr>
<td>Code Example:</td>
<td><code>WriteSPI1(0x3FFF);</code></td>
</tr>
</tbody>
</table>

### OpenSPI1

<table>
<thead>
<tr>
<th>Description:</th>
<th>This function configures the SPI module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>spi.h</td>
</tr>
<tr>
<td>Prototype:</td>
<td><code>void OpenSPI1(unsigned int config1, unsigned int config2);</code> <code>void OpenSPI2(unsigned int config1, unsigned int config2);</code></td>
</tr>
<tr>
<td>Arguments:</td>
<td><code>config1</code> This contains the parameters to be configured in the SPIxCON register as defined below:</td>
</tr>
<tr>
<td></td>
<td><strong>Framed SPI support Enable/Disable</strong></td>
</tr>
<tr>
<td></td>
<td>FRAME_ENABLE_ON</td>
</tr>
<tr>
<td></td>
<td>FRAME_ENABLE_OFF</td>
</tr>
<tr>
<td></td>
<td><strong>Frame Sync Pulse direction control</strong></td>
</tr>
<tr>
<td></td>
<td>FRAME_SYNC_INPUT</td>
</tr>
<tr>
<td></td>
<td>FRAME_SYNC_OUTPUT</td>
</tr>
<tr>
<td></td>
<td><strong>SDO Pin Control bit</strong></td>
</tr>
<tr>
<td></td>
<td>DISABLE_SDO_PIN</td>
</tr>
<tr>
<td></td>
<td>ENABLE_SDO_PIN</td>
</tr>
<tr>
<td></td>
<td><strong>Word/Byte Communication mode</strong></td>
</tr>
<tr>
<td></td>
<td>SPI_MODE16_ON</td>
</tr>
<tr>
<td></td>
<td>SPI_MODE16_OFF</td>
</tr>
<tr>
<td></td>
<td><strong>SPI! Data Input Sample phase</strong></td>
</tr>
<tr>
<td></td>
<td>SPI_SMP_ON</td>
</tr>
<tr>
<td></td>
<td>SPI_SMP_OFF</td>
</tr>
<tr>
<td></td>
<td><strong>SPI Clock Edge Select</strong></td>
</tr>
<tr>
<td></td>
<td>SPI_CKE_ON</td>
</tr>
<tr>
<td></td>
<td>SPI_CKE_OFF</td>
</tr>
<tr>
<td></td>
<td><strong>SPI slave select enable</strong></td>
</tr>
<tr>
<td></td>
<td>SLAVE_SELECT_ENABLE_ON</td>
</tr>
<tr>
<td></td>
<td>SLAVE_SELECT_ENABLE_OFF</td>
</tr>
</tbody>
</table>
OpenSPI1 (Continued)

OpenSPI2

SPI Clock polarity select
CLK_POL_ACTIVE_LOW
CLK_POL_ACTIVE_HIGH

SPI Mode Select bit
MASTER_ENABLE_ON
MASTER_ENABLE_OFF

Secondary Prescale select
SEC_PRESCAL_1_1
SEC_PRESCAL_2_1
SEC_PRESCAL_3_1
SEC_PRESCAL_4_1
SEC_PRESCAL_5_1
SEC_PRESCAL_6_1
SEC_PRESCAL_7_1
SEC_PRESCAL_8_1

Primary Prescale select
PRI_PRESCAL_1_1
PRI_PRESCAL_4_1
PRI_PRESCAL_16_1
PRI_PRESCAL_64_1

config2  This contains the parameters to be configured in the
SPIxSTAT register as defined below:

SPI Enable/Disable
SPI_ENABLE
SPI_DISABLE

SPI Idle mode operation
SPI_IDLE_CON
SPI_IDLE_STOP

Clear Receive Overflow Flag bit
SPI_RX_OVFLOW_CLR

Return Value:  None
Remarks:  This functions initializes the SPI module and sets the Idle mode
operation.

Source File:
OpenSPI1.c
OpenSPI2.c

Code Example:
config1 = FRAME_ENABLE_OFF &
FRAME_SYNC_OUTPUT &
ENABLE_SDO_PIN &
SPI_MODE16_ON &
SPI_SMP_ON &
SPI_CKE_OFF &
SLAVE_SELECT_ENABLE_OFF &
CLK_POL_ACTIVE_HIGH &
MASTER_ENABLE_ON &
SEC_PRESCAL_7_1 &
PRI_PRESCAL_64_1;

config2 = SPI_ENABLE &
SPI_IDLE_CON &
SPI_RX_OVFLOW_CLR OpenSPI1(config1, config2);
putsSPI1, putsSPI2

**Description:** This function writes a string of data to be transmitted into the SPI transmit buffer.

**Include:**

spi.h

**Prototype:**

void putsSPI1(unsigned int length,
              unsigned int *wrptr);

void putsSPI2(unsigned int length,
              unsigned int *wrptr);

**Arguments:**

- `length` This is the number of data words/bytes to be transmitted.
- `wrptr` This is the pointer to the string of data to be transmitted.

**Return Value:** None

**Remarks:**

This function writes the specified length of data words/bytes to be transmitted into the transmit buffer. Once the transmit buffer is full, it waits until the data gets transmitted and then writes the next data into the Transmit register. The control remains in this function if SPI module is disabled while SPITBF bit is set.

**Source File:**

putsSPI1.c

putsSPI2.c

**Code Example:**

putsSPI1(10, Txdata_loc);

getsSPI1, getsSPI2

**Description:** This function reads a string of data of specified length and stores it into the location specified.

**Include:**

spi.h

**Prototype:**

unsigned int getsSPI1(
    unsigned int length,
    unsigned int *rdptr,
    unsigned int spi_data_wait);

unsigned int getsSPI2(
    unsigned int length,
    unsigned int *rdptr,
    unsigned int spi_data_wait);

**Arguments:**

- `length` This is the length of the string to be received.
- `rdptr` This is the pointer to the location where the data received have to be stored.
- `spi_data_wait` This is the time-out count for which the module has to wait before return.

If the time-out count is ‘N’, the actual time out would be about (19 * N – 1) instruction cycles.

**Return Value:** This function returns the number of bytes yet to be received. If the return value is a ‘0’, it indicates that the complete string has been received. If the return value is a non-zero, it indicates that the complete string has not been received.

**Remarks:** None

**Source File:**

getsSPI1.c

getsSPI2.c

**Code Example:**

Datarem = getsSPI1(6, Rxdata_loc, 40);
### getcSPI1, getcSPI2

**Description:** This function is identical to ReadSPI1 and ReadSPI2.

**Source File:** #define to ReadSPI1 and ReadSPI2 in spi.h

### putcSPI1, putcSPI2

**Description:** This function is identical to WriteSPI1 and WriteSPI2.

**Source File:** #define to WriteSPI1 and WriteSPI2 in spi.h

### 3.14.2 Individual Macros

#### EnableIntSPI1, EnableIntSPI2

- **Description:** This macro enables the SPI interrupt.
- **Include:** spi.h
- **Arguments:** None
- **Remarks:** This macro sets SPI Interrupt Enable bit of Interrupt Enable Control register.
- **Code Example:** EnableIntSPI1;

#### DisableIntSPI1, DisableIntSPI2

- **Description:** This macro disables the SPI interrupt.
- **Include:** spi.h
- **Arguments:** None
- **Remarks:** This macro clears SPI Interrupt Enable bit of Interrupt Enable Control register.
- **Code Example:** DisableIntSPI2;

#### SetPriorityIntSPI1, SetPriorityIntSPI2

- **Description:** This macro sets priority for SPI interrupt.
- **Include:** spi.h
- **Arguments:** priority
- **Remarks:** This macro sets SPI Interrupt Priority bits of Interrupt Priority Control register.
- **Code Example:** SetPriorityIntSPI2(2);
3.14.3 Example of Use

#define __dsPIC30F6014__
#include<p30fxxxx.h>
#include<spi.h>
/* Data received at SPI2 */
unsigned int datard;
void __attribute__((__interrupt__)) _SPI1Interrupt(void)
{    
    IFS0bits.SPI1IF = 0;
}
void __attribute__((__interrupt__)) _SPI2Interrupt(void)
{    
    IFS1bits.SPI2IF = 0;
    SPI1STATbits SPIROV = 0; /* Clear SPI1 receive overflow flag if set */
}
int main(void)
{
    /* Holds the information about SPI configuartion */
    unsigned int SPICONValue;
    /* Holds the information about SPI Enable/Disable */
    unsigned int SPISTATValue;
    /*Timeout value during which timer1 is ON */
    int timeout;
    /* Turn off SPI modules */
    CloseSPI1();
    CloseSPI2();
    TMR1 = 0 ;
    timeout = 0;
    TRISDbits.TRISD0 = 0;
    /* Configure SPI2 interrupt */
    ConfigIntSPI2(SPI_INT_EN & SPI_INT_PRI_6);
    /* Configure SPI1 module to transmit 16 bit timer1 value in master mode */
    SPICONValue = FRAME_ENABLE_OFF & FRAME_SYNC_OUTPUT &
                  ENABLE_SDO_PIN & SPI_MODE16_ON &
                  SPI_SMP_ON & SPI_CKE_OFF &
                  SLAVE_SELECT_ENABLE_OFF &
                  CLK_POL_ACTIVE_HIGH &
                  MASTER_ENABLE_ON &
                  SEC_PRESCAL_7_1 &
                  PRI_PRESCAL_64_1;
    SPISTATValue = SPI_ENABLE & SPI_IDLE_CON &
                   SPI_RX_OVFLOW_CLR;
    OpenSPI1(SPICONValue, SPISTATValue );

/* Configure SPI2 module to receive 16 bit timer value in slave mode */
SPICONValue = FRAME_ENABLE_OFF & FRAME_SYNC_OUTPUT &
ENABLE_SDO_PIN & SPI_MODE16_ON &
SPI_SMP_OFF & SPI_CKE_OFF &
SLAVE_SELECT_ENABLE_OFF &
CLK_POL_ACTIVE_HIGH &
MASTER_ENABLE_OFF &
SEC_PRESCAL_7_1 &
PRI_PRESCAL_64_1;
SPISTATValue = SPI_ENABLE & SPI_IDLE_CON &
PI_RX_OVFLOW_CLR;
OpenSPI2(SPICONValue, SPISTATValue);
T1CON = 0X8000;
while(timeout< 100 )
{
    timeout = timeout+2 ;
}
T1CON = 0;
WriteSPI1(TMR1);
while(SPI1STATbits.SPITBF);
while(!DataRdySPI2());
datard = ReadSPI2();
if(datard <= 600)
{
    PORTDbits.RD0 = 1;
}

/* Turn off SPI module and clear IF bit */
CloseSPI1();
CloseSPI2();
return 0;
3.15 QEI FUNCTIONS

This section contains a list of individual functions for QEI module and an example of use of the functions. Functions may be implemented as macros.

3.15.1 Individual Functions

CloseQEI

Description: This function turns off the QEI module
Include: qei.h
Prototype: void closeQEI(void);
Arguments: None
Return Value None
Remarks: This function disables the QEI module and clears the QEI Interrupt Enable and Flag bits.
Source File: CloseQEI.c
Code Example: CloseQEI();

ConfigIntQEI

Description: This function Configure the QEI Interrupt.
Include: qei.h
Prototype: void ConfigIntQEI(unsigned int config);
Arguments: config QEI interrupt priority and enable/disable information as defined below:
QEI Interrupt enable/disable
QEI_INT_ENABLE
QEI_INT_DISABLE
QEI Interrupt priority
QEI_INT_PRI_0
QEI_INT_PRI_1
QEI_INT_PRI_2
QEI_INT_PRI_3
QEI_INT_PRI_4
QEI_INT_PRI_5
QEI_INT_PRI_6
QEI_INT_PRI_7
Return Value None
Remarks: This function clears the Interrupt Flag bit, sets the interrupt priority and enables/disables the interrupt.
Source File: ConfigIntQEI.c
Code Example: ConfigIntQEI(QEI_INT_ENABLE & QEI_INT_PRI_1);
OpenQEI

Description: This function configure the QEI.

Include: qei.h

Prototype: void OpenQEI(unsigned int config1, unsigned int config2);

Arguments: config1 This contains the parameters to be configured in the QEIxCON register as defined below:

- Position Counter Direction Selection Control
  - QEI_DIR_SEL_QEB
  - QEI_DIR_SEL_CNTRL

- Timer Clock Source Select bit
  - QEI_EXT_CLK
  - QEI_INT_CLK

- Position Counter Reset Enable
  - QEI_INDEX_RESET_ENABLE
  - QEI_INDEX_RESET_DISABLE

- Timer Input Clock Prescale Select bits
  - QEI_CLK_PRESCALE_1
  - QEI_CLK_PRESCALE_8
  - QEI_CLK_PRESCALE_64
  - QEI_CLK_PRESCALE_256

- Timer Gated Time Accumulation Enable
  - QEI_GATED_ACC_ENABLE
  - QEI_GATED_ACC_DISABLE

- Position Counter Direction State Output Enable
  - QEI_LOGIC_CONTROL_IO
  - QEI_NORMAL_IO

- Phase A and Phase B Input Swap Select bit
  - QEI_INPUTS_SWAP
  - QEI_INPUTS_NOSWAP

- QEI Mode of operation select
  - QEI_MODE_x4_MATCH
  - QEI_MODE_x4_PULSE
  - QEI_MODE_x2_MATCH
  - QEI_MODE_x2_PULSE
  - QEI_MODE_TIMER
  - QEI_MODE_OFF

- Position Counter Direction Status
  - QEI_UP_COUNT
  - QEI_DOWN_COUNT

config2 This contains the parameters to be configured in the DFLTxCON register.

In 4x Quadrature Count Mode:

- Required State of Phase A input signal for match on index pulse
  - MATCH_INDEX_PHASEA_HIGH
  - MATCH_INDEX_PHASEA_LOW

- Required State of Phase B input signal for match on index pulse
  - MATCH_INDEX_PHASEB_HIGH
  - MATCH_INDEX_PHASEB_LOW
OpenQEI (Continued)

In 2x Quadrature Count Mode:
Phase input signal for index state match
MATCH_INDEX_INPUT_PHASEA
MATCH_INDEX_INPUT_PHASEB
Phase input signal state for match on index pulse
MATCH_INDEX_INPUT_HIGH
MATCH_INDEX_INPUT_LOW
Enable/Disable interrupt due to position count event
POS_CNT_ERR_INT_ENABLE
POS_CNT_ERR_INT_DISABLE
QEA/QEB Digital Filter Clock Divide Select bits
QEI_QE_CLK_DIVIDE_1_1
QEI_QE_CLK_DIVIDE_1_2
QEI_QE_CLK_DIVIDE_1_4
QEI_QE_CLK_DIVIDE_1_16
QEI_QE_CLK_DIVIDE_1_32
QEI_QE_CLK_DIVIDE_1_64
QEI_QE_CLK_DIVIDE_1_128
QEI_QE_CLK_DIVIDE_1_256
QEA/QEB Digital Filter Output Enable
QEI_QE_OUT_ENABLE
QEI_QE_OUT_DISABLE

Return Value: None
Remarks: This function configures the QEICON and DFLTCON registers of QEI module.
This function also clears the QEICON<CNTERR> bit.

Source File: OpenQEI.c
Code Example:
OpenQEI(QEI_DIR_SEL_QEB & QEI_INT_CLK & QEI_INDEX_RESET_ENABLE & QEI_CLK_PRESCALE_1 & QEI_NORMAL_IO & QEI_MODE_TIMER & QEI_UP_COUNT,0);

ReadQEI

Description: This function read the position count value from the POSCNT register.
Include: qei.h
Prototype: unsigned int ReadQEI(void);
Arguments: None
Remarks: None
Return Value: This functions returns the contents of the POSCNT register.
Source File: ReadQEI.c
Code Example:
unsigned int pos_count;
pos_count = ReadQEI();
### 3.15.2 Individual Macros

#### WriteQEI

**Description:** This function sets the maximum count value for QEI.

**Include:** qei.h

**Prototype:**

```c
void WriteQEI(unsigned int position);
```

**Arguments:**

- `position`  
  This is the value to be stored into the MAXCNT register.

**Return Value**  
None

**Remarks**  
None

**Source File:** WriteQEI.c

**Code Example:**

```c
define unsigned int position = 0x3FFF;
WriteQEI(position);
```

#### EnableIntQEI

**Description:** This macro enables the QEI interrupt.

**Include:** qei.h

**Arguments:** None

**Remarks:** This macro sets QEI Interrupt Enable bit of Interrupt Enable Control register.

**Code Example:**

```c
EnableIntQEI;
```

#### DisableIntQEI

**Description:** This macro disables the QEI interrupt.

**Include:** qei.h

**Arguments:** None

**Remarks:** This macro clears QEI Interrupt Enable bit of Interrupt Enable Control register.

**Code Example:**

```c
DisableIntQEI;
```

#### SetPriorityIntQEI

**Description:** This macro sets priority for QEI interrupt.

**Include:** qei.h

**Arguments:** `priority`

**Remarks:** This macro sets QEI Interrupt Priority bits of Interrupt Priority Control register.

**Code Example:**

```c
SetPriorityIntQEI(7);
```
### 3.15.3 Example of Use

```c
#define __dsPIC30F6010__
#include <p30fxxxx.h>
#include<qei.h>
unsigned int pos_value;

void __attribute__((__interrupt__)) _QEIInterrupt(void)
{
    PORTDbits.RD1 = 1;    /* turn off LED on RD1 */
    POSCNT = 0;
    IFS2bits.QEIIF = 0;    /* Clear QEI interrupt flag */
}

int main(void)
{
    unsigned int max_value;
    TRISDbits.TRISD1 = 0;
    PORTDbits.RD1 = 1;    /* turn off LED on RD1 */
    
    /* Enable QEI Interrupt and Priority to "1" */
    ConfigIntQEI(QEI_INT_PRI_1 & QEI_INT_ENABLE);
    POSCNT  = 0;
    MAXCNT = 0xFFFF;
    OpenQEI(QEI_INT_CLK & QEI_INDEX_RESET_ENABLE &
           QEI_CLK_PRESCALE_256 &
           QEI_GATED_ACC_DISABLE & QEI_INPUTS_NOSWAP &
           QEI_MODE_TIMER & QEI_DIR_SEL_CNTRL &
           QEI_IDLE_CON, 0);
    QEICONbits.UPDN = 1;
    while(1)
    {
        pos_value = ReadQEI();
        if(pos_value >= 0x7FFF)
        {
            PORTDbits.RD1 = 0; /* turn on LED on RD1 */
        }
        PORTDbits.RD1 = 0; /* turn on LED on RD1 */
    }
    CloseQEI();
}
```
3.16 PWM FUNCTIONS

This section contains a list of individual functions for PWM module and an example of use of the functions. Functions may be implemented as macros.

3.16.1 Individual Functions

CloseMCPWM

Description: This function turns off the Motor Control PWM module.
Include: `pwm.h`
Prototype: `void closeMCPWM(void);`
Arguments: None
Return Value: None
Remarks: This function disables the Motor control PWM module and clears the PWM, Fault A and Fault B Interrupt Enable and Flag bits. This function also clears the PTCON, PWMCON1 and PWMCON2 registers.
Source File: `CloseMCPWM.c`
Code Example: `CloseMCPWM();`

ConfigIntMCPWM

Description: This function configures the PWM interrupts.
Include: `pwm.h`
Prototype: `void ConfigIntMCPWM(unsigned int config);`
Arguments: `config` PWM interrupt priority and enable/disable information as defined below:
- **PWM Interrupt enable/disable**
  - `PWM_INT_EN`
  - `PWM_INT_DIS`
- **PWM Interrupt priority**
  - `PWM_INT_PR0`
  - `PWM_INT_PR1`
  - `PWM_INT_PR2`
  - `PWM_INT_PR3`
  - `PWM_INT_PR4`
  - `PWM_INT_PR5`
  - `PWM_INT_PR6`
  - `PWM_INT_PR7`
- **Fault A Interrupt enable/disable**
  - `PWM_FLTA_EN_INT`
  - `PWM_FLTA_DIS_INT`
- **Fault A Interrupt priority**
  - `PWM_FLTA_INT_PR0`
  - `PWM_FLTA_INT_PR1`
  - `PWM_FLTA_INT_PR2`
  - `PWM_FLTA_INT_PR3`
  - `PWM_FLTA_INT_PR4`
  - `PWM_FLTA_INT_PR5`
  - `PWM_FLTA_INT_PR6`
  - `PWM_FLTA_INT_PR7`
- **Fault B Interrupt enable/disable**
  - `PWM_FLTB_EN_INT`
  - `PWM_FLTB_DIS_INT`
Fault B Interrupt priority
PWM_FLTB_INT_PR0
PWM_FLTB_INT_PR1
PWM_FLTB_INT_PR2
PWM_FLTB_INT_PR3
PWM_FLTB_INT_PR4
PWM_FLTB_INT_PR5
PWM_FLTB_INT_PR6
PWM_FLTB_INT_PR7

Return Value
None

Remarks:
This function clears the Interrupt Flag bit, sets the interrupt priority and enables/disables the interrupt.

Source File:
ConfigIntMCPWM.c

Code Example:
ConfigIntMCPWM(PWM_INT_EN & PWM_INT_PR5 & PWM_FLTA_EN_INT & PWM_FLTA_INT_PR6 & PWM_FLTB_EN_INT & PWM_FLTB_INT_PR7);

OpenMCPWM

Description:
This function configure the motor control PWM module.

Include:
pwm.h

Prototype:
void OpenMCPWM(unsigned int period,
unsigned int sptime,
unsigned int config1,
unsigned int config2,
unsigned int config3);

Arguments:
period This contains the PWM timebase period value to be stored in PTPER register.
sptime This contains the special event compare value to be stored in SEVTCMP register.
config1 This contains the parameters to be configured in the PTCON register as defined below:

PWM module enable/disable
PWM_EN
PWM_DIS

Idle mode enable/disable
PWM_IDLE_STOP
PWM_IDLE_CON

Output post scaler select
PWM_OP_SCALE1
PWM_OP_SCALE2
.......
PWM_OP_SCALE15
PWM_OP_SCALE16

Input prescaler select
PWM_IPCLK_SCALE1
PWM_IPCLK_SCALE4
PWM_IPCLK_SCALE16
PWM_IPCLK_SCALE64
OpenMCPWM (Continued)

PWM mode of operation
PWM_MOD_FREE
PWM_MOD_SING
PWM_MOD_UPDN
PWM_MOD_DBL

`config2` This contains the parameters to be configured in the PWMCON1 register as defined below:

PWM I/O pin pair
PWM_MOD4_COMP
PWM_MOD3_COMP
PWM_MOD2_COMP
PWM_MOD1_COMP
PWM_MOD4_IND
PWM_MOD3_IND
PWM_MOD2_IND
PWM_MOD1_IND

PWM H/L I/O enable/disable select
PWM_PEN4H
PWM_PDIS4H
PWM_PEN3H
PWM_PDIS3H
PWM_PEN2H
PWM_PDIS2H
PWM_PEN1H
PWM_PDIS1H
PWM_PEN4L
PWM_PDIS4L
PWM_PEN3L
PWM_PDIS3L
PWM_PEN2L
PWM_PDIS2L
PWM_PEN1L
PWM_PDIS1L

Bit defines related to PWM4 is available only for certain devices and the suitable data sheet should be referred to.

`config3` This contains the parameters to be configured in the PWMCON2 register as defined below:

Special event post scaler
PWM_SEVOPS1
PWM_SEVOPS2
.....
PWM_SEVOPS15
PWM_SEVOPS16

Output Override synchronization select
PWM_OSYNC_PWM
PWM_OSYNC_Tcy

PWM update enable/disable
PWM_UDIS
PWM_UEN

**Return Value** None

**Remarks:** This function configures the PTPER, SEVTCMP, PTCON, PWMCON1 and PWMCON2 registers.
### OpenMCPWM (Continued)

<table>
<thead>
<tr>
<th>Source File:</th>
<th>OpenMCPWM.c</th>
</tr>
</thead>
</table>
| Code Example: | period = 0x7fff;  
sptime = 0x0;  
cconfig1 = PWM_EN & PWM_PTSIDL_DIS &  
PWM_OP_SCALE16 &  
PWM_IPCLK_SCALE16 &  
PWM_MOD_UPDN;  
cconfig2 = PWM_MOD1_COMP & PWM_PDIS4H &  
PWM_PDIS3H & PWM_PDIS2H &  
PWM_PEN1H & PWM_PDIS4L &  
PWM_PDIS3L & PWM_PDIS2L &  
PWM_PEN1L;  
cconfig3 = PWM_SEVOPS1 & PWM_OSYNC_PWM &  
PWM_UEN;  
OpenMCPWM(period, sptime, config1,  
config2, config3); |

### OverrideMCPWM

<table>
<thead>
<tr>
<th>Description:</th>
<th>This function configures the OVDCON register.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>pwm.h</td>
</tr>
<tr>
<td>Prototype:</td>
<td>void OverrideMCPWM(unsigned int config);</td>
</tr>
</tbody>
</table>
| Arguments:   | config This contains the parameters to be configured in the  
OVDCON register as defined below: |

#### Output controlled by PWM generator

<table>
<thead>
<tr>
<th>PWM_GEN_4H</th>
<th>PWM_GEN_3H</th>
<th>PWM_GEN_2H</th>
<th>PWM_GEN_1H</th>
<th>PWM_GEN_4L</th>
<th>PWM_GEN_3L</th>
<th>PWM_GEN_2L</th>
<th>PWM_GEN_1L</th>
</tr>
</thead>
</table>

Bit defines related to PWM4 is available only for certain devices and the suitable data sheet should be referred to.

#### Output controlled by POUT bits

<table>
<thead>
<tr>
<th>PWM_POUT_4H</th>
<th>PWM_POUT_4L</th>
<th>PWM_POUT_3H</th>
<th>PWM_POUT_3L</th>
<th>PWM_POUT_2H</th>
<th>PWM_POUT_2L</th>
<th>PWM_POUT_1H</th>
<th>PWM_POUT_1L</th>
</tr>
</thead>
</table>

Bit defines related to PWM4 is available only for certain devices and the suitable data sheet should be referred to.
OverrideMCPWM (Continued)

PWM Manual Output bits
PWM_POUT4H_ACT
PWM_POUT4H_INACT
PWM_POUT4L_ACT
PWM_POUT4L_INACT
PWM_POUT3H_ACT
PWM_POUT3H_INACT
PWM_POUT3L_ACT
PWM_POUT3L_INACT
PWM_POUT2H_ACT
PWM_POUT2H_INACT
PWM_POUT2L_ACT
PWM_POUT2L_INACT
PWM_POUT1H_ACT
PWM_POUT1H_INACT
PWM_POUT1L_ACT
PWM_POUT1L_INACT

Bit defines related to PWM4 is available only for certain devices and the suitable data sheet should be referred to.

Return Value
None

Remarks:
This function configures the PWM Output Override and Manual Control bits of the OVDCON register.

Source File:
OverrideMCPWM.c

Code Example:
config = PWM_GEN_1L &
        PWM_GEN_1H &
        PWM_POUT1L_INACT &
        PWM_POUT3L_INACT;
OverrideMCPWM(config);

SetDCMCPWM

Description:
This function configures the Duty Cycle register and updates the ‘PWM Update Disable’ bit in the PWMCON2 register.

Include:
pwm.h

Prototype:
void SetDCMCPWM(
    unsigned int dutycyclereg,
    unsigned int dutycycle,
    char updatedisable);

Arguments:
dutycyclereg This is the pointer to the Duty Cycle register.
dutycycle This is the value to be stored in the Duty Cycle register.
updatedisable This is the value to be loaded into the ‘Update Disable’ bit of the PWMCON2 register.

Return Value
None

Remarks:
None

Source File:
SetDCMCPWM.c

Code Example:
dutycyclereg = 1;
dutycycle = 0xFFF;
updatedisable = 0;
SetDCMCPWM(dutycyclereg, dutycycle, updatedisable);
### SetMCPWMDeadTimeAssignment

**Description:** This function configures the assignment of dead-time units to PWM output pairs.

**Include:** `pwm.h`

**Prototype:**

```c
void SetMCPWMDeadTimeAssignment (unsigned int config);
```

**Arguments:**
- `config`  This contains the parameters to be configured in the DTCON2 register as defined below:

**Dead-Time Select bits for PWM4 signal**  
- `PWM_DTS4A_UA`
- `PWM_DTS4A_UB`
- `PWM_DTS4I_UA`
- `PWM_DTS4I_UB`

Note: Bit defines related to PWM4 is available only for certain devices and the suitable data sheet should be referred to.

**Dead-Time Select bits for PWM3 signal**  
- `PWM_DTS3A_UA`
- `PWM_DTS3A_UB`
- `PWM_DTS3I_UA`
- `PWM_DTS3I_UB`

**Dead-Time Select bits for PWM2 signal**  
- `PWM_DTS2A_UA`
- `PWM_DTS2A_UB`
- `PWM_DTS2I_UA`
- `PWM_DTS2I_UB`

**Dead-Time Select bits for PWM1 signal**  
- `PWM_DTS1A_UA`
- `PWM_DTS1A_UB`
- `PWM_DTS1I_UA`
- `PWM_DTS1I_UB`

**Return Value:** None

**Remarks:** None

**Source File:** `SetMCPWMDeadTimeAssignment.c`

**Code Example:**

```c
SetMCPWMDeadTimeAssignment(PWM_DTS3A_UA & PWM_DTS2I_UA & PWM_DTS1I_UA);
```
**SetMCPWMDeadTimeGeneration**

**Description:**
This function configures dead-time values and clock prescalers.

**Include:**
`pwm.h`

**Prototype:**
```c
void SetMCPWMDeadTimeGeneration(
    unsigned int config);
```

**Arguments:**
- `config`: This contains the parameters to be configured in the DTCON1 register as defined below:

  **Dead-Time Unit B Prescale Select bits**
  - `PWM_DTBPS8`
  - `PWM_DTBPS4`
  - `PWM_DTBPS2`
  - `PWM_DTBPS1`

  **Dead-Time Unit A Prescale Select constants**
  - `PWM_DTA0`
  - `PWM_DTA1`
  - `PWM_DTA2`
  - `................`  
  - `PWM_DTA62`
  - `PWM_DTA63`

  **Dead-Time Unit B Prescale Select constants**
  - `PWM_DTB0`
  - `PWM_DTB1`
  - `PWM_DTB2`
  - `................`  
  - `PWM_DTB62`
  - `PWM_DTB63`

  **Dead-Time Unit A Prescale Select bits**
  - `PWM_DTAPS8`
  - `PWM_DTAPS4`
  - `PWM_DTAPS2`
  - `PWM_DTAPS1`

**Return Value:**
None

**Remarks:**
None

**Source File:**
`SetMCPWMDeadTimeGeneration.c`

**Code Example:**
```c
SetMCPWMDeadTimeGeneration(PWM_DTBPS16 & PWM_DT54 & PWM_DTAPS8);
```
SetMCPWMFaultA

Description: This function configures Fault A Override bits, Fault A Mode bit and Fault Input A Enable bits of PWM.

Include:  pwm.h

Prototype:  void SetMCPWMFaultA(unsigned int config);

Arguments:  config  This contains the parameters to be configured in the FLTACON register as defined below:

Fault Input A PWM Override Value bits
PWM_OVA4H_ACTIVE
PWM_OVA3H_ACTIVE
PWM_OVA2H_ACTIVE
PWM_OVA1H_ACTIVE
PWM_OVA4L_ACTIVE
PWM_OVA3L_ACTIVE
PWM_OVA2L_ACTIVE
PWM_OVA1L_ACTIVE
PWM_OVA4H_INACTIVE
PWM_OVA3H_INACTIVE
PWM_OVA2H_INACTIVE
PWM_OVA1H_INACTIVE
PWM_OVA4L_INACTIVE
PWM_OVA3L_INACTIVE
PWM_OVA2L_INACTIVE
PWM_OVA1L_INACTIVE

Bit defines related to PWM4 is available only for certain devices and the suitable data sheet should be referred to.

Fault A Mode bit
PWM_FLTA_MODE_CYCLE
PWM_FLTA_MODE_LATCH

Fault Input A Enable bits.
PWM_FLTA4_EN
PWM_FLTA4_DIS
PWM_FLTA3_EN
PWM_FLTA3_DIS
PWM_FLTA2_EN
PWM_FLTA2_DIS
PWM_FLTA1_EN
PWM_FLTA1_DIS

Bit defines related to PWM4 is available only for certain devices and the suitable data sheet should be referred to.

Return Value  None

Remarks:  None

Source File:  SetMCPWMFaultA.c

Code Example:  SetMCPWMFaultA(PWM_OVA3L_INACTIVE & PWM_FLTA_MODE_LATCH & PWM_FLTA1_DIS);
SetMCPWMFaultB

**Description:** This function configures Fault B Override bits, Fault B Mode bit and Fault Input B Enable bits of PWM.

**Include:** pvm.h

**Prototype:**
```c
void SetMCPWMFaultB(unsigned int config);
```

**Arguments:**
- `config` This contains the parameters to be configured in the FLTBCON register as defined below:
- FLTBCON register is available only for certain devices and the suitable data sheet should be referred to.

**Fault Input B PWM Override Value bits**
- PWM_OVB4H_ACTIVE
- PWM_OVB3H_ACTIVE
- PWM_OVB2H_ACTIVE
- PWM_OVB1H_ACTIVE
- PWM_OVB4L_ACTIVE
- PWM_OVB3L_ACTIVE
- PWM_OVB2L_ACTIVE
- PWM_OVB1L_ACTIVE
- PWM_OVB4H_INACTIVE
- PWM_OVB3H_INACTIVE
- PWM_OVB2H_INACTIVE
- PWM_OVB1H_INACTIVE
- PWM_OVB4L_INACTIVE
- PWM_OVB3L_INACTIVE
- PWM_OVB2L_INACTIVE
- PWM_OVB1L_INACTIVE

**Fault B Mode bit**
- PWM_FLTB_MODE_CYCLE
- PWM_FLTB_MODE_LATCH

**Fault Input B Enable bits,**
- PWM_FLTB4_EN
- PWM_FLTB4_DIS
- PWM_FLTB3_EN
- PWM_FLTB3_DIS
- PWM_FLTB2_EN
- PWM_FLTB2_DIS
- PWM_FLTB1_EN
- PWM_FLTB1_DIS

**Return Value** None

**Remarks** None

**Source File:** SetMCPWMFaultB.c

**Code Example:**
```c
SetMCPWMFaultB(PWM_OVB3L_INACTIVE &
                PWM_FLTB_MODE_LATCH &
                PWM_FLTB2_DIS);
```
3.16.2 Individual Macros

### EnableIntMCPWM

**Description:** This macro enables the PWM interrupt.

**Include:** pwm.h

**Arguments:** None

**Remarks:** This macro sets PWM Interrupt Enable bit of Interrupt Enable Control register.

**Code Example:**

```c
EnableIntMCPWM;
```

### DisableIntMCPWM

**Description:** This macro disables the PWM interrupt.

**Include:** pwm.h

**Arguments:** None

**Remarks:** This macro clears PWM Interrupt Enable bit of Interrupt Enable Control register.

**Code Example:**

```c
DisableIntMCPWM;
```

### SetPriorityIntMCPWM

**Description:** This macro sets priority for PWM interrupt.

**Include:** pwm.h

**Arguments:** priority

**Remarks:** This macro sets PWM Interrupt Priority bits of Interrupt Priority Control register.

**Code Example:**

```c
SetPriorityIntMCPWM(7);
```

### EnableIntFLTA

**Description:** This macro enables the FLTA interrupt.

**Include:** pwm.h

**Arguments:** None

**Remarks:** This macro sets FLTA Interrupt Enable bit of Interrupt Enable Control register.

**Code Example:**

```c
EnableIntFLTA;
```

### DisableIntFLTA

**Description:** This macro disables the FLTA interrupt.

**Include:** pwm.h

**Arguments:** None

**Remarks:** This macro clears FLTA Interrupt Enable bit of Interrupt Enable Control register.

**Code Example:**

```c
DisableIntFLTA;
```
### SetPriorityIntFLTA

**Description:** This macro sets priority for FLTA interrupt.

**Include:** `pwm.h`

**Arguments:** `priority`

**Remarks:** This macro sets FLTA Interrupt Priority bits of Interrupt Priority Control register.

**Code Example:**
```
SetPriorityIntFLTA(7);
```

### EnableIntFLTB

**Description:** This macro enables the FLTB interrupt.

**Include:** `pwm.h`

**Arguments:** None

**Remarks:** This macro sets FLTB Interrupt Enable bit of Interrupt Enable Control register.

**Code Example:**
```
EnableIntFLTB;
```

### DisableIntFLTB

**Description:** This macro disables the FLTB interrupt.

**Include:** `pwm.h`

**Arguments:** None

**Remarks:** This macro clears FLTB Interrupt Enable bit of Interrupt Enable Control register.

**Code Example:**
```
DisableIntFLTB;
```

### SetPriorityIntFLTB

**Description:** This macro sets priority for FLTB interrupt.

**Include:** `pwm.h`

**Arguments:** `priority`

**Remarks:** This macro sets FLTB Interrupt Priority bits of Interrupt Priority Control register.

**Code Example:**
```
SetPriorityIntFLTB(1);
```
3.16.3 Example of Use

#define __dsPIC30F6010__
#include <p30fxxxx.h>
#include<pwm.h>
void __attribute__((__interrupt__)) _PWMInterrupt(void)
{
  IFS2bits.PWMIF = 0;
}

int main()
{
  /* Holds the PWM interrupt configuration value*/
  unsigned int config;
  /* Holds the value to be loaded into dutycycle register */
  unsigned int period;
  /* Holds the value to be loaded into special event compare register */
  unsigned int sptime;
  /* Holds PWM configuration value */
  unsigned int config1;
  /* Holds the value be loaded into PWMCON1 register */
  unsigned int config2;
  /* Holds the value to configure the special event trigger postscale and dutycycle */
  unsigned int config3;
  /* The value of 'dutycyclereg' determines the duty cycle register(PDCx) to be written */
  unsigned int dutycyclereg;
  unsigned int dutycycle;
  unsigned char updatedisable;

  /* Configure pwm interrupt enable/disable and set interrupt priorities */
  config = (PWM_INT_EN & PWM_FLTA_DIS_INT & PWM_INT_PR1
            & PWM_FLTA_INT_PR0
            & PWM_FLTB_DIS_INT & PWM_FLTB_INT_PR0);
  ConfigIntMCPWM(config);
  /* Configure PWM to generate square wave of 50% duty cycle */
  dutycyclereg = 1;
  dutycycle = 0x3FFF;
  updatedisable = 0;
  SetDCMCPWM(dutycyclereg,dutycycle,updatedisable);
  period = 0x7fff;
  sptime = 0x0;
  config1 = (PWM_EN & PWM_PTSIDL_DIS & PWM_OP_SCALE16
             & PWM_IPCLK_SCALE16 &
             PWM_MOD_UPDN);
  config2 = (PWM_MOD1_COMP & PWM_PDIS4H & PWM_PDIS3H &
             PWM_PDIS2H & PWM_PEN1H & PWM_PDIS4L &
             PWM_PDIS3L & PWM_PDIS2L & PWM_PEN1L);
  config3 = (PWM_SEVOPS1 & PWM_OSYNC_PWM & PWM_UEN);
  OpenMCPWM(period,sptime,config1,config2,config3);
  while(1);
}
3.17 I²C™ FUNCTIONS

This section contains a list of individual functions for I²C module and an example of use of the functions. Functions may be implemented as macros.

3.17.1 Individual Functions

**CloseI2C**

Description: This function turns off the I²C module
Include: i2c.h
Prototype: void CloseI2C(void);
Arguments: None
Return Value None
Remarks: This function disables the I²C module and clears the Master and Slave Interrupt Enable and Flag bits.
Source File: CloseI2C.c
Code Example: CloseI2C();

**ConfigIntI2C**

Description: This function configures the I²C Interrupt.
Include: i2c.h
Prototype: void ConfigIntI2C(unsigned int config);
Arguments: config I²C interrupt priority and enable/disable information as defined below:

<table>
<thead>
<tr>
<th>I²C master Interrupt enable/disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>MI2C_INT_ON</td>
</tr>
<tr>
<td>MI2C_INT_OFF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I²C slave Interrupt enable/disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI2C_INT_ON</td>
</tr>
<tr>
<td>SI2C_INT_OFF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I²C master Interrupt priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>MI2C_INT_PRI_7</td>
</tr>
<tr>
<td>MI2C_INT_PRI_6</td>
</tr>
<tr>
<td>MI2C_INT_PRI_5</td>
</tr>
<tr>
<td>MI2C_INT_PRI_4</td>
</tr>
<tr>
<td>MI2C_INT_PRI_3</td>
</tr>
<tr>
<td>MI2C_INT_PRI_2</td>
</tr>
<tr>
<td>MI2C_INT_PRI_1</td>
</tr>
<tr>
<td>MI2C_INT_PRI_0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I²C slave Interrupt priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI2C_INT_PRI_7</td>
</tr>
<tr>
<td>SI2C_INT_PRI_6</td>
</tr>
<tr>
<td>SI2C_INT_PRI_5</td>
</tr>
<tr>
<td>SI2C_INT_PRI_4</td>
</tr>
<tr>
<td>SI2C_INT_PRI_3</td>
</tr>
<tr>
<td>SI2C_INT_PRI_2</td>
</tr>
<tr>
<td>SI2C_INT_PRI_1</td>
</tr>
<tr>
<td>SI2C_INTPRI_0</td>
</tr>
</tbody>
</table>

Return Value None
### ConfigIntI2C (Continued)

**Remarks:** This function clears the Interrupt Flag bits, sets the interrupt priorities of master and slave and enables/disables the interrupt.

**Source File:** ConfigIntI2C.c

**Code Example:**
```
ConfigIntI2C(MI2C_INT_ON & MI2C_INT_PRI_3
 & SI2C_INT_ON & SI2C_INT_PRI_5);
```

### AckI2C

**Description:** Generates I^2^C bus Acknowledge condition.

**Include:** i2c.h

**Prototype:**
```
void AckI2C(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:** This function generates an I^2^C bus Acknowledge condition.

**Source File:** AckI2C.c

**Code Example:**
```
AckI2C();
```

### DataRdyI2C

**Description:** This function provides status back to user if I2CRCV register contain data.

**Include:** i2c.h

**Prototype:**
```
unsigned char DataRdyI2C(void);
```

**Arguments:** None

**Return Value:** This function returns ‘1’ if there is data in I2CRCV register; else return ‘0’ which indicates no data in I2CRCV register.

**Remarks:** This function determines if there is any byte to read from I2CRCV register.

**Source File:** DataRdyI2C.c

**Code Example:**
```
if(DataRdyI2C());
```

### IdleI2C

**Description:** This function generates Wait condition until I^2^C bus is Idle.

**Include:** i2c.h

**Prototype:**
```
void IdleI2C(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:** This function will be in a wait state until Start Condition Enable bit, Stop Condition Enable bit, Receive Enable bit, Acknowledge Sequence Enable bit of I^2^C Control register and Transmit Status bit I^2^C Status register are clear. The IdleI2C function is required since the hardware I^2^C peripheral does not allow for spooling of bus sequence. The I^2^C peripheral must be in Idle state before an I^2^C operation can be initiated or write collision will be generated.

**Source File:** IdleI2C.c

**Code Example:**
```
IdleI2C();
```
### MastergetsI2C

**Description:** This function reads predetermined data string length from the I²C bus.

**Include:** i2c.h

**Prototype:**

```c
unsigned int MastergetsI2C(unsigned int length, 
unsigned char *rdptr, unsigned int i2c_data_wait);
```

**Arguments:**

- `length`: Number of bytes to read from I²C device.
- `rdptr`: Character type pointer to RAM for storage of data read from I²C device.
- `i2c_data_wait`: This is the time-out count for which the module has to wait before return. If the time-out count is 'N', the actual time out would be about (20 * N – 1) instruction cycles.

**Return Value**

This function returns '0' if all bytes have been sent or number of bytes read from I²C bus if its not able to read the data with in the specified `i2c_data_wait` time out value.

**Remarks:**

This routine reads a predefined data string from the I²C bus.

**Source File:** MastergetsI2C.c

**Code Example:**

```c
unsigned char string[10];
unsigned char *rdptr;
unsigned int length, i2c_data_wait;
length = 9;
rdptr = string;
i2c_data_wait = 152;
MastergetsI2C(length, rdptr, i2c_data_wait);
```

### MasterputsI2C

**Description:** This function is used to write out a data string to the I²C bus.

**Include:** i2c.h

**Prototype:**

```c
unsigned int MasterputsI2C(unsigned char *wrptr);
```

**Arguments:**

- `wrptr`: Character type pointer to data objects in RAM. The data objects are written to the I²C device.

**Return Value**

This function returns -3 if a write collision occurred. This function returns '0' if the null character was reached in data string.

**Remarks:**

This function writes a string to the I²C bus until a null character is reached. Each byte is written via a call to the `MasterputcI2C` function. The actual called function body is termed `MasterWriteI2C`. `MasterWriteI2C` and `MasterputcI2C` refer to the same function via a #define statement in the i2c.h

**Source File:** MasterputsI2C.c

**Code Example:**

```c
unsigned char string[] = " MICROCHIP ";
unsigned char *wrptr;
wrptr = string;
MasterputsI2C( wrptr);
```
MasterReadI2C

Description: This function is used to read a single byte from I²C bus.
Include: i2c.h
Prototype: unsigned char MasterReadI2C(void);
Arguments: None
Return Value: The return value is the data byte read from the I²C bus.
Remarks: This function reads in a single byte from the I²C bus. This function performs the same function as MastergetcI2C.
Source File: MasterReadI2C.c
Code Example: unsigned char value;
value = MasterReadI2C();

MasterWriteI2C

Description: This function is used to write out a single data byte to the I²C device.
Include: i2c.h
Prototype: unsigned char MasterWriteI2C(unsigned char data_out);
Arguments: data_out A single data byte to be written to the I²C bus device.
Return Value: This function returns -1 if there was a write collision else it returns a 0.
Remarks: This function writes out a single data byte to the I²C bus device. This function performs the same function as MasterputcI2C.
Source File: MasterWriteI2C.c
Code Example: MasterWriteI2C('a');

NotAckI2C

Description: Generates I²C bus Not Acknowledge condition.
Include: i2c.h
Prototype: void NotAckI2C(void);
Arguments: None
Return Value: None
Remarks: This function generates an I²C bus Not Acknowledge condition.
Source File: NotAckI2C.c
Code Example: NotAckI2C();
## OpenI2C

### Description:
Configures the I2C module.

### Include:
`i2c.h`

### Prototype:
```c
void OpenI2C(unsigned int config1, unsigned int config2);
```

### Arguments:
- `config1`  
  This contains the parameter to configure the I2CCON register
  - **I2C Enable bit**
    - I2C_ON
    - I2C_OFF
  - **I2C Stop in Idle Mode bit**
    - I2C_IDLE_STOP
    - I2C_IDLE_CON
  - **SCL Release Control bit**
    - I2C_CLK_REL
    - I2C_CLK_HLD
  - **Intelligent Peripheral Management Interface Enable bit**
    - I2C_IPMI_EN
    - I2C_IPMI_DIS
  - **10-bit Slave Address bit**
    - I2C_10BIT_ADD
    - I2C_7BIT_ADD
  - **Disable Slew Rate Control bit**
    - I2C_SLW_DIS
    - I2C_SLW_EN
  - **SMBus Input Level bits**
    - I2C_SM_EN
    - I2C_SM_DIS
  - **General Call Enable bit**
    - I2C_GCALL_EN
    - I2C_GCALL_DIS
  - **SCL Clock Stretch Enable bit**
    - I2C_STR_EN
    - I2C_STR_DIS
  - **Acknowledge Data bit**
    - I2C_ACK
    - I2C_NACK
  - **Acknowledge Sequence Enable bit**
    - I2C_ACK_EN
    - I2C_ACK_DIS
  - **Receive Enable bit**
    - I2C_RCV_EN
    - I2C_RCV_DIS
  - **Stop Condition Enable bit**
    - I2C_STOP_EN
    - I2C_STOP_DIS
  - **Repeated Start Condition Enable bit**
    - I2C_RESTART_EN
    - I2C_RESTART_DIS
  - **Start Condition Enable bit**
    - I2C_START_EN
    - I2C_START_DIS

- `config2`  
  computed value for the baud rate generator
### OpenI2C (Continued)

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
<tr>
<td>Remarks</td>
<td>This function configures the I²C Control register and I²C Baud Rate Generator register.</td>
</tr>
<tr>
<td>Source File</td>
<td>OpenI2C.c</td>
</tr>
<tr>
<td>Code Example</td>
<td>OpenI2C();</td>
</tr>
</tbody>
</table>

### RestartI2C

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Generates I²C Bus Restart condition.</td>
</tr>
<tr>
<td>Include</td>
<td>i2c.h</td>
</tr>
<tr>
<td>Prototype</td>
<td>void RestartI2C(void);</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
<tr>
<td>Remarks</td>
<td>This function generates an I²C Bus Restart condition.</td>
</tr>
<tr>
<td>Source File</td>
<td>RestartI2C.c</td>
</tr>
<tr>
<td>Code Example</td>
<td>RestartI2C();</td>
</tr>
</tbody>
</table>

### SlavegetsI2C

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This function reads pre-determined data string length from the I²C bus.</td>
</tr>
<tr>
<td>Include</td>
<td>i2c.h</td>
</tr>
<tr>
<td>Prototype</td>
<td>unsigned int SlavegetsI2C(unsigned char *rdptr, unsigned int i2c_data_wait);</td>
</tr>
<tr>
<td>Arguments</td>
<td>rdptr Character type pointer to RAM for storage of data read from I²C device.</td>
</tr>
<tr>
<td></td>
<td>i2c_data_wait This is the time-out count for which the module has to wait before return.</td>
</tr>
<tr>
<td></td>
<td>If the time-out count is 'N', the actual time out would be about (20*N - 1) instruction cycles.</td>
</tr>
<tr>
<td>Return Value</td>
<td>Returns the number of bytes received from the I²C bus.</td>
</tr>
<tr>
<td>Remarks</td>
<td>This routine reads a predefined data string from the I²C bus.</td>
</tr>
<tr>
<td>Source File</td>
<td>SlavegetsI2C.c</td>
</tr>
<tr>
<td>Code Example</td>
<td>unsigned char string[12];</td>
</tr>
<tr>
<td></td>
<td>unsigned char *rdptr;</td>
</tr>
<tr>
<td></td>
<td>rdptr = string;</td>
</tr>
<tr>
<td></td>
<td>i2c_data_out = 0x11;</td>
</tr>
<tr>
<td></td>
<td>SlavegetsI2C(rdptr, i2c_data_wait);</td>
</tr>
</tbody>
</table>
SlaveputsI2C

Description: This function is used to write out a data string to the I²C bus.
Include: i2c.h
Prototype: unsigned int SlaveputsI2C(unsigned char *wrptr);
Arguments: wrptr Character type pointer to data objects in RAM. The data objects are written to the I²C device.
Return Value This function returns '0' if the null character was reached in the data string.
Remarks: This routine writes a data string out to the I²C bus until a null character is reached.
Source File: SlaveputsI2C.c
Code Example: unsigned char string[] = "MICROCHIP";
unsigned char *rdptr;
rdptr = string;
SlaveputsI2C(rdptr);

SlaveReadI2C

Description: This function is used to read a single byte from the I²C bus.
Include: i2c.h
Prototype: unsigned char SlaveReadI2C(void);
Arguments: None
Return Value The return value is the data byte read from the I²C bus.
Remarks: This function reads in a single byte from the I²C bus. This function performs the same function as SlavegetcI2C.
Source File: SlaveReadI2C.c
Code Example: unsigned char value;
value = SlaveReadI2C();

SlaveWriteI2C

Description: This function is used to write out a single byte to the I²C bus.
Include: i2c.h
Prototype: void SlaveWriteI2C(unsigned char data_out);
Arguments: data_out A single data byte to be written to the I²C bus device.
Return Value None
Remarks: This function writes out a single data byte to the I²C bus device. This function performs the same function as SlaveputcI2C.
Source File: SlaveWriteI2C.c
Code Example: SlaveWriteI2C('a');
### StartI2C

**Description:** Generates I²C Bus Start condition.

**Include:** `i2c.h`

**Prototype:**

```c
void StartI2C(void);
```

**Arguments:** None

**Return Value** None

**Remarks:** This function generates a I²C Bus Start condition.

**Source File:** `StartI2C.c`

**Code Example:**

```c
StartI2C();
```

### StopI2C

**Description:** Generates I²C Bus Stop condition.

**Include:** `i2c.h`

**Prototype:**

```c
void StopI2C(void);
```

**Arguments:** None

**Return Value** None

**Remarks:** This function generates a I²C Bus Stop condition.

**Source File:** `StopI2C.c`

**Code Example:**

```c
StopI2C();
```

### 3.17.2 Individual Macros

#### EnableIntMI2C

**Description:** This macro enables the master I²C interrupt.

**Include:** `i2c.h`

**Arguments:** None

**Remarks:** This macro sets Master I²C Enable bit of Interrupt Enable Control register.

**Code Example:**

```c
EnableIntMI2C;
```

#### DisableIntMI2C

**Description:** This macro disables the master I²C interrupt.

**Include:** `i2c.h`

**Arguments:** None

**Remarks:** This macro clears Master I²C Interrupt Enable bit of Interrupt Enable Control register.

**Code Example:**

```c
DisableIntMI2C;
```
SetPriorityIntMI2C

Description: This macro sets priority for master I^2^C interrupt.
Include: i2c.h
Arguments: priority
Remarks: This macro sets Master I^2^C Interrupt Priority bits of Interrupt Priority Control register.
Code Example: SetPriorityIntMI2C(1);

EnableIntSI2C

Description: This macro enables the slave I^2^C interrupt.
Include: i2c.h
Arguments: None
Remarks: This macro sets Slave I^2^C Enable bit of Interrupt Enable Control register.
Code Example: EnableIntSI2C;

DisableIntSI2C

Description: This macro disables the slave I^2^C interrupt.
Include: i2c.h
Arguments: None
Remarks: This macro clears Slave I^2^C Interrupt Enable bit of Interrupt Enable Control register.
Code Example: DisableIntSI2C;

SetPriorityIntSI2C

Description: This macro sets priority for master I^2^C interrupt.
Include: i2c.h
Arguments: priority
Remarks: This macro sets Master I^2^C Interrupt Priority bits of Interrupt Priority Control register.
Code Example: SetPriorityIntSI2C(4);
3.17.3 Example of Use

#define __dsPIC30F6014__
#include <p30fxxxx.h>
#include<i2c.h>

void main(void )
{
  unsigned int config2, config1;
  unsigned char *wrptr;
  unsigned char tx_data[] =
  {'M','I','C','R','O','C','H','I','P','
  \0'};
  wrptr   = tx_data;
  /* Baud rate is set for 100 Khz */
  config2 = 0x11;
  /* Configure I2C for 7 bit address mode */
  config1 = (I2C_ON & I2C_IDLE_CON & I2C_CLK_HLD
 & I2C_IPMI_DIS & I2C_7BIT_ADD
 & I2C_SLW_DIS & I2C_SM_DIS &
 I2C_GCALL_DIS & I2C_STR_DIS &
 I2C_NACK & I2C_ACK_DIS & I2C_RCV_DIS &
 I2C_STOP_DIS & I2C_RESTART_DIS
 & I2C_START_DIS);
  OpenI2C(config1,config2);
  IdleI2C();
  StartI2C();
  /* Wait till Start sequence is completed */
  while(I2CCONbits.SEN );
  /* Write Slave address and set master for transmission */
  MasterWriteI2C(0xE);
  /* Wait till address is transmitted */
  while(I2CSTATbits.TBF);
  while(I2CSTATbits.ACKSTAT);
  /* Transmit string of data */
  MasterputsI2C(wrptr);
  StopI2C();
  /* Wait till stop sequence is completed */
  while(I2CCONbits.PEN);
  CloseI2C();
}
Chapter 4. Standard C Libraries with Math Functions

4.1 INTRODUCTION

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

Additionally, some 16-bit standard C library helper functions, and standard functions that must be modified for use with 16-bit devices, are in the library libpic30-omf.a.

4.1.1 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.

4.1.2 C Code Applications

The MPLAB C30 C compiler install directory (c:\Program Files\Microchip\MPLAB C30) 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

In addition, there is a file, ResourceGraphs.pdf, which contains diagrams of resources used by each function, located in lib.

4.1.3 Chapter Organization

This chapter is organized as follows:

• Using the Standard C Libraries

libc-omf.a

• <assert.h> diagnostics
• <ctype.h> character handling
• <errno.h> errors
• <float.h> floating-point characteristics
• <limits.h> implementation-defined limits
• <locale.h> localization
• <setjmp.h> non-local jumps
• <signal.h> signal handling
• <stdarg.h> variable argument lists
• <stddef.h> common definitions
• <stdio.h> input and output
• <stdlib.h> utility functions
• <string.h> string functions
• <time.h> date and time functions

libm-omf.a

• <math.h> mathematical functions
libpic30-omf.a
  • pic30-libs

4.2 USING THE STANDARD C LIBRARIES

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

4.2.1 Header Files

All standard C library entities are declared or defined in one or more standard headers (See list in Section 4.1.3 “Chapter Organization”.) 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.

4.2.2 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 three library files: libc-omf.a, libm-omf.a, and libpic30-omf.a. (See Section 1.2 “OMF-Specific Libraries/Start-up Modules” for more on OMF-specific libraries.) These libraries will be included automatically if linking is performed using the MPLAB C30 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 ASM30, MPLAB LINK30 and Utilities User’s Guide” (DS51317) and “MPLAB C30 C Compiler User’s Guide” (DS51284) for more information on the heap. |
4.3  <ASSERT.H> DIAGNOSTICS

The header file assert.h consists of a single macro that is useful for debugging logic errors in programs. By using the assert statement in critical locations where certain conditions should be true, the logic of the program may be tested.

Assertion testing may be turned off without removing the code by defining NDEBUG before including <assert.h>. If the macro NDEBUG is defined, assert() is ignored and no code is generated.

assert

Description: If the expression is false, an assertion message is printed to stderr and the program is aborted.

Include: <assert.h>

Prototype: void assert(int expression);

Argument: expression  The expression to test.

Remarks: The expression evaluates to zero or non-zero. If zero, the assertion fails, and a message is printed to stderr. The message includes the source file name (__FILE__), the source line number (__LINE__), the expression being evaluated and the message. The macro then calls the function abort(). If the macro _VERBOSE_DEBUGGING is defined, a message will be printed to stderr each time assert() is called.

Example:

```c
#include <assert.h> /* for assert */

int main(void)
{
    int a;

    a = 2 * 2;
    assert(a == 4); /* if true-nothing prints */
    assert(a == 6); /* if false-print message */
        /* and abort */
}
```

Output:

```
sampassert.c:9 a == 6 -- assertion failed
ABRT
```

with _VERBOSE_DEBUGGING defined:

```
sampassert.c:8 a == 4 -- OK
sampassert.c:9 a == 6 -- assertion failed
ABRT
```
4.4 <CTYPE.H> CHARACTER HANDLING

The header file `ctype.h` consists of functions that are useful for classifying and mapping characters. Characters are interpreted according to the Standard C locale.

**isalnum**

**Description:** Test for an alphanumeric character.

**Include:** `<ctype.h>`

**Prototype:**

```c
int isalnum(int c);
```

**Argument:** `c` The character to test.

**Return Value:** Returns a non-zero integer value if the character is alphanumeric; otherwise, returns zero.

**Remarks:** Alphanumeric characters are included within the ranges A-Z, a-z or 0-9.

**Example:**

```c
#include <ctype.h> /* for isalnum */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;

    ch = '3';
    if (isalnum(ch))
        printf("3 is an alphanumeric\n");
    else
        printf("3 is NOT an alphanumeric\n");

    ch = '#';
    if (isalnum(ch))
        printf("# is an alphanumeric\n");
    else
        printf("# is NOT an alphanumeric\n");
}
```

**Output:**

3 is an alphanumeric
# is NOT an alphanumeric

**isalpha**

**Description:** Test for an alphabetic character.

**Include:** `<ctype.h>`

**Prototype:**

```c
int isalpha(int c);
```

**Argument:** `c` The character to test.

**Return Value:** Returns a non-zero integer value if the character is alphabetic; otherwise, returns zero.

**Remarks:** Alphabetic characters are included within the ranges A-Z or a-z.
### isalpha (Continued)

**Example:**
```
#include <ctype.h> /* for isalpha */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;

    ch = 'B';
    if (isalpha(ch))
        printf("B is alphabetic\n");
    else
        printf("B is NOT alphabetic\n");

    ch = '#';
    if (isalpha(ch))
        printf("# is alphabetic\n");
    else
        printf("# is NOT alphabetic\n");
}
```

**Output:**
```
B is alphabetic
# is NOT alphabetic
```

### iscntrl

**Description:** Test for a control character.

**Include:** `<ctype.h>`

**Prototype:**
```
int iscntrl(int c);
```

**Argument:**
```
c    character to test.
```

**Return Value:** Returns a non-zero integer value if the character is a control character; otherwise, returns zero.

**Remarks:** A character is considered to be a control character if its ASCII value is in the range 0x00 to 0x1F inclusive, or 0x7F.

**Example:**
```
#include <ctype.h> /* for iscntrl */
#include <stdio.h> /* for printf */

int main(void)
{
    char ch;

    ch = 'B';
    if (iscntrl(ch))
        printf("B is a control character\n");
    else
        printf("B is NOT a control character\n");

    ch = 't';
    if (iscntrl(ch))
        printf("A tab is a control character\n");
    else
        printf("A tab is NOT a control character\n");
}
```

**Output:**
```
B is NOT a control character
A tab is a control character
```
isdigit

Description: Test for a decimal digit.
Include: <ctype.h>
Prototype: int isdigit(int c);
Argument: c character to test.
Return Value: Returns a non-zero integer value if the character is a digit; otherwise, returns zero.
Remarks: A character is considered to be a digit character if it is in the range of '0'- '9'.
Example:
#include <ctype.h> /* for isdigit */
#include <stdio.h> /* for printf */

int main(void)
{
  int ch;
  ch = '3';
  if (isdigit(ch))
    printf("3 is a digit\n");
  else
    printf("3 is NOT a digit\n");

  ch = '#';
  if (isdigit(ch))
    printf("# is a digit\n");
  else
    printf("# is NOT a digit\n");
}

Output:
3 is a digit
# is NOT a digit

isgraph

Description: Test for a graphical character.
Include: <ctype.h>
Prototype: int isgraph (int c);
Argument: c character to test
Return Value: Returns a non-zero integer value if the character is a graphical character; otherwise, returns zero.
Remarks: A character is considered to be a graphical character if it is any printable character except a space.
Example:
#include <ctype.h> /* for isgraph */
#include <stdio.h> /* for printf */

int main(void)
{
  int ch;
isgraph (Continued)

ch = '3';
if (isgraph(ch))
    printf("3 is a graphical character\n");
else
    printf("3 is NOT a graphical character\n");

ch = '#';
if (isgraph(ch))
    printf("# is a graphical character\n");
else
    printf("# is NOT a graphical character\n");

ch = ' ';
if (isgraph(ch))
    printf("a space is a graphical character\n");
else
    printf("a space is NOT a graphical character\n");
}

Output:
3 is a graphical character
# is a graphical character
a space is NOT a graphical character

islower

Description: Test for a lower case alphabetic character.
Include: <ctype.h>
Prototype: int islower (int c);
Argument: c character to test
Return Value: Returns a non-zero integer value if the character is a lower case alphabetic character; otherwise, returns zero.
Remarks: A character is considered to be a lower case alphabetic character if it is in the range of 'a'- 'z'.
Example: #include <ctype.h> /* for islower */
          #include <stdio.h> /* for printf */

          int main(void)
          {
              int ch;

              ch = 'B';
              if (islower(ch))
                  printf("B is lower case\n");
              else
                  printf("B is NOT lower case\n");

              ch = 'b';
              if (islower(ch))
                  printf("b is lower case\n");
              else
                  printf("b is NOT lower case\n");
          }

Output:
B is NOT lower case
b is lower case
### isprint

**Description:** Test for a printable character (includes a space).

**Include:** 
<ctype.h>

**Prototype:**
int isprint (int c);

**Argument:**
c character to test

**Return Value:** Returns a non-zero integer value if the character is printable; otherwise, returns zero.

**Remarks:** A character is considered to be a printable character if it is in the range 0x20 to 0x7e inclusive.

**Example:**
```c
#include <ctype.h> /* for isprint */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;
    ch = '&';
    if (isprint(ch))
        printf("& is a printable character\n");
    else
        printf("& is NOT a printable character\n");

    ch = '\t';
    if (isprint(ch))
        printf("a tab is a printable character\n");
    else
        printf("a tab is NOT a printable character\n");
}
```

**Output:**
& is a printable character
a tab is NOT a printable character

### ispunct

**Description:** Test for a punctuation character.

**Include:** 
<ctype.h>

**Prototype:**
int ispunct (int c);

**Argument:**
c character to test

**Return Value:** Returns a non-zero integer value if the character is a punctuation character; otherwise, returns zero.

**Remarks:** A character is considered to be a punctuation character if it is a printable character which is neither a space nor an alphanumeric character. Punctuation characters consist of the following:

! " # $ % & ' ( ) ; < = > ? @ \ [ ] ^ _ { } ~
Example:  

```c
#include <ctype.h> /* for ispunct */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;
    ch = '&';
    if (ispunct(ch))
        printf("& is a punctuation character\n");
    else
        printf("& is NOT a punctuation character\n");
    ch = '\t';
    if (ispunct(ch))
        printf("a tab is a punctuation character\n");
    else
        printf("a tab is NOT a punctuation character\n");
}
```

Output:

```
& is a punctuation character
a tab is NOT a punctuation character
```

### isspace

**Description:** Test for a white-space character.

**Include:** `<ctype.h>`

**Prototype:** `int isspace (int c);`

**Argument:** `c` character to test

**Return Value:** Returns a non-zero integer value if the character is a white-space character; otherwise, returns zero.

**Remarks:** A character is considered to be a white-space character if it is one of the following: space (' '), form feed ('\f'), newline ('\n'), carriage return ('\r'), horizontal tab ('\t'), or vertical tab ('\v').

**Example:**

```c
#include <ctype.h> /* for isspace */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;
    ch = '&';
    if (isspace(ch))
        printf("& is a white-space character\n");
    else
        printf("& is NOT a white-space character\n");
    ch = '\t';
    if (isspace(ch))
        printf("a tab is a white-space character\n");
    else
        printf("a tab is NOT a white-space character\n");
}
```

**Output:**

```
& is NOT a white-space character
a tab is a white-space character
```
**isupper**

**Description:** Test for an upper case letter.

**Include:** `<ctype.h>`

**Prototype:**

```c
int isupper (int c);
```

**Argument:** `c` character to test

**Return Value:** Returns a non-zero integer value if the character is an upper case alphabetic character; otherwise, returns zero.

**Remarks:** A character is considered to be an upper case alphabetic character if it is in the range of 'A'-'Z'.

**Example:**

```c
#include <ctype.h> /* for isupper */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;

    ch = 'B';
    if (isupper(ch))
        printf("B is upper case\n");
    else
        printf("B is NOT upper case\n");

    ch = 'b';
    if (isupper(ch))
        printf("b is upper case\n");
    else
        printf("b is NOT upper case\n");
}
```

**Output:**

```
B is upper case
b is NOT upper case
```

**isxdigit**

**Description:** Test for a hexadecimal digit.

**Include:** `<ctype.h>`

**Prototype:**

```c
int isxdigit (int c);
```

**Argument:** `c` character to test

**Return Value:** Returns a non-zero integer value if the character is a hexadecimal digit; otherwise, returns zero.

**Remarks:** A character is considered to be a hexadecimal digit character if it is in the range of '0'-'9', 'A'-'F', or 'a'-'f'. Note: The list does not include the leading 0x because 0x is the prefix for a hexadecimal number but is not an actual hexadecimal digit.
Example:

```c
#include <ctype.h> /* for isxdigit */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;
    ch = 'B';
    if (isxdigit(ch))
        printf("B is a hexadecimal digit\n");
    else
        printf("B is NOT a hexadecimal digit\n");
    ch = 't';
    if (isxdigit(ch))
        printf("t is a hexadecimal digit\n");
    else
        printf("t is NOT a hexadecimal digit\n");
}
```

Output:

B is a hexadecimal digit
"t is NOT a hexadecimal digit"

tolower

Description: Convert a character to a lower case alphabetical character.

Include: `<ctype.h>`

Prototype: `int tolower (int c);`

Argument: `c` The character to convert to lower case.

Return Value: Returns the corresponding lower case alphabetical character if the argument was originally upper case; otherwise, returns the original character.

Remarks: Only upper case alphabetical characters may be converted to lower case.

Example:

```c
#include <ctype.h> /* for tolower */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;
    ch = 'B';
    printf("B changes to lower case %c\n",
           tolower(ch));
    ch = 'b';
    printf("b remains lower case %c\n",
           tolower(ch));
    ch = '@';
    printf("@ has no lower case, ");
    printf("so %c is returned\n", tolower(ch));
}
```

Output:

B changes to lower case b
"b remains lower case b
@ has no lower case, so @ is returned

isxdigit (Continued)
### toupper

**Description:** Convert a character to an upper case alphabetical character.

**Include:** `<ctype.h>`

**Prototype:**

```c
int toupper (int c);
```

**Argument:**

- `c` The character to convert to upper case.

**Return Value:** Returns the corresponding upper case alphabetical character if the argument was originally lower case; otherwise, returns the original character.

**Remarks:** Only lower case alphabetical characters may be converted to upper case.

**Example:**

```c
#include <ctype.h> /* for toupper */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;

    ch = 'b';
    printf("b changes to upper case %c\n", toupper(ch));

    ch = 'B';
    printf("B remains upper case %c\n", toupper(ch));

    ch = '@';
    printf("@ has no upper case, ");
    printf("so %c is returned\n", toupper(ch));
}
```

**Output:**

b changes to upper case B
B remains upper case B
@ has no upper case, so @ is returned
4.5  <ERRNO.H> ERRORS

The header file `errno.h` consists of macros that provide error codes that are reported by certain library functions (see individual functions). The variable `errno` may return any value greater than zero. To test if a library function encounters an error, the program should store the value zero in `errno` immediately before calling the library function. The value should be checked before another function call could change the value. At program start-up, `errno` is zero. Library functions will never set `errno` to zero.

<table>
<thead>
<tr>
<th>EDOM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Include:</strong></td>
</tr>
<tr>
<td><strong>Remarks:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ERANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Include:</strong></td>
</tr>
<tr>
<td><strong>Remarks:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>errno</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Include:</strong></td>
</tr>
<tr>
<td><strong>Remarks:</strong></td>
</tr>
</tbody>
</table>
4.6  <FLOAT.H> FLOATING-POINT CHARACTERISTICS

The header file float.h consists of macros that specify various properties of floating-point types. These properties include number of significant figures, size limits, and what rounding mode is used.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
<th>Include</th>
<th>Value</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBL_DIG</td>
<td>Number of decimal digits of precision in a double precision floating-point value</td>
<td>&lt;float.h&gt;</td>
<td>6 by default, 15 if the switch -fno-short-double is used</td>
<td>By default, a double type is the same size as a float type (32-bit representation). The -fno-short-double switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.</td>
</tr>
<tr>
<td>DBL_EPSILON</td>
<td>The difference between 1.0 and the next larger representable double precision floating-point value</td>
<td>&lt;float.h&gt;</td>
<td>1.192093e-07 by default, 2.220446e-16 if the switch -fno-short-double is used</td>
<td>By default, a double type is the same size as a float type (32-bit representation). The -fno-short-double switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.</td>
</tr>
<tr>
<td>DBL_MANT_DIG</td>
<td>Number of base-FLT_RADIX digits in a double precision floating-point significand</td>
<td>&lt;float.h&gt;</td>
<td>24 by default, 53 if the switch -fno-short-double is used</td>
<td>By default, a double type is the same size as a float type (32-bit representation). The -fno-short-double switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.</td>
</tr>
<tr>
<td>DBL_MAX</td>
<td>Maximum finite double precision floating-point value</td>
<td>&lt;float.h&gt;</td>
<td>3.402823e+38 by default, 1.797693e+308 if the switch -fno-short-double is used</td>
<td>By default, a double type is the same size as a float type (32-bit representation). The -fno-short-double switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.</td>
</tr>
</tbody>
</table>
### DBL_MAX_10_EXP

**Description:** Maximum integer value for a double precision floating-point exponent in base 10

**Include:** `<float.h>`

**Value:** 38 by default, 308 if the switch `-fno-short-double` is used

**Remarks:** By default, a double type is the same size as a float type (32-bit representation). The `-fno-short-double` switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.

### DBL_MAX_EXP

**Description:** Maximum integer value for a double precision floating-point exponent in base FLT_RADIX

**Include:** `<float.h>`

**Value:** 128 by default, 1024 if the switch `-fno-short-double` is used

**Remarks:** By default, a double type is the same size as a float type (32-bit representation). The `-fno-short-double` switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.

### DBL_MIN

**Description:** Minimum double precision floating-point value

**Include:** `<float.h>`

**Value:** 1.175494e-38 by default, 2.225074e-308 if the switch `-fno-short-double` is used

**Remarks:** By default, a double type is the same size as a float type (32-bit representation). The `-fno-short-double` switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.

### DBL_MIN_10_EXP

**Description:** Minimum negative integer value for a double precision floating-point exponent in base 10

**Include:** `<float.h>`

**Value:** -37 by default, -307 if the switch `-fno-short-double` is used

**Remarks:** By default, a double type is the same size as a float type (32-bit representation). The `-fno-short-double` switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Include</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBL_MIN_EXP</td>
<td>Minimum negative integer value for a double precision floating-point exponent in base FLT_RADIX</td>
<td><code>&lt;float.h&gt;</code></td>
<td>-125 by default, -1021 if the switch <code>-fno-short-double</code> is used</td>
</tr>
<tr>
<td>FLT_DIG</td>
<td>Number of decimal digits of precision in a single precision floating-point value</td>
<td><code>&lt;float.h&gt;</code></td>
<td>6</td>
</tr>
<tr>
<td>FLT_EPSILON</td>
<td>The difference between 1.0 and the next larger representable single precision floating-point value</td>
<td><code>&lt;float.h&gt;</code></td>
<td>1.192093e-07</td>
</tr>
<tr>
<td>FLT_MANT_DIG</td>
<td>Number of base-FLT_RADIX digits in a single precision floating-point significand</td>
<td><code>&lt;float.h&gt;</code></td>
<td>24</td>
</tr>
<tr>
<td>FLT_MAX</td>
<td>Maximum finite single precision floating-point value</td>
<td><code>&lt;float.h&gt;</code></td>
<td>3.402823e+38</td>
</tr>
<tr>
<td>FLT_MAX_10_EXP</td>
<td>Maximum integer value for a single precision floating-point exponent in base 10</td>
<td><code>&lt;float.h&gt;</code></td>
<td>38</td>
</tr>
</tbody>
</table>
### FLT_MAX_EXP

**Description:** Maximum integer value for a single precision floating-point exponent in base FLT_RADIX  
**Include:** `<float.h>`  
**Value:** 128

### FLT_MIN

**Description:** Minimum single precision floating-point value  
**Include:** `<float.h>`  
**Value:** 1.175494e-38

### FLT_MIN_10_EXP

**Description:** Minimum negative integer value for a single precision floating-point exponent in base 10  
**Include:** `<float.h>`  
**Value:** -37

### FLT_MIN_EXP

**Description:** Minimum negative integer value for a single precision floating-point exponent in base FLT_RADIX  
**Include:** `<float.h>`  
**Value:** -125

### FLT_RADIX

**Description:** Radix of exponent representation  
**Include:** `<float.h>`  
**Value:** 2  
**Remarks:** The base representation of the exponent is base-2 or binary.

### FLT_ROUNDS

**Description:** Represents the rounding mode for floating-point operations  
**Include:** `<float.h>`  
**Value:** 1  
**Remarks:** Rounds to the nearest representable value

### LDBL_DIG

**Description:** Number of decimal digits of precision in a long double precision floating-point value  
**Include:** `<float.h>`  
**Value:** 15
### LDBL_EPSILON
**Description:** The difference between 1.0 and the next larger representable long double precision floating-point value
**Include:** `<float.h>`
**Value:** 2.220446e-16

### LDBL_MANT_DIG
**Description:** Number of base-FLT_RADIX digits in a long double precision floating-point significand
**Include:** `<float.h>`
**Value:** 53

### LDBL_MAX
**Description:** Maximum finite long double precision floating-point value
**Include:** `<float.h>`
**Value:** 1.797693e+308

### LDBL_MAX_10_EXP
**Description:** Maximum integer value for a long double precision floating-point exponent in base 10
**Include:** `<float.h>`
**Value:** 308

### LDBL_MAX_EXP
**Description:** Maximum integer value for a long double precision floating-point exponent in base FLT_RADIX
**Include:** `<float.h>`
**Value:** 1024

### LDBL_MIN
**Description:** Minimum long double precision floating-point value
**Include:** `<float.h>`
**Value:** 2.225074e-308

### LDBL_MIN_10_EXP
**Description:** Minimum negative integer value for a long double precision floating-point exponent in base 10
**Include:** `<float.h>`
**Value:** -307
### 4.7 `<LIMITS.H>` IMPLEMENTATION-DEFINED LIMITS

The header file `limits.h` consists of macros that define the minimum and maximum values of integer types. Each of these macros can be used in `#if` preprocessing directives.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Include</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CHAR_BIT</strong></td>
<td>Number of bits to represent type <code>char</code></td>
<td><code>&lt;limits.h&gt;</code></td>
<td>8</td>
</tr>
<tr>
<td><strong>CHAR_MAX</strong></td>
<td>Maximum value of a <code>char</code></td>
<td><code>&lt;limits.h&gt;</code></td>
<td>127</td>
</tr>
<tr>
<td><strong>CHAR_MIN</strong></td>
<td>Minimum value of a <code>char</code></td>
<td><code>&lt;limits.h&gt;</code></td>
<td>-128</td>
</tr>
<tr>
<td><strong>INT_MAX</strong></td>
<td>Maximum value of an <code>int</code></td>
<td><code>&lt;limits.h&gt;</code></td>
<td>32767</td>
</tr>
<tr>
<td><strong>INT_MIN</strong></td>
<td>Minimum value of an <code>int</code></td>
<td><code>&lt;limits.h&gt;</code></td>
<td>-32768</td>
</tr>
<tr>
<td><strong>LDBL_MIN_EXP</strong></td>
<td>Minimum negative integer value for a long double precision floating-point exponent in base <code>FLT_RADIX</code></td>
<td><code>&lt;float.h&gt;</code></td>
<td>-1021</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
<td>Include</td>
<td>Value</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------</td>
<td>-------------</td>
<td>------------------</td>
</tr>
<tr>
<td>LLONG_MIN</td>
<td>Minimum value of a long long int</td>
<td>&lt;limits.h&gt;</td>
<td>-922372036854775808</td>
</tr>
<tr>
<td>LONG_MAX</td>
<td>Maximum value of a long int</td>
<td>&lt;limits.h&gt;</td>
<td>2147483647</td>
</tr>
<tr>
<td>LONG_MIN</td>
<td>Minimum value of a long int</td>
<td>&lt;limits.h&gt;</td>
<td>-2147483648</td>
</tr>
<tr>
<td>MB_LEN_MAX</td>
<td>Maximum number of bytes in a multibyte character</td>
<td>&lt;limits.h&gt;</td>
<td>1</td>
</tr>
<tr>
<td>SCHAR_MAX</td>
<td>Maximum value of a signed char</td>
<td>&lt;limits.h&gt;</td>
<td>127</td>
</tr>
<tr>
<td>SCHAR_MIN</td>
<td>Minimum value of a signed char</td>
<td>&lt;limits.h&gt;</td>
<td>-128</td>
</tr>
<tr>
<td>SHRT_MAX</td>
<td>Maximum value of a short int</td>
<td>&lt;limits.h&gt;</td>
<td>32767</td>
</tr>
<tr>
<td>SHRT_MIN</td>
<td>Minimum value of a short int</td>
<td>&lt;limits.h&gt;</td>
<td>-32768</td>
</tr>
</tbody>
</table>
4.8 `<LOCALE.H>` LOCALIZATION

This compiler defaults to the C locale and does not support any other locales; therefore it does not support the header file `locale.h`. The following would normally be found in this file:

- `struct lconv`
- `NULL`
- `LC_ALL`
- `LC_COLLATE`
- `LC_CTYPE`
- `LC_MONETARY`
- `LC_NUMERIC`
- `LC_TIME`
- `localeconv`
- `setlocale`
## 4.9  `<SETJMP.H>` NON-LOCAL JUMPS

The header file `setjmp.h` consists of a type, a macro and a function that allow control transfers to occur that bypass the normal function call and return process.

### jmp_buf

<table>
<thead>
<tr>
<th>Description</th>
<th>A type that is an array used by <code>setjmp</code> and <code>longjmp</code> to save and restore the program environment.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td><code>&lt;setjmp.h&gt;</code></td>
</tr>
<tr>
<td>Prototype</td>
<td><code>typedef int jmp_buf[_NSETJMP];</code></td>
</tr>
<tr>
<td>Remarks</td>
<td><code>_NSETJMP</code> is defined as 16 + 2 that represents 16 registers and a 32-bit return address.</td>
</tr>
</tbody>
</table>

### setjmp

<table>
<thead>
<tr>
<th>Description</th>
<th>A macro that saves the current state of the program for later use by <code>longjmp</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td><code>&lt;setjmp.h&gt;</code></td>
</tr>
<tr>
<td>Prototype</td>
<td><code>#define setjmp(jmp_buf env)</code></td>
</tr>
<tr>
<td>Argument</td>
<td><code>env</code> variable where environment is stored</td>
</tr>
<tr>
<td>Return Value</td>
<td>If the return is from a direct call, <code>setjmp</code> returns zero. If the return is from a call to <code>longjmp</code>, <code>setjmp</code> returns a non-zero value. <strong>Note:</strong> If the argument <code>val</code> from <code>longjmp</code> is 0, <code>setjmp</code> returns 1.</td>
</tr>
<tr>
<td>Example</td>
<td>See <code>longjmp</code>.</td>
</tr>
</tbody>
</table>

### longjmp

<table>
<thead>
<tr>
<th>Description</th>
<th>A function that restores the environment saved by <code>setjmp</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td><code>&lt;setjmp.h&gt;</code></td>
</tr>
<tr>
<td>Prototype</td>
<td><code>void longjmp(jmp_buf env, int val);</code></td>
</tr>
<tr>
<td>Arguments</td>
<td><code>env</code> variable where environment is stored</td>
</tr>
<tr>
<td></td>
<td><code>val</code> value to be returned to <code>setjmp</code> call.</td>
</tr>
<tr>
<td>Remarks</td>
<td>The value parameter <code>val</code> should be non-zero. If <code>longjmp</code> is invoked from a nested signal handler (that is, invoked as a result of a signal raised during the handling of another signal), the behavior is undefined.</td>
</tr>
</tbody>
</table>
4.10  <SIGNAL.H> SIGNAL HANDLING

The header file signal.h consists of a type, several macros and two functions that specify how the program handles signals while it is executing. A signal is a condition that may be reported during the program execution. Signals are synchronous, occurring under software control via the raise function.

A signal may be handled by:

- Default handling (SIG_DFL); the signal is treated as a fatal error and execution stops
- Ignoring the signal (SIG_IGN); the signal is ignored and control is returned to the user application
- Handling the signal with a function designated via signal.

By default all signals are handled by the default handler, which is identified by SIG_DFL.

The type sig_atomic_t is an integer type that the program access atomically. When this type is used with the keyword volatile, the signal handler can share the data objects with the rest of the program.

<table>
<thead>
<tr>
<th>sig_atomic_t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description: A type used by a signal handler</td>
</tr>
<tr>
<td>Include:     &lt;signal.h&gt;</td>
</tr>
<tr>
<td>Prototype:   typedef int sig_atomic_t;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SIG_DFL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description: Used as the second argument and/or the return value for signal to specify that the default handler should be used for a specific signal.</td>
</tr>
<tr>
<td>Include: &lt;signal.h&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SIG_ERR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description: Used as the return value for signal when it cannot complete a request due to an error.</td>
</tr>
<tr>
<td>Include: &lt;signal.h&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SIG_IGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description: Used as the second argument and/or the return value for signal to specify that the signal should be ignored.</td>
</tr>
<tr>
<td>Include: &lt;signal.h&gt;</td>
</tr>
</tbody>
</table>
SIGABRT

Description: Name for the abnormal termination signal.
Include: <signal.h>
Prototype: #define SIGABRT
Remarks: SIGABRT represents an abnormal termination signal and is used in conjunction with raise or signal. The default raise behavior (action identified by SIG_DFL) is to output to the standard error stream:

    abort - terminating

See the example accompanying signal to see general usage of signal names and signal handling.
Example:

    #include <signal.h> /* for raise, SIGABRT */
    #include <stdio.h> /* for printf */

    int main(void)
    {
        raise(SIGABRT);
        printf("Program never reaches here.");
    }

    Output:
    ABRT

    Explanation:
    ABRT stands for "abort".

SIGFPE

Description: Signals floating-point error such as for division by zero or result out of range.
Include: <signal.h>
Prototype: #define SIGFPE
Remarks: SIGFPE is used as an argument for raise and/or signal. When used, the default behavior is to print an arithmetic error message and terminate the calling program. This may be overridden by a user function that defines the signal handler actions. See signal for an example of a user defined function.
Example:

    #include <signal.h> /* for raise, SIGFPE */
    #include <stdio.h> /* for printf */

    int main(void)
    {
        raise(SIGFPE);
        printf("Program never reaches here");
    }

    Output:
    FPE

    Explanation:
    FPE stands for "floating-point error".
SIGILL

Description: Signals illegal instruction.
Include: <signal.h>
Prototype: #define SIGILL
Remarks: SIGILL is used as an argument for raise and/or signal. When used, the default behavior is to print an invalid executable code message and terminate the calling program. This may be overridden by a user function that defines the signal handler actions. See signal for an example of a user defined function.
Example: #include <signal.h> /* for raise, SIGILL */
#include <stdio.h> /* for printf */

int main(void)
{
    raise(SIGILL);
    printf("Program never reaches here");
}

Output: ILL
Explanation: ILL stands for “illegal instruction”.

SIGINT

Description: Interrupt signal.
Include: <signal.h>
Prototype: #define SIGINT
Remarks: SIGINT is used as an argument for raise and/or signal. When used, the default behavior is to print an interruption message and terminate the calling program. This may be overridden by a user function that defines the signal handler actions. See signal for an example of a user defined function.
Example: #include <signal.h> /* for raise, SIGINT */
#include <stdio.h> /* for printf */

int main(void)
{
    raise(SIGINT);
    printf("Program never reaches here.");
}

Output: INT
Explanation: INT stands for “interruption”.

© 2005 Microchip Technology Inc.
### SIGSEGV

**Description:** Signals invalid access to storage.

**Include:**

```c
#include <signal.h>
```

**Prototype:**

```c
#define SIGSEGV
```

**Remarks:** SIGSEGV is used as an argument for `raise` and/or `signal`. When used, the default behavior is to print an invalid storage request message and terminate the calling program. This may be overridden by a user function that defines the signal handler actions. See `signal` for an example of a user defined function.

**Example:**

```c
#include <signal.h> /* for raise, SIGSEGV */
#include <stdio.h> /* for printf */

int main(void)
{
    raise(SIGSEGV);
    printf("Program never reaches here.");
}
```

**Output:**

```
SIGSEGV
```

**Explanation:**

SIGSEGV stands for “invalid storage access”.

### SIGTERM

**Description:** Signals a termination request

**Include:**

```c
#include <signal.h>
```

**Prototype:**

```c
#define SIGTERM
```

**Remarks:** SIGTERM is used as an argument for `raise` and/or `signal`. When used, the default behavior is to print a termination request message and terminate the calling program. This may be overridden by a user function that defines the signal handler actions. See `signal` for an example of a user defined function.

**Example:**

```c
#include <signal.h> /* for raise, SIGTERM */
#include <stdio.h> /* for printf */

int main(void)
{
    raise(SIGTERM);
    printf("Program never reaches here.");
}
```

**Output:**

```
SIGTERM
```

**Explanation:**

SIGTERM stands for “termination request”.
### raise

**Description:**
Reports a synchronous signal.

**Include:**
`<signal.h>`

**Prototype:**
`int raise(int sig);`

**Argument:**
`sig` signal name

**Return Value:**
Returns a 0 if successful; otherwise, returns a non-zero value.

**Remarks:**
`raise` sends the signal identified by `sig` to the executing program.

**Example:**
```c
#include <signal.h>   /* for raise, signal, */
                    /* SIGILL, SIG_DFL */
#include <stdlib.h>   /* for div, div_t */
#include <stdio.h>    /* for printf */
#include <p30f6014.h> /* for INTCON1bits */

void __attribute__((__interrupt__))(__interrupt__)
_MathError(void)
{
    raise(SIGILL);
    INTCON1bits.MATHERR = 0;
}

void illegalinsn(int idsig)
{
    printf("Illegal instruction executed\n");
    exit(1);
}

int main(void)
{
    int x, y;
    div_t z;

    signal(SIGILL, illegalinsn);
    x = 7;
    y = 0;
    z = div(x, y);
    printf("Program never reaches here");
}
```

**Output:**
Illegal instruction executed

**Explanation:**
This example requires the linker script p30f6014.gld. There are three parts to this example.

First, an interrupt handler is written for the interrupt vector `_MathError` to handle a math error by sending an illegal instruction signal (SIGILL) to the executing program. The last statement in the interrupt handler clears the exception flag.

Second, the function `illegalinsn` will print an error message and call `exit`.

Third, in `main`, `signal (SIGILL, illegalinsn)` sets the handler for SIGILL to the function `illegalinsn`.

When a math error occurs, due to a divide by zero, the _MathError interrupt vector is called, which in turn will raise a signal that will call the handler function for SIGILL, which is the function `illegalinsn`. Thus error messages are printed and the program is terminated.
**signal**

**Description:** Controls interrupt signal handling.

**Include:**

```
#include <signal.h>
```

**Prototype:**

```
void (*signal(int sig, void(*func)(int)))(int);
```

**Arguments:**

- `sig` signal name
- `func` function to be executed

**Return Value:** Returns the previous value of `func`.

**Example:**

```c
#include <signal.h> /* for signal, raise, */
#include <stdio.h>  /* for printf */

/* Signal handler function */
void mysigint(int id)
{
    printf("SIGINT received\n");
}

int main(void)
{
    /* Override default with user defined function */
    signal(SIGINT, mysigint);
    raise(SIGINT);

    /* Ignore signal handler */
    signal(SIGILL, SIG_IGN);
    raise(SIGILL);
    printf("SIGILL was ignored\n");

    /* Use default signal handler */
    raise(SIGFPE);
    printf("Program never reaches here.\n");
}
```

**Output:**

SIGINT received
SIGILL was ignored
FPE

**Explanation:**

The function `mysigint` is the user-defined signal handler for `SIGINT`. Inside the main program, the function `signal` is called to set up the signal handler (`mysigint`) for the signal `SIGINT` that will override the default actions. The function `raise` is called to report the signal `SIGINT`. This causes the signal handler for `SIGINT` to use the user-defined function (`mysigint`) as the signal handler so it prints the "SIGINT received" message.

Next, the function `signal` is called to set up the signal handler `SIG_IGN` for the signal `SIGILL`. The constant `SIG_IGN` is used to indicate the signal should be ignored. The function `raise` is called to report the signal `SIGILL` that is ignored.

The function `raise` is called again to report the signal `SIGFPE`. Since there is no user defined function for `SIGFPE`, the default signal handler is used so the message "FPE" is printed (which stands for "arithmetic error - terminating"). Then the calling program is terminated. The `printf` statement is never reached.
4.11 `<STDARG.H>` VARIABLE ARGUMENT LISTS

The header file `stdarg.h` supports functions with variable argument lists. This allows functions to have arguments without corresponding parameter declarations. There must be at least one named argument. The variable arguments are represented by ellipses (...). An object of type `va_list` must be declared inside the function to hold the arguments. `va_start` will initialize the variable to an argument list, `va_arg` will access the argument list, and `va_end` will end the use of the argument.

### va_list

**Description:** The type `va_list` declares a variable that will refer to each argument in a variable-length argument list.

**Include:** `<stdarg.h>`

**Example:** See `va_arg`.

### va_arg

**Description:** Gets the current argument

**Include:** `<stdarg.h>`

**Prototype:**

```c
#define va_arg(va_list ap, Ty)
```

**Argument:**

- `ap` pointer to list of arguments
- `Ty` type of argument to be retrieved

**Return Value:** Returns the current argument

**Remarks:** `va_start` must be called before `va_arg`.

**Example:**

```c
#include <stdio.h>   /* for printf */
#include <stdarg.h>  /* for va_arg, va_start, va_list, va_end */

void tprint(const char *fmt, ...)
{
  va_list ap;

  va_start(ap, fmt);
  while (*fmt)
  { switch (*fmt)
```

```c
  
```
va_arg (Continued)

case '%':
    fmt++;
    if (*fmt == 'd')
    {
        int d = va_arg(ap, int);
        printf("<%d> is an integer\n", d);
    }
    else if (*fmt == 's')
    {
        char *s = va_arg(ap, char*);
        printf("<%s> is a string\n", s);
    }
    else
    {
        printf("%c is an unknown format\n", *fmt);
    }
    fmt++;
    break;
default:
    printf("%c is unknown\n", *fmt);
    fmt++;
    break;
}
va_end(ap);
}

int main(void)
{
    tprint("%d%s.%c", 83, "This is text.", 'a');
}

Output:
<83> is an integer
<This is text.> is a string
. is unknown
%c is an unknown format
4.12 <STDDEF.H> COMMON DEFINITIONS

The header file stddef.h consists of several types and macros that are of general use in programs.

`ptrdiff_t`
- Description: The type of the result of subtracting two pointers.
- Include: <stddef.h>

`size_t`
- Description: The type of the result of the sizeof operator.
- Include: <stddef.h>

`wchar_t`
- Description: A type that holds a wide character value.
- Include: <stddef.h>

`NULL`
- Description: The value of a null pointer constant.
- Include: <stddef.h>
### offsetof

**Description:**
Gives the offset of a structure member from the beginning of the structure.

**Include:**
`<stddef.h>`

**Prototype:**
```
#define offsetof(T, mbr) ...
```

**Arguments:**
- `T`  name of structure
- `mbr`  name of member in structure `T`

**Return Value:**
Returns the offset in bytes of the specified member (`mbr`) from the beginning of the structure.

**Remarks:**
The macro `offsetof` is undefined for bitfields. An error message will occur if bitfields are used.

**Example:**
```
#include <stddef.h> /* for offsetof */
#include <stdio.h>  /* for printf */

struct info {
    char item1[5];
    int item2;
    char item3;
    float item4;
};

int main(void)
{
    printf("Offset of item1 = %d\n", 
            offsetof(struct info,item1));
    printf("Offset of item2 = %d\n", 
            offsetof(struct info,item2));
    printf("Offset of item3 = %d\n", 
            offsetof(struct info,item3));
    printf("Offset of item4 = %d\n", 
            offsetof(struct info,item4));
}
```

**Output:**
- Offset of item1 = 0
- Offset of item2 = 6
- Offset of item3 = 8
- Offset of item4 = 10

**Explanation:**
This program shows the offset in bytes of each structure member from the start of the structure. Although `item1` is only 5 bytes (`char item1[5]`), padding is added so the address of `item2` falls on an even boundary. The same occurs with `item3`; it is 1 byte (`char item3`) with 1 byte of padding.
4.13 <STDIO.H> INPUT AND OUTPUT

The header file stdio.h consists of types, macros and functions that provide support to perform input and output operations on files and streams. When a file is opened it is associated with a stream. A stream is a pipeline for the flow of data into and out of files. Because different systems use different properties, the stream provides more uniform properties to allow reading and writing of the files.

Streams can be text streams or binary streams. Text streams consist of a sequence of characters divided into lines. Each line is terminated with a newline (\n) character. The characters may be altered in their internal representation, particularly in regards to line endings. Binary streams consist of sequences of bytes of information. The bytes transmitted to the binary stream are not altered. There is no concept of lines, the file is just a series of bytes.

At start-up three streams are automatically opened: stdin, stdout, and stderr. stdin provides a stream for standard input, stdout is standard output and stderr is the standard error. Additional streams may be created with the fopen function. See fopen for the different types of file access that are permitted. These access types are used by fopen and freopen.

The type FILE is used to store information about each opened file stream. It includes such things as error indicators, end-of-file indicators, file position indicators, and other internal status information needed to control a stream. Many functions in the stdio use FILE as an argument.

There are three types of buffering: unbuffered, line buffered and fully buffered. Unbuffered means a character or byte is transferred one at a time. Line buffered collects and transfers an entire line at a time (i.e., the newline character indicates the end of a line). Fully buffered allows blocks of an arbitrary size to be transmitted. The functions setbuf and setvbuf control file buffering.

The stdio.h file also contains functions that use input and output formats. The input formats, or scan formats, are used for reading data. Their descriptions can be found under scanf, but they are also used by fscanf and sscanf. The output formats, or print formats, are used for writing data. Their descriptions can be found under printf. These print formats are also used by fprintf, sprintf, vfprintf, vprintf and vsprintf.

Certain compiler options may affect how standard I/O performs. In an effort to provide a more tailored version of the formatted I/O routines, the tool chain may convert a call to a printf or scanf style function to a different call. The options are summarized below:

- The -msmart-io option, when enabled, will attempt to convert printf, scanf and other functions that use the input output formats to an integer only variant. The functionality is the same as that of the C standard forms, minus the support for floating-point output. -msmart-io=0 disables this feature and no conversion will take place. -msmart-io=1 or -msmart-io (the default) will convert a function call if it can be proven that an I/O function will never be presented with a floating-point conversion. -msmart-io=2 is more optimistic than the default and will assume that non-constant format strings or otherwise unknown format strings will not contain a floating-point format. In the event that -msmart-io=2 is used with a floating-point format, the format letter will appear as literal text and its corresponding argument will not be consumed.
- -fno-short-double will cause the compiler to generate calls to formatted I/O routines that support double as if it were a long double type.

Mixing modules compiled with these options may result in a larger executable size, or incorrect execution if large and small double-sized data is shared across modules.
<table>
<thead>
<tr>
<th>Description</th>
<th>Include</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>Stores information for a file stream.</td>
</tr>
<tr>
<td>fpos_t</td>
<td>Type of a variable used to store a file position.</td>
</tr>
<tr>
<td>size_t</td>
<td>The result type of the sizeof operator.</td>
</tr>
<tr>
<td>_IOFBF</td>
<td>Indicates full buffering.</td>
</tr>
<tr>
<td>Remarks:</td>
<td>Used by the function setvbuf.</td>
</tr>
<tr>
<td>_IOLBF</td>
<td>Indicates line buffering.</td>
</tr>
<tr>
<td>Remarks:</td>
<td>Used by the function setvbuf.</td>
</tr>
<tr>
<td>_IONBF</td>
<td>Indicates no buffering.</td>
</tr>
<tr>
<td>Remarks:</td>
<td>Used by the function setvbuf.</td>
</tr>
<tr>
<td>BUFSIZ</td>
<td>Defines the size of the buffer used by the function setbuf.</td>
</tr>
<tr>
<td>Value:</td>
<td>512</td>
</tr>
</tbody>
</table>
### EOF

**Description:** A negative number indicating the end-of-file has been reached or to report an error condition.

**Include:** `<stdio.h>`

**Remarks:** If an end-of-file is encountered, the end-of-file indicator is set. If an error condition is encountered, the error indicator is set. Error conditions include write errors and input or read errors.

### FILENAME_MAX

**Description:** Maximum number of characters in a filename including the null terminator.

**Include:** `<stdio.h>`

**Value:** 260

### FOPEN_MAX

**Description:** Defines the maximum number of files that can be simultaneously open

**Include:** `<stdio.h>`

**Value:** 8

**Remarks:** `stderr, stdin and stdout` are included in the `FOPEN_MAX` count.

### L_tmpnam

**Description:** Defines the number of characters for the longest temporary filename created by the function `tmpnam`.

**Include:** `<stdio.h>`

**Value:** 16

**Remarks:** `L_tmpnam` is used to define the size of the array used by `tmpnam`.

### NULL

**Description:** The value of a null pointer constant

**Include:** `<stdio.h>`

### SEEK_CUR

**Description:** Indicates that `fseek` should seek from the current position of the file pointer

**Include:** `<stdio.h>`

**Example:** See example for `fseek`. 
### SEEK_END

**Description:** Indicates that `fseek` should seek from the end of the file.

**Include:** `<stdio.h>`

**Example:** See example for `fseek`.

### SEEK_SET

**Description:** Indicates that `fseek` should seek from the beginning of the file.

**Include:** `<stdio.h>`

**Example:** See example for `fseek`.

### stderr

**Description:** File pointer to the standard error stream.

**Include:** `<stdio.h>`

### stdin

**Description:** File pointer to the standard input stream.

**Include:** `<stdio.h>`

### stdout

**Description:** File pointer to the standard output stream.

**Include:** `<stdio.h>`

### TMP_MAX

**Description:** The maximum number of unique filenames the function `tmpnam` can generate.

**Include:** `<stdio.h>`

**Value:** 32
clearerr

Description: Resets the error indicator for the stream

Include: <stdio.h>

Prototype: void clearerr(FILE *stream);

Argument: stream stream to reset error indicators

Remarks: The function clears the end-of-file and error indicators for the given stream (i.e., feof and ferror will return false after the function clearerr is called).

Example:
/* This program tries to write to a file that is */
/* readonly. This causes the error indicator to */
/* be set. The function ferror is used to check */
/* the error indicator. The function clearerr is */
/* used to reset the error indicator so the next */
/* time ferror is called it will not report an */
/* error. */
#include <stdio.h> /* for ferror, clearerr, */
/* printf, fprintf, fopen, */
/* fclose, FILE, NULL */

int main(void)
{
    FILE *myfile;

    if ((myfile = fopen("sampclearerr.c", "r")) == NULL)
        printf("Cannot open file\n");
    else
    {
        fprintf(myfile, "Write this line to the "
                   "file.\n");
        if (ferror(myfile))
            printf("Error\n");
        else
            printf("No error\n");
        clearerr(myfile);
        if (ferror(myfile))
            printf("Still has Error\n");
        else
            printf("Error indicator reset\n");

        fclose(myfile);
    }
}

Output:
Error
Error indicator reset
fclose

Description: Close a stream.

Include: `<stdio.h>

Prototype: int fclose(FILE *stream);

Argument: stream pointer to the stream to close

Return Value: Returns 0 if successful; otherwise, returns EOF if any errors were detected.

Remarks: fclose writes any buffered output to the file.

Example: #include <stdio.h> /* for fopen, fclose, printf, FILE, NULL, EOF */

    int main(void)
    {
        FILE *myfile1, *myfile2;
        int y;

        if ((myfile1 = fopen("afile1", "w+")) == NULL)
            printf("Cannot open afile1\n");
        else
        {
            printf("afile1 was opened\n");

            y = fclose(myfile1);
            if (y == EOF)
                printf("afile1 was not closed\n");
            else
                printf("afile1 was closed\n");
        }
    }

Output:
afile1 was opened
afile1 was closed
feof

Description: Tests for end-of-file
Include: <stdio.h>
Prototype: int feof(FILE *stream);
Argument: stream stream to check for end-of-file
Return Value: Returns non-zero if stream is at the end-of-file; otherwise, returns zero.
Example:

```c
#include <stdio.h> /* for feof, fgetc, fputc, */
/* fopen, fclose, FILE, */
/* NULL */

int main(void)
{
    FILE *myfile;
    int y = 0;

    if( (myfile = fopen( "afile.txt", "rb" )) == NULL )
        printf( "Cannot open file\n" );
    else
    {
        for (;;)
        {
            y = fgetc(myfile);
            if (feof(myfile))
                break;
            fputc(y, stdout);
        }
    fclose( myfile );
}
```

Input: Contents of afile.txt (used as input):
This is a sentence.

Output:
This is a sentence.
ferror

Description: Tests if error indicator is set.

Include: <stdio.h>

Prototype: int ferror(FILE *stream);

Argument: stream pointer to FILE structure

Return Value: Returns a non-zero value if error indicator is set; otherwise, returns a zero.

Example:
/* This program tries to write to a file that is */
/* readonly. This causes the error indicator to */
/* be set. The function ferror is used to check */
/* the error indicator and find the error. The */
/* function clearerr is used to reset the error */
/* indicator so the next time ferror is called */
/* it will not report an error. */

#include <stdio.h> /* for ferror, clearerr, */
    /* printf, fprintf, */
    /* fopen, fclose, */
    /* FILE, NULL */

int main(void)
{
    FILE *myfile;

    if ((myfile = fopen("sampclearerr.c", "r")) ==
        NULL)
        printf("Cannot open file\n");
    else
        {
            fprintf(myfile, "Write this line to the "
                    "file.\n");
            if (ferror(myfile))
                printf("Error\n");
            else
                printf("No error\n");
            clearerr(myfile);
            if (ferror(myfile))
                printf("Still has Error\n");
            else
                printf("Error indicator reset\n");

            fclose(myfile);
        }
}

Output:
Error
Error indicator reset
flush
Description: Flushed the buffer in the specified stream.
Include: <stdio.h>
Prototype: int fflush(FILE *stream);
Argument: stream pointer to the stream to flush.
Return Value: Returns EOF if a write error occurs; otherwise, returns zero for success.
Remarks: If stream is a null pointer, all output buffers are written to files. fflush has no effect on an unbuffered stream.

fgets
Description: Get a character from a stream
Include: <stdio.h>
Prototype: int fgetc(FILE *stream);
Argument: stream pointer to the open stream
Return Value: Returns the character read or EOF if a read error occurs or end-of-file is reached.
Remarks: The function reads the next character from the input stream, advances the file-position indicator and returns the character as an unsigned char converted to an int.
Example:
#include <stdio.h> /* for fgetc, printf, */
     /* fclose, FILE, */
     /* NULL, EOF */

    int main(void)
    {
      FILE *buf;
      char y;

      if ((buf = fopen("afile.txt", "r")) == NULL)
        printf("Cannot open afile.txt\n");
      else
        { 
          y = fgetc(buf);
          while (y != EOF)
          {
            printf("%c", y);
            y = fgetc(buf);
          }
          fclose(buf);
        }
    }

Input:
Contents of afile.txt (used as input):
Short Longer string

Output:
S|h|o|r|t|t|
|L|o|n|g|e|r|s|t|r|i|n|g|
fgetpos

Description: Gets the stream’s file position.

Include: <stdio.h>

Prototype: int fgetpos(FILE *stream, fpos_t *pos);

Arguments: stream target stream
pos position-indicator storage

Return Value: Returns 0 if successful; otherwise, returns a non-zero value.

Remarks: The function stores the file-position indicator for the given stream in *pos if successful, otherwise, fgetpos sets errno.

Example:
/* This program opens a file and reads bytes at */
/* several different locations. The fgetpos */
/* function notes the 8th byte. 21 bytes are */
/* read then 18 bytes are read. Next the */
/* fsetpos function is set based on the */
/* fgetpos position and the previous 21 bytes */
/* are reread. */

#include <stdio.h> /* for fgetpos, fread, */
/* printf, fopen, fclose, */
/* FILE, NULL, perror, */
/* fpos_t, sizeof */

int main(void)
{
    FILE *myfile;
    fpos_t pos;
    char buf[25];

    if ((myfile = fopen("sampfgetpos.c", "rb")) == NULL)
        printf("Cannot open file
");
    else
    {
        fread(buf, sizeof(char), 8, myfile);
        if (fgetpos(myfile, &pos) != 0)
            perror("fgetpos error");
        else
        {
            fread(buf, sizeof(char), 21, myfile);
            printf("Bytes read: %.21s
", buf);
            fread(buf, sizeof(char), 18, myfile);
            printf("Bytes read: %.18s
", buf);
        }

        if (fsetpos(myfile, &pos) != 0)
            perror("fsetpos error");

        fread(buf, sizeof(char), 21, myfile);
        printf("Bytes read: %.21s
", buf);
        fclose(myfile);
    }
}

Output:
Bytes read: program opens a file
Bytes read: and reads bytes at
Bytes read: program opens a file
fgets (Get a string from a stream)

**Description:**
Get a string from a stream

**Include:**
<stdio.h>

**Prototype:**
char *fgets(char *s, int n, FILE *stream);

**Arguments:**
- `s`: pointer to the storage string
- `n`: maximum number of characters to read
- `stream`: pointer to the open stream.

**Return Value:**
Returns a pointer to the string `s` if successful; otherwise, returns a null pointer.

**Remarks:**
The function reads characters from the input stream and stores them into the string pointed to by `s` until it has read `n-1` characters, stores a newline character or sets the end-of-file or error indicators. If any characters were stored, a null character is stored immediately after the last read character in the next element of the array. If `fgets` sets the error indicator, the array contents are indeterminate.

**Example:**
```
#include <stdio.h> /* for fgets, printf, */
/* fopen, fclose,     */
/* FILE, NULL         */
#define MAX 50

int main(void)
{
    FILE *buf;
    char s[MAX];

    if ((buf = fopen("afile.txt", "r")!= NULL)
        printf("Cannot open afile.txt\n");
    else
    {
        while (fgets(s, MAX, buf) != NULL)
        {
            printf("%s\n", s);
        }
        fclose(buf);
    }
}
```

**Input:**
Contents of `afile.txt` (used as input):
Short
Longer string

**Output:**
Short
| Longer string
|
fopen

Description: Opens a file.

Include: <stdio.h>

Prototype: FILE *fopen(const char *filename, const char *mode);

Arguments: filename name of the file
mode type of access permitted

Return Value: Returns a pointer to the open stream. If the function fails a null pointer is returned.

Remarks: Following are the types of file access:
r - opens an existing text file for reading
w - opens an empty text file for writing. (An existing file will be overwritten.)
a - opens a text file for appending. (A file is created if it doesn't exist.)
rb - opens an existing binary file for reading.
wb - opens an empty binary file for writing. (An existing file will be overwritten.)
ab - opens a binary file for appending. (A file is created if it doesn't exist.)
r+ - opens an existing text file for reading and writing.
w+ - opens an empty text file for reading and writing. (An existing file will be overwritten.)
a+ - opens a text file for reading and appending. (A file is created if it doesn't exist.)
r+b or rb+ - opens an existing binary file for reading and writing.
w+b or wb+ - opens an empty binary file for reading and writing. (An existing file will be overwritten.)
a+b or ab+ - opens a binary file for reading and appending. (A file is created if it doesn't exist.)

Example: #include <stdio.h> /* for fopen, fclose, */
          /* printf, FILE, */
          /* NULL, EOF */

          int main(void)
          {
              FILE *myfile1, *myfile2;
              int y;
if ((myfile1 = fopen("afile1", "r")) == NULL)
   printf("Cannot open afile1\n");
else
   {
      printf("afile1 was opened\n");
      y = fclose(myfile1);
      if (y == EOF)
         printf("afile1 was not closed\n");
      else
         printf("afile1 was closed\n");
   }
if ((myfile1 = fopen("afile1", "w+")) == NULL)
   printf("Second try, cannot open afile1\n");
else
   {
      printf("Second try, afile1 was opened\n");
      y = fclose(myfile1);
      if (y == EOF)
         printf("afile1 was not closed\n");
      else
         printf("afile1 was closed\n");
   }
if ((myfile2 = fopen("afile2", "w+")) == NULL)
   printf("Cannot open afile2\n");
else
   {
      printf("afile2 was opened\n");
      y = fclose(myfile2);
      if (y == EOF)
         printf("afile2 was not closed\n");
      else
         printf("afile2 was closed\n");
   }
}

Output:
Cannot open afile1
Second try, afile1 was opened
afile1 was closed
afile2 was opened
afile2 was closed

Explanation:
afile1 must exist before it can be opened for reading (r) or the fopen function will fail. If the fopen function opens a file for writing (w+) it does not have to already exist. If it doesn't exist, it will be created and then opened.
### fprintf

**Description:** Prints formatted data to a stream.

**Include:**
```c
<stdio.h>
```

**Prototype:**
```c
int fprintf(FILE *stream, const char *format, ...);
```

**Arguments:**
- `stream` pointer to the stream in which to output data
- `format` format control string
- `...` optional arguments

**Return Value:** Returns number of characters generated or a negative number if an error occurs.

**Remarks:** The format argument has the same syntax and use that it has in `printf`.

**Example:**
```c
#include <stdio.h> /* for fopen, fclose, */
/* fprintf, printf, */
/* FILE, NULL */

int main(void)
{
    FILE *myfile;
    int y;
    char s[]="Print this string";
    int x = 1;
    char a = '\n';

    if ((myfile = fopen("afile", "w")) == NULL)
        printf("Cannot open afile\n");
    else
    {
        y = fprintf(myfile, "%s %d time%c", s, x, a);

        printf("Number of characters printed "
            "to file = %d", y);

        fclose(myfile);
    }
}
```

**Output:**
- Number of characters printed to file = 25
- Contents of `afile`:
  - Print this string 1 time
fputc
Description: Puts a character to the stream.
Include: <stdio.h>
Prototype: int fputc(int c, FILE *stream);
Arguments: c character to be written
stream pointer to the open stream
Return Value: Returns the character written or EOF if a write error occurs.
Remarks: The function writes the character to the output stream, advances the
file-position indicator and returns the character as an unsigned char converted to an int.
Example:
#include <stdio.h> /* for fputc, EOF, stdout */

int main(void)
{
    char *y;
    char buf[] = "This is text\n";
    int x;
    x = 0;

    for (y = buf; (x != EOF) && (*y != '\0'); y++)
    {
        x = fputc(*y, stdout);
        fputc('|', stdout);
    }
}
Output:
T|h|i|s| |i|s| |t|e|x|t|
|
fputs
Description: Puts a string to the stream.
Include: <stdio.h>
Prototype: int fputs(const char *s, FILE *stream);
Arguments: s string to be written
stream pointer to the open stream
Return Value: Returns a non-negative value if successful; otherwise, returns EOF.
Remarks: The function writes characters to the output stream up to but not
including the null character.
Example:
#include <stdio.h> /* for fputs, stdout */

int main(void)
{  
    char buf[] = "This is text\n";

    fputs(buf, stdout);
    fputs("|", stdout);
}
Output:
This is text
fread

**Description:** Reads data from the stream.

**Include:**
```c
#include <stdio.h>
```

**Prototype:**
```c
size_t fread(void *ptr, size_t size, size_t nelem, FILE *stream);
```

**Arguments:**
- `ptr` pointer to the storage buffer
- `size` size of item
- `nelem` maximum number of items to be read
- `stream` pointer to the stream

**Return Value:** Returns the number of complete elements read up to `nelem` whose size is specified by size.

**Remarks:** The function reads characters from a given stream into the buffer pointed to by `ptr` until the function stores `size * nelem` characters or sets the end-of-file or error indicator. `fread` returns `n/size` where `n` is the number of characters it read. If `n` is not a multiple of `size`, the value of the last element is indeterminate. If the function sets the error indicator, the file-position indicator is indeterminate.

**Example:**
```c
#include <stdio.h> /* for fread, fwrite, */
/* printf, fopen, fclose, */
/* sizeof, FILE, NULL */

int main(void)
{
    FILE *buf;
    int x, numwrote, numread;
    double nums[10], readnums[10];

    if ((buf = fopen("afile.out", "w+")) != NULL)
    {
        for (x = 0; x < 10; x++)
        {
            nums[x] = 10.0/(x + 1);
            printf("10.0/%d = %f\n", x+1, nums[x]);
        }

        numwrote = fwrite(nums, sizeof(double),
                          10, buf);
        printf("Wrote %d numbers\n\n", numwrote);
        fclose(buf);
    }
    else
    {
        printf("Cannot open afile.out\n");
    }
}
```
if ((buf = fopen("afile.out", "r+")) != NULL) {
    numread = fread(readnums, sizeof(double),
                    10, buf);
    printf("Read %d numbers
", numread);
    for (x = 0; x < 10; x++)
        { printf("%d * %f = %f
", x+1, readnums[x],
                  (x + 1) * readnums[x]);
        }
    fclose(buf);
} else
    printf("Cannot open afile.out
");

Output:
10.0/1 = 10.000000
10.0/2 = 5.000000
10.0/3 = 3.333333
10.0/4 = 2.500000
10.0/5 = 2.000000
10.0/6 = 1.666667
10.0/7 = 1.428571
10.0/8 = 1.250000
10.0/9 = 1.111111
10.0/10 = 1.000000
Wrote 10 numbers

Read 10 numbers
1 * 10.000000 = 10.000000
2 * 5.000000 = 10.000000
3 * 3.333333 = 10.000000
4 * 2.500000 = 10.000000
5 * 2.000000 = 10.000000
6 * 1.666667 = 10.000000
7 * 1.428571 = 10.000000
8 * 1.250000 = 10.000000
9 * 1.111111 = 10.000000
10 * 1.000000 = 10.000000

Explanation:
This program uses fwrite to save 10 numbers to a file in binary form.
This allows the numbers to be saved in the same pattern of bits as the
program is using which provides more accuracy and consistency. Using
fprintf would save the numbers as text strings which could cause
the numbers to be truncated. Each number is divided into 10 to
produce a variety of numbers. Retrieving the numbers with fread to a
new array and multiplying them by the original number shows the
numbers were not truncated in the save process.
freopen

**Description:** Reassigns an existing stream to a new file.

**Include:** `<stdio.h>`

**Prototype:**
```c
FILE *freopen(const char *filename, const char *mode, FILE *stream);
```

**Arguments:**
- `filename` name of the new file
- `mode` type of access permitted
- `stream` pointer to the currently open stream

**Return Value:** Returns a pointer to the new open file. If the function fails a null pointer is returned.

**Remarks:** The function closes the file associated with the stream as though `fclose` was called. Then it opens the new file as though `fopen` was called. `freopen` will fail if the specified stream is not open. See `fopen` for the possible types of file access.

**Example:**
```c
#include <stdio.h> /* for fopen, freopen, */
/* printf, fclose, */
/* FILE, NULL */

int main(void)
{
    FILE *myfile1, *myfile2;
    int y;

    if ((myfile1 = fopen("afile1", "w+")) == NULL)
        printf("Cannot open afile1\n");
    else
        printf("afile1 was opened\n");

    if ((myfile2 = freopen("afile2", "w+", myfile1)) == NULL)
        {
            printf("Cannot open afile2\n");
            fclose(myfile1);
        }
    else
    {
        printf("afile2 was opened\n");
        fclose(myfile2);
    }
}
```

**Output:**
afile1 was opened
afile2 was opened

**Explanation:**
This program uses `myfile2` to point to the stream when `freopen` is called so if an error occurs, `myfile1` will still point to the stream and can be closed properly. If the `freopen` call is successful, `myfile2` can be used to close the stream properly.

fscanf

**Description:** Scans formatted text from a stream.

**Include:** `<stdio.h>`
Prototype:  int fscanf(FILE *stream, const char *format, ...);

Arguments:  
  stream  pointer to the open stream from which to read data  
  format  format control string  
  ...  optional arguments

Return Value:  Returns the number of items successfully converted and assigned. If no items are assigned, a 0 is returned. EOF is returned if end-of-file is encountered before the first conversion or if an error occurs.

Remarks:  The format argument has the same syntax and use that it has in scanf.

Example:  
```c
#include <stdio.h> /* for fopen, fscanf, fclose, fprintf, fseek, printf, FILE, NULL, SEEK_SET */

int main(void)
{
    FILE *myfile;
    char s[30];
    int x;
    char a;

    if ((myfile = fopen("afile", "w+")) == NULL)
        printf("Cannot open afile\n");
    else
    {
        fprintf(myfile, "%s %d times%c", 
               "Print this string", 100, '\n');

        fseek(myfile, 0L, SEEK_SET);
        fscanf(myfile, "%s", s);
        printf("%s
", s);
        fscanf(myfile, "%s", s);
        printf("%s
", s);
        fscanf(myfile, "%s", s);
        printf("%s
", s);
        fscanf(myfile, "%s", s);
        printf("%s
", s);
        fscanf(myfile, "%d", &x);
        printf("%d
", x);
        fscanf(myfile, "%s", s);
        printf("%s
", s);
        fscanf(myfile, "%s", a);
        printf("%s
", a);

        fclose(myfile);
    }
}
```

Input:  
Contents of afile:
Print this string 100 times

Output:  
Print this string 100 times
**fseek**

**Description:** Moves file pointer to a specific location.

**Include:** `<stdio.h>`

**Prototype:**

```c
int fseek(FILE *stream, long offset, int mode);
```

**Arguments:**

- `stream` stream in which to move the file pointer.
- `offset` value to add to the current position
- `mode` type of seek to perform

**Return Value:** Returns 0 if successful; otherwise, returns a non-zero value and sets `errno`.

**Remarks:**

- `mode` can be one of the following:
  - `SEEK_SET` – seeks from the beginning of the file
  - `SEEK_CUR` – seeks from the current position of the file pointer
  - `SEEK_END` – seeks from the end of the file

**Example:**

```c
#include <stdio.h> /* for fseek, fgets, printf, fopen, fclose, FILE, NULL, perror, SEEK_SET, SEEK_CUR, SEEK_END */

int main(void) {
    FILE *myfile;
    char s[70];
    int y;

    myfile = fopen("afile.out", "w+");
    if (myfile == NULL)  
        printf("Cannot open afile.out\n");
    else
    {
        fprintf(myfile, "This is the beginning, ");
        fprintf(myfile, "this is the middle and ");
        fprintf(myfile, "this is the end." );

        y = fseek(myfile, 0L, SEEK_SET);
        if (y)
            perror("Fseek failed");
        else
        {
            fgets(s, 22, myfile);
            printf("%s\n", s);
        }

        y = fseek(myfile, 2L, SEEK_CUR);
        if (y)
            perror("Fseek failed");
        else
        {
            fgets(s, 70, myfile);
            printf("%s\n", s);
        }
}
```
fseek (Continued)

```c
y = fseek(myfile, -16L, SEEK_END);
if (y)
    perror("Fseek failed");
else
{
    fgets(s, 70, myfile);
    printf("\"%s\"\n", s);
}
fclose(myfile);
}
```

Output:
"This is the beginning"
"this is the middle and this is the end."
"this is the end."

Explanation:
The file `afile.out` is created with the text, “This is the beginning, this is the middle and this is the end”. The function `fseek` uses an offset of zero and `SEEK_SET` to set the file pointer to the beginning of the file. `fgets` then reads 22 characters which are “This is the beginning”, and adds a null character to the string.

Next, `fseek` uses an offset of two and `SEEK_CURRENT` to set the file pointer to the current position plus two (skipping the comma and space). `fgets` then reads up to the next 70 characters. The first 39 characters are “this is the middle and this is the end”. It stops when it reads EOF and adds a null character to the string.

Finally, `fseek` uses an offset of negative 16 characters and `SEEK_END` to set the file pointer to 16 characters from the end of the file. `fgets` then reads up to 70 characters. It stops at the EOF after reading 16 characters “this is the end”. and adds a null character to the string.

---

**fsetpos**

Sets the stream’s file position.

Include: `<stdio.h>`

Prototype: `int fsetpos(FILE *stream, const fpos_t *pos);`

Arguments:
- `stream` target stream
- `pos` position-indicator storage as returned by an earlier call to `fgetpos`

Return Value: Returns 0 if successful; otherwise, returns a non-zero value.

Remarks: The function sets the file-position indicator for the given stream in `*pos` if successful; otherwise, `fsetpos` sets `errno`. 
fsetpos (Continued)

Example: /* This program opens a file and reads bytes at */
/* several different locations. The fgetpos */
/* function notes the 8th byte. 21 bytes are */
/* read then 18 bytes are read. Next the */
/* fsetpos function is set based on the */
/* fgetpos position and the previous 21 bytes */
/* are reread. */

#include <stdio.h> /* for fgetpos, fread, */
/* printf, fopen, fclose, */
/* FILE, NULL, perror, */
/* fpos_t, sizeof */

int main(void)
{
    FILE  *myfile;
    fpos_t pos;
    char   buf[25];

    if ((myfile = fopen("sampfgetpos.c", "rb")) ==
        NULL)
        printf("Cannot open file\n");
    else
    {
        fread(buf, sizeof(char), 8, myfile);
        if (fgetpos(myfile, &pos) != 0)
            perror("fgetpos error");
        else
        {
            fread(buf, sizeof(char), 21, myfile);
            printf("Bytes read: %.21s\n", buf);
            fread(buf, sizeof(char), 18, myfile);
            printf("Bytes read: %.18s\n", buf);
        }

        if (fsetpos(myfile, &pos) != 0)
            perror("fsetpos error");

        fread(buf, sizeof(char), 21, myfile);
        printf("Bytes read: %.21s\n", buf);
        fclose(myfile);
    }
}

Output:
Bytes read: program opens a file
Bytes read: and reads bytes at
Bytes read: program opens a file
**ftell**

Description: Gets the current position of a file pointer.

Include: `<stdio.h>`

Prototype: `long ftell(FILE *stream);`

Argument: `stream` stream in which to get the current file position

Return Value: Returns the position of the file pointer if successful; otherwise, returns -1.

Example:
```
#include <stdio.h> /* for ftell, fread, printf, fprintf, fopen, fclose, sizeof, FILE, NULL */

int main(void)
{
    FILE *myfile;
    char s[75];
    long y;

    myfile = fopen("afile.out", "w+");
    if (myfile == NULL)
        printf("Cannot open afile.out\n");
    else
    {
        fprintf(myfile,"This is a very long sentence 
        "for input into the file named 
        "afile.out for testing.");

        fclose(myfile);

        if ((myfile = fopen("afile.out", "rb")) != NULL)
            {
                printf("Read some characters:\n");
                fread(s, sizeof(char), 29, myfile);
                printf("\t"\%s\"\n", s);
                y = ftell(myfile);
                printf("The current position of the 
                "file pointer is %ld\n", y);
                fclose(myfile);
            }
    }
}
```

Output:
Read some characters:
"This is a very long sentence 
The current position of the file pointer is 29
fwrite

Description: Writes data to the stream.

Include: `<stdio.h>`

Prototype: `size_t fwrite(const void *ptr, size_t size, size_t nelem, FILE *stream);`

Arguments: `ptr` pointer to the storage buffer
`size` size of item
`nelem` maximum number of items to be read
`stream` pointer to the open stream

Return Value: Returns the number of complete elements successfully written, which will be less than `nelem` only if a write error is encountered.

Remarks: The function writes characters to a given stream from a buffer pointed to by `ptr` up to `nelem` elements whose size is specified by `size`. The file position indicator is advanced by the number of characters successfully written. If the function sets the error indicator, the file-position indicator is indeterminate.

Example:
```
#include <stdio.h> /* for fread, fwrite, */
/* printf, fopen, fclose, */
/* sizeof, FILE, NULL */

int main(void)
{
    FILE *buf;
    int x, numwrote, numread;
    double nums[10], readnums[10];

    if ((buf = fopen("afile.out", "w+")) != NULL) {
        for (x = 0; x < 10; x++)
            nums[x] = 10.0/(x + 1);
        printf("10.0/%d = %f\n", x+1, nums[x]);
    }
    numwrote = fwrite(nums, sizeof(double),
                       10, buf);
    printf("Wrote %d numbers\n\n", numwrote);
    fclose(buf);
}
else
    printf("Cannot open afile.out\n");
```
fwrite (Continued)

if ((buf = fopen("afile.out", "r+")) != NULL)
{
    numread = fread(readnums, sizeof(double),
                10, buf);
    printf("Read %d numbers\n", numread);
    for (x = 0; x < 10; x++)
    {
        printf("%d * %f = %f\n", x+1, readnums[x],
                (x + 1) * readnums[x]);
    }
    fclose(buf);
} else
    printf("Cannot open afile.out\n");

Output:
10.0/1 = 10.000000
10.0/2 = 5.000000
10.0/3 = 3.333333
10.0/4 = 2.500000
10.0/5 = 2.000000
10.0/6 = 1.666667
10.0/7 = 1.428571
10.0/8 = 1.250000
10.0/9 = 1.111111
10.0/10 = 1.000000
Wrote 10 numbers
Read 10 numbers
1 * 10.000000 = 10.000000
2 * 5.000000 = 10.000000
3 * 3.333333 = 10.000000
4 * 2.500000 = 10.000000
5 * 2.000000 = 10.000000
6 * 1.666667 = 10.000000
7 * 1.428571 = 10.000000
8 * 1.250000 = 10.000000
9 * 1.111111 = 10.000000
10 * 1.000000 = 10.000000

Explanation:
This program uses fwrite to save 10 numbers to a file in binary form. This allows the numbers to be saved in the same pattern of bits as the program is using which provides more accuracy and consistency. Using fprintf would save the numbers as text strings, which could cause the numbers to be truncated. Each number is divided into 10 to produce a variety of numbers. Retrieving the numbers with fread to a new array and multiplying them by the original number shows the numbers were not truncated in the save process.
getc

**Description:**
Get a character from the stream.

**Include:**
<stdio.h>

**Prototype:**
int getc(FILE *stream);

**Argument:**
stream  pointer to the open stream

**Return Value:**
Returns the character read or EOF if a read error occurs or end-of-file is reached.

**Remarks:**
getc is the same as the function fgetc.

**Example:**
#include <stdio.h> /* for getc, printf, */
/* fopen, fclose, */
/* FILE, NULL, EOF */

```c
int main(void)
{
    FILE *buf;
    char y;

    if ((buf = fopen("afile.txt", "r")) == NULL)
        printf("Cannot open afile.txt\n");
    else
    {
        y = getc(buf);
        while (y != EOF)
        {
            printf("%c", y);
            y = getc(buf);
        }
        fclose(buf);
    }
}
```

**Input:**
Contents of afile.txt (used as input):
Short
Longer string

**Output:**
S|h|o|r|t|
|L|o|n|g|e|r|s|t|r|i|n|g|
getchar

Description: Get a character from stdin.
Include: <stdio.h>
Prototype: int getchar(void);
Return Value: Returns the character read or EOF if a read error occurs or end-of-file is reached.
Remarks: Same effect as fgetc with the argument stdin.
Example:

```c
#include <stdio.h> /* for getchar, printf */

int main(void)
{
    char y;
    y = getchar();
    printf("%c\n", y);
    y = getchar();
    printf("%c\n", y);
    y = getchar();
    printf("%c\n", y);
    y = getchar();
    printf("%c\n", y);
    y = getchar();
    printf("%c\n", y);
}
```

Input:
Contents of UartIn.txt (used as stdin input for simulator):
Short
Longer string
Output:
S|h|o|r|t|

gets

Description: Get a string from stdin.
Include: <stdio.h>
Prototype: char *gets(char *s);
Argument: s pointer to the storage string
Return Value: Returns a pointer to the string s if successful; otherwise, returns a null pointer
Remarks: The function reads characters from the stream stdin and stores them into the string pointed to by s until it reads a newline character (which is not stored) or sets the end-of-file or error indicators. If any characters were read, a null character is stored immediately after the last read character in the next element of the array. If gets sets the error indicator, the array contents are indeterminate.
gets (Continued)

Example:

```c
#include <stdio.h> /* for gets, printf */

int main(void)
{
    char y[50];
    gets(y);  
    printf("Text: %s\n", y);
}
```

Input:
Contents of UartIn.txt (used as stdin input for simulator):
Short
Longer string

Output:
Text: Short

perror

Description: Prints an error message to stderr.

Include: `<stdio.h>`

Prototype: `void perror(const char *s);`

Argument: `s` string to print

Return Value: None.

Remarks: The string s is printed followed by a colon and a space. Then an error message based on `errno` is printed followed by a newline

Example:

```c
#include <stdio.h> /* for perror, fopen, */
/* fclose, printf, */
/* FILE, NULL */
int main(void)
{
    FILE *myfile;
    if ((myfile = fopen("samp.fil", "r+")) == NULL)
        perror("Cannot open samp.fil");
    else
        printf("Success opening samp.fil\n");

    fclose(myfile);
}
```

Output:
Cannot open samp.fil: file open error
**printf**

**Description:**
Prints formatted text to stdout.

**Include:**
<stdio.h>

**Prototype:**
int printf(const char *format, ...);

**Arguments:**
format format control string
... optional arguments

**Return Value:**
Returns number of characters generated or a negative number if an error occurs.

**Remarks:**
There must be exactly the same number of arguments as there are format specifiers. If there are less arguments than match the format specifiers, the output is undefined. If there are more arguments than match the format specifiers, the remaining arguments are discarded. Each format specifier begins with a percent sign followed by optional fields and a required type as shown here:

```
%[flags][width][.precision][size]type
```

**flags**
- left justify the value within a given field width
- 0 Use 0 for the pad character instead of space (which is the default)
- + generate a plus sign for positive signed values
- space generate a space or signed values that have neither a plus nor a minus sign
- # to prefix 0 on an octal conversion, to prefix 0x or 0X on a hexadecimal conversion, or to generate a decimal point and fraction digits that are otherwise suppressed on a floating-point conversion

**width**
specify the number of characters to generate for the conversion. If the asterisk (*) is used instead of a decimal number, the next argument (which must be of type int) will be used for the field width. If the result is less than the field width, pad characters will be used on the left to fill the field. If the result is greater than the field width, the field is expanded to accommodate the value without padding.

**precision**
The field width can be followed with dot (.) and a decimal integer representing the precision that specifies one of the following:
- minimum number of digits to generate on an integer conversion
- number of fraction digits to generate on an e, E, or f conversion
- maximum number of significant digits to generate on a g or G conversion
- maximum number of characters to generate from a C string on an s conversion

If the period appears without the integer the integer is assumed to be zero. If the asterisk (*) is used instead of a decimal number, the next argument (which must be of type int) will be used for the precision.
printf (Continued)

size
   h modifier – used with type d, i, o, u, x, X; converts the value to a short int or unsigned short int
   h modifier – used with n; specifies that the pointer points to a short int
   l modifier – used with type d, i, o, u, x, X; converts the value to a long int or unsigned long int
   l modifier – used with n; specifies that the pointer points to a long int
   l modifier – used with c; specifies a wide character
   l modifier – used with type e, E, f, F, g, G; converts the value to a double
   ll modifier – used with type d, i, o, u, x, X; converts the value to a long long int or unsigned long long int
   ll modifier – used with n; specifies that the pointer points to a long long int
   L modifier – used with e, E, f, G; converts the value to a long double

type
   d, i  signed int
   o  unsigned int in octal
   u  unsigned int in decimal
   x  unsigned int in lowercase hexadecimal
   X  unsigned int in uppercase hexadecimal
   e, E  double in scientific notation
   f  double decimal notation
   g, G  double (takes the form of e, E or f as appropriate)
   c  char - a single character
   s  string
   p  value of a pointer
   n  the associated argument shall be an integer pointer into which is placed the number of characters written so far. No characters are printed.
   %  A % character is printed

Example:

```c
#include <stdio.h> /* for printf */

int main(void)
{
    /* print a character right justified in a 3 character space. */
    printf("%3c\n", 'a');

    /* print an integer, left justified (as specified by the minus sign in the format string) in a 4 character space. Print a */
    /* second integer that is right justified in a 4 character space using the pipe (|) as a separator between the integers. */
    printf("%-4d|%4d\n", -4, 4);

    /* print a number converted to octal in 4 digits. */
    printf("%3o\n", 10);
```

/* print a number converted to hexadecimal */
/* format with a 0x prefix. */
printf("%#x\n", 28);

/* print a float in scientific notation */
printf("%E\n", 1.1e20);

/* print a float with 2 fraction digits */
printf("%2f\n", -3.346);

/* print a long float with %E, %e, or %f */
/* whichever is the shortest version */
printf("%Lg\n", .02L);

Output:
a
-4  |   4
0012
0xc
1.100000E+20
-3.35
0.02

putc

Description: Puts a character to the stream.
Include: <stdio.h>
Prototype: int putc(int c, FILE *stream);
Arguments: c character to be written
            stream pointer to FILE structure
Return Value: Returns the character or EOF if an error occurs or end-of-file is reached.
Remarks: putc is the same as the function fputc.
Example: #include <stdio.h> /* for putc, EOF, stdout */

    int main(void) {

        char *y;
        char buf[] = "This is text\n";
        int x;

        x = 0;

        for (y = buf; (x != EOF) && (*y != '\0'); y++)
            {
                x = putc(*y, stdout);
                putc('|', stdout);
            }

    }

Output:
This is text
|
**putchar**

Description: Put a character to stdout.

Include: `<stdio.h>`

Prototype: `int putchar(int c);`

Argument: `c` character to be written

Return Value: Returns the character or EOF if an error occurs or end-of-file is reached.

Remarks: Same effect as `fputc` with `stdout` as an argument.

Example:
```
#include <stdio.h> /* for putchar, printf, */
/* EOF, stdout          */

int main(void)
{
    char *y;
    char buf[] = "This is text\n";
    int x;
    x = 0;
    for (y = buf; (x != EOF) && (*y != '\0'); y++)
        x = putchar(*y);
}
```

Output: This is text

**puts**

Description: Put a string to stdout.

Include: `<stdio.h>`

Prototype: `int puts(const char *s);`

Argument: `s` string to be written

Return Value: Returns a non-negative value if successful; otherwise, returns EOF.

Remarks: The function writes characters to the stream `stdout`. A newline character is appended. The terminating null character is not written to the stream.

Example:
```
#include <stdio.h> /* for puts */

int main(void)
{
    char buf[] = "This is text\n";
    puts(buf);
    puts("|");
}
```

Output: This is text
### remove

**Description:** Deletes the specified file.

**Include:** `<stdio.h>`

**Prototype:**

```c
int remove(const char *filename);
```

**Argument:**

- `filename` name of file to be deleted.

**Return Value:** Returns 0 if successful, -1 if not.

**Remarks:** If `filename` does not exist or is open, `remove` will fail.

**Example:**

```c
#include <stdio.h> /* for remove, printf */

int main(void)
{
    if (remove("myfile.txt") != 0)
        printf("Cannot remove file");
    else
        printf("File removed");
}
```

**Output:**

File removed

### rename

**Description:** Renames the specified file.

**Include:** `<stdio.h>`

**Prototype:**

```c
int rename(const char *old, const char *new);
```

**Arguments:**

- `old` pointer to the old name
- `new` pointer to the new name.

**Return Value:** Return 0 if successful, non-zero if not.

**Remarks:** The new name must not already exist in the current working directory, the old name must exist in the current working directory.

**Example:**

```c
#include <stdio.h> /* for rename, printf */

int main(void)
{
    if (rename("myfile.txt","newfile.txt") != 0)
        printf("Cannot rename file");
    else
        printf("File renamed");
}
```

**Output:**

File renamed
**rewind**

**Description:** Resets the file pointer to the beginning of the file.

**Include:** `<stdio.h>`

**Prototype:**
```c
void rewind(FILE *stream);
```

**Argument:** `stream` stream to reset the file pointer

**Remarks:** The function calls `fseek(stream, 0L, SEEK_SET)` and then clears the error indicator for the given stream.

**Example:**
```c
#include <stdio.h> /* for rewind, fopen, */
/* fscanf, fclose, */
/* fprintf, printf, */
/* FILE, NULL */

int main(void)
{
    FILE *myfile;
    char s[] = "cookies";
    int x = 10;

    if ((myfile = fopen("afile", "w+")) == NULL)
        printf("Cannot open afile\n");
    else
    {
        fprintf(myfile, "%d %s", x, s);
        printf("I have %d %s.\n", x, s);

        /* set pointer to beginning of file */
        rewind(myfile);
        fscanf(myfile, "%d %s", &x, &s);
        printf("I ate %d %s.\n", x, s);

        fclose(myfile);
    }
}
```

**Output:**
I have 10 cookies.
I ate 10 cookies.
**scanf**

<table>
<thead>
<tr>
<th>Description:</th>
<th>Scans formatted text from stdin.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td><code>&lt;stdio.h&gt;</code></td>
</tr>
<tr>
<td>Prototype:</td>
<td><code>int scanf(const char *format, ...);</code></td>
</tr>
<tr>
<td>Argument:</td>
<td><code>format</code> format control string</td>
</tr>
<tr>
<td></td>
<td><code>...</code> optional arguments</td>
</tr>
<tr>
<td>Return Value:</td>
<td>Returns the number of items successfully converted and assigned. If no items are assigned, a 0 is returned. EOF is returned if an input failure is encountered before the first.</td>
</tr>
<tr>
<td>Remarks:</td>
<td>Each format specifier begins with a percent sign followed by optional fields and a required type as shown here: <code>%[*][width][modifier]type</code></td>
</tr>
</tbody>
</table>

- `*` indicates assignment suppression. This will cause the input field to be skipped and no assignment made.
- `width` specify the maximum number of input characters to match for the conversion not including white space that can be skipped.
- `modifier`
  - `h modifier` used with type `d`, `i`, `o`, `u`, `x`, `X`; converts the value to a short int or unsigned short int.
  - `l modifier` used with type `d`, `i`, `o`, `u`, `x`, `X`; converts the value to a long int or unsigned long int.
  - `ll modifier` used with type `d`, `i`, `o`, `u`, `x`, `X`; converts the value to a long long int or unsigned long long int.
  - `L modifier` used with type `e`, `E`, `f`, `F`, `g`, `G`; converts the value to a long double.
scanf (Continued)

<table>
<thead>
<tr>
<th>type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>d,i</td>
<td>signed int</td>
</tr>
<tr>
<td>o</td>
<td>unsigned int in octal</td>
</tr>
<tr>
<td>u</td>
<td>unsigned int in decimal</td>
</tr>
<tr>
<td>x</td>
<td>unsigned int in lowercase hexadecimal</td>
</tr>
<tr>
<td>X</td>
<td>unsigned int in uppercase hexadecimal</td>
</tr>
<tr>
<td>e,E</td>
<td>double in scientific notation</td>
</tr>
<tr>
<td>f</td>
<td>double decimal notation</td>
</tr>
<tr>
<td>g,G</td>
<td>double (takes the form of e, E or f as appropriate)</td>
</tr>
<tr>
<td>c</td>
<td>char - a single character</td>
</tr>
<tr>
<td>s</td>
<td>string</td>
</tr>
<tr>
<td>p</td>
<td>value of a pointer</td>
</tr>
<tr>
<td>n</td>
<td>the associated argument shall be an integer pointer into, which is placed the number of characters read so far. No characters are scanned.</td>
</tr>
<tr>
<td>p</td>
<td>value of a pointer</td>
</tr>
<tr>
<td>n</td>
<td>the associated argument shall be an integer pointer into, which is placed the number of characters read so far. No characters are scanned.</td>
</tr>
</tbody>
</table>

Character array. Allows a search of a set of characters. A caret (^) immediately after the left bracket ( [ ) inverts the scanset and allows any ASCII character except those specified between the brackets. A dash character (-) may be used to specify a range beginning with the character before the dash and ending the character after the dash. A null character can not be part of the scanset.

A % character is scanned

Example:

```c
#include <stdio.h> /* for scanf, printf */

int main(void)
{
    int number, items;
    char letter;
    char color[30], string[30];
    float salary;

    printf("Enter your favorite number, 
       favorite letter, ");
    printf("favorite color desired salary 
       and SSN:\n");
    items = scanf("%d %c [%A-Za-z] %f %s", &number, 
       &letter, &color, &salary, &string);

    printf("Number of items scanned = %d\n", items);
    printf("Favorite number = %d, ", number);
    printf("Favorite letter = %c\n", letter);
    printf("Favorite color = %s, ", color);
    printf("Desired salary = $%.2f\n", salary);
    printf("Social Security Number = %s, ", string);
}
```

Input:
Contents of UartIn.txt (used as stdin input for simulator):
5 T Green 300000 123-45-6789

Output:
Enter your favorite number, favorite letter, favorite color, desired salary and SSN:
Number of items scanned = 5
Favorite number = 5, Favorite letter = T
Favorite color = Green, Desired salary = $300000.00
Social Security Number = 123-45-6789
setbuf

Description: Defines how a stream is buffered.

Include: `<stdio.h>`

Prototype: `void setbuf(FILE *stream, char *buf);`

Arguments: 
- `stream` pointer to the open stream
- `buf` user allocated buffer

Remarks: `setbuf` must be called after `fopen` but before any other function calls that operate on the stream. If `buf` is a null pointer, `setbuf` calls the function `setvbuf(stream, 0, _IONBF, BUFSIZ)` for no buffering; otherwise `setbuf` calls `setvbuf(stream, buf, _IOFBF, BUFSIZ)` for full buffering with a buffer of size `BUFSIZ`. See `setvbuf`.

Example:
```c
#include <stdio.h> /* for setbuf, printf, */ 
/* fopen, fclose, */ 
/* FILE, NULL, BUFSIZ */

int main(void)
{
    FILE *myfile1, *myfile2;
    char buf[BUFSIZ];

    if ((myfile1 = fopen("afile1", "w+")) != NULL)
    {
        setbuf(myfile1, NULL);
        printf("myfile1 has no buffering\n");
        fclose(myfile1);
    }

    if ((myfile2 = fopen("afile2", "w+")) != NULL)
    {
        setbuf(myfile2, buf);
        printf("myfile2 has full buffering");
        fclose(myfile2);
    }
}
```

Output:
myfile1 has no buffering
myfile2 has full buffering
### setvbuf

**Description:**
Defines the stream to be buffered and the buffer size.

**Include:**
```c
<stdio.h>
```

**Prototype:**
```c
int setvbuf(FILE *stream, char *buf, int mode, size_t size);
```

**Arguments:**
- `stream` pointer to the open stream
- `buf` user allocated buffer
- `mode` type of buffering
- `size` size of buffer

**Return Value:**
Returns 0 if successful

**Remarks:**
- `setvbuf` must be called after `fopen` but before any other function calls that operate on the stream. For mode use one of the following:
  - `_IOFBF` — for full buffering
  - `_IOLBF` — for line buffering
  - `_IONBF` — for no buffering

**Example:**
```c
#include <stdio.h> /* for setvbuf, fopen, */
/* printf, FILE, NULL, */
/* _IONBF, _IOFBF      */

int main(void)
{
    FILE *myfile1, *myfile2;
    char buf[256];

    if ((myfile1 = fopen("afile1", "w+")) != NULL)
    {
        if (setvbuf(myfile1, NULL, _IONBF, 0) == 0)
            printf("myfile1 has no buffering\n");
        else
            printf("Unable to define buffer stream "
                   "and/or size\n");
    }
    fclose(myfile1);

    if ((myfile2 = fopen("afile2", "w+")) != NULL)
    {
        if (setvbuf(myfile2, buf, _IOFBF, sizeof(buf)) == 0)
            printf("myfile2 has a buffer of %d "
                   "characters\n", sizeof(buf));
        else
            printf("Unable to define buffer stream "
                   "and/or size\n");
    }
    fclose(myfile2);
}
```

**Output:**
- myfile1 has no buffering
- myfile2 has a buffer of 256 characters
**sprintf**

**Description:** Prints formatted text to a string

**Include:**
```
<stdio.h>
```

**Prototype:**
```
int sprintf(char *s, const char *format, ...);
```

**Arguments:**
- `s` storage string for output
- `format` format control string
- `...` optional arguments

**Return Value:** Returns the number of characters stored in `s` excluding the terminating null character.

**Remarks:** The format argument has the same syntax and use that it has in `printf`.

**Example:**
```
#include <stdio.h> /* for sprintf, printf */

int main(void)
{
    char sbuf[100], s[]="Print this string";
    int x = 1, y;
    char a = '\n';

    y = sprintf(sbuf, "%s %d time%c", s, x, a);

    printf("Number of characters printed to 
        "string buffer = %d\n", y);
    printf("String = %s\n", sbuf);
}
```

**Output:**
```
Number of characters printed to string buffer = 25
String = Print this string 1 time
```

---

**sscanf**

**Description:** Scans formatted text from a string

**Include:**
```
<stdio.h>
```

**Prototype:**
```
int sscanf(const char *s, const char *format, ...);
```

**Arguments:**
- `s` storage string for input
- `format` format control string
- `...` optional arguments

**Return Value:** Returns the number of items successfully converted and assigned. If no items are assigned, a 0 is returned. EOF is returned if an input error is encountered before the first conversion.

**Remarks:** The format argument has the same syntax and use that it has in `scanf`. 
**sscanf (Continued)**

**Example:**

```c
#include <stdio.h> /* for sscanf, printf */

int main(void)
{
    char s[] = "5 T green 3000000.00";
    int number, items;
    char letter;
    char color[10];
    float salary;

    items = sscanf(s, "%d %c %s %f", &number, &letter, &color, &salary);

    printf("Number of items scanned = %d\n", items);
    printf("Favorite number = %d\n", number);
    printf("Favorite letter = %c\n", letter);
    printf("Favorite color = %s\n", color);
    printf("Desired salary = $%.2f\n", salary);
}
```

**Output:**

```
Number of items scanned = 4
Favorite number = 5
Favorite letter = T
Favorite color = green
Desired salary = $3000000.00
```

**tmpfile**

**Description:** Creates a temporary file

**Include:** `<stdio.h>`

**Prototype:** `FILE *tmpfile(void)`

**Return Value:** Returns a stream pointer if successful; otherwise, returns a NULL pointer.

**Remarks:** `tmpfile` creates a file with a unique filename. The temporary file is opened in `w+b` (binary read/write) mode. It will automatically be removed when `exit` is called; otherwise the file will remain in the directory.

**Example:**

```c
#include <stdio.h> /* for tmpfile, printf, */
/* FILE, NULL */

int main(void)
{
    FILE *mytempfile;

    if ((mytempfile = tmpfile()) == NULL)
        printf("Cannot create temporary file\n");
    else
        printf("Temporary file was created\n");
}
```

**Output:**

```
Temporary file was created
```
### tmpnam

**Description:** Creates a unique temporary filename

**Include:** `<stdio.h>`

**Prototype:**

```c
char *tmpnam(char *s);
```

**Argument:** `s` pointer to the temporary name

**Return Value:** Returns a pointer to the filename generated and stores the filename in `s`. If it can not generate a filename, the NULL pointer is returned.

**Remarks:** The created filename will not conflict with an existing file name. Use `L_tmpnam` to define the size of array the argument of `tmpnam` points to.

**Example:**

```c
#include <stdio.h> /* for tmpnam, L_tmpnam, */
/* printf, NULL          */

int main(void)
{
    char *myfilename;
    char mybuf[L_tmpnam];
    char *myptr = (char *) &mybuf;

    if ((myfilename = tmpnam(myptr)) == NULL)
        printf("Cannot create temporary file name");
    else
        printf("Temporary file %s was created",
                myfilename);
}
```

**Output:**

Temporary file ctm00001.tmp was created

---

### ungetc

**Description:** Pushes character back onto stream.

**Include:** `<stdio.h>`

**Prototype:**

```c
int ungetc(int c, FILE *stream);
```

**Argument:**

- `c` character to be pushed back
- `stream` pointer to the open stream

**Return Value:** Returns the pushed character if successful; otherwise, returns EOF

**Remarks:** The pushed back character will be returned by a subsequent read on the stream. If more than one character is pushed back, they will be returned in the reverse order of their pushing. A successful call to a file positioning function (`fseek`, `fsetpos` or `rewind`) cancels any pushed back characters. Only one character of pushback is guaranteed. Multiple calls to `ungetc` without an intervening read or file positioning operation may cause a failure.
Example:

```c
#include <stdio.h> /* for ungetc, fgetc, printf, fopen, fclose, */
/* FILE, NULL, EOF */

int main(void)
{
    FILE *buf;
    char y, c;
    if ((buf = fopen("afile.txt", "r")) == NULL)
        printf("Cannot open afile.txt\n");
    else
    {
        y = fgetc(buf);
        while (y != EOF)
        {
            if (y == 'r')
            {
                c = ungetc(y, buf);
                if (c != EOF)
                {
                    printf("2");
                    y = fgetc(buf);
                }
            }
            printf("%c", y);
            y = fgetc(buf);
        }
    fclose(buf);
}
```

**Input:**
Contents of afile.txt (used as input):
Short
Longer string

**Output:**
Sho2rt
Longe2r st2ring
vfprintf

Description: Prints formatted data to a stream using a variable length argument list.

Include:

```
#include <stdio.h>
#include <stdarg.h>
```

Prototype:

```
#include <stdio.h>
#include <stdarg.h>

int vfprintf(FILE *stream, const char *format, va_list ap);
```

Arguments:

- `stream` pointer to the open stream
- `format` format control string
- `ap` pointer to a list of arguments

Return Value: Returns number of characters generated or a negative number if an error occurs.

Remarks: The format argument has the same syntax and use that it has in printf.

To access the variable length argument list, the `ap` variable must be initialized by the macro `va_start` and may be reinitialized by additional calls to `va_arg`. This must be done before the `vfprintf` function is called. Invoke `va_end` after the function returns. For more details see `stdarg.h`.

Example:

```
#include <stdio.h>  /* for vfprintf, fopen, */
#include <stdarg.h>  /* for va_start, */
#include <stdlib.h>  /* for NULL */

FILE *myfile;

void errmsg(const char *fmt, ...) {
  va_list ap;
  va_start(ap, fmt);
  vfprintf(myfile, fmt, ap);
  va_end(ap);
}

int main(void) {
  int num = 3;
  if ((myfile = fopen("afile.txt", "w")) == NULL) {
    printf("Cannot open afile.txt\n");
  } else {
    errmsg("Error: The letter '%c' is not %s\n", 'a', "an integer value.");
    errmsg("Error: Requires %d%s%c", num, " or more characters.", '\n');
  }
  fclose(myfile);
}
```

Output:

Contents of afile.txt

Error: The letter 'a' is not an integer value.
Error: Requires 3 or more characters.
## vprintf

**Description:**
Prints formatted text to stdout using a variable length argument list

**Include:**
```c
<stdio.h>
<stdarg.h>
```

**Prototype:**
```c
int vprintf(const char *format, va_list ap);
```

**Arguments:**
- `format`  format control string
- `ap`  pointer to a list of arguments

**Return Value:**
Returns number of characters generated or a negative number if an error occurs.

**Remarks:**
The format argument has the same syntax and use that it has in printf.

To access the variable length argument list, the `ap` variable must be initialized by the macro `va_start` and may be reinitialized by additional calls to `va_arg`. This must be done before the `vprintf` function is called. Invoke `va_end` after the function returns. For more details see `stdarg.h`

**Example:**
```c
#include <stdio.h>   /* for vprintf, printf */
#include <stdarg.h>  /* for va_start, va_list, va_end */

void errmsg(const char *fmt, ...)
{
    va_list ap;
    va_start(ap, fmt);
    printf("Error: ");
    vprintf(fmt, ap);
    va_end(ap);
}

int main(void)
{
    int num = 3;

    errmsg("The letter '%c' is not %s\n", 'a',
            "an integer value.");
    errmsg("Requires %d%s\n", num,
            " or more characters.\n");
}
```

**Output:**
```
Error: The letter 'a' is not an integer value.
Error: Requires 3 or more characters.
```
vsprintf

Description: Prints formatted text to a string using a variable length argument list

Include:
- `<stdio.h>`
- `<stdarg.h>`

Prototype:
```
int vsprintf(char *s, const char *format, va_list ap);
```

Arguments:
- `s`  storage string for output
- `format`  format control string
- `ap`  pointer to a list of arguments

Return Value: Returns number of characters stored in `s` excluding the terminating null character.

Remarks: The format argument has the same syntax and use that it has in `printf`. To access the variable length argument list, the `ap` variable must be initialized by the macro `va_start` and may be reinitialized by additional calls to `va_arg`. This must be done before the `vsprintf` function is called. Invoke `va_end` after the function returns. For more details see `<stdarg.h>`

Example:
```
#include <stdio.h>   /* for vsprintf, printf */
#include <stdarg.h>  /* for va_start, va_list, va_end */

void errmsg(const char *fmt, ...)
{
    va_list ap;
    char buf[100];

    va_start(ap, fmt);
    vsprintf(buf, fmt, ap);
    va_end(ap);
    printf("Error: %s", buf);
}

int main(void)
{
    int num = 3;

    errmsg("The letter '%c' is not '%s'\n", 'a', "an integer value.");
    errmsg("Requires %d%s\n", num, " or more characters.\n");
}
```

Output:
Error: The letter 'a' is not an integer value.
Error: Requires 3 or more characters.
4.14  `<stdlib.h>` UTILITY FUNCTIONS

The header file `stdlib.h` consists of types, macros and functions that provide text conversions, memory management, searching and sorting abilities, and other general utilities.

<table>
<thead>
<tr>
<th><strong>div_t</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> A type that holds a quotient and remainder of a signed integer division with operands of type <code>int</code>.</td>
</tr>
<tr>
<td><strong>Include:</strong> <code>&lt;stdlib.h&gt;</code></td>
</tr>
<tr>
<td><strong>Prototype:</strong> <code>typedef struct { int quot, rem; } div_t;</code></td>
</tr>
<tr>
<td><strong>Remarks:</strong> This is the structure type returned by the function <code>div</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>ldiv_t</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> A type that holds a quotient and remainder of a signed integer division with operands of type <code>long</code>.</td>
</tr>
<tr>
<td><strong>Include:</strong> <code>&lt;stdlib.h&gt;</code></td>
</tr>
<tr>
<td><strong>Prototype:</strong> <code>typedef struct { long quot, rem; } ldiv_t;</code></td>
</tr>
<tr>
<td><strong>Remarks:</strong> This is the structure type returned by the function <code>ldiv</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>size_t</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> The type of the result of the <code>sizeof</code> operator.</td>
</tr>
<tr>
<td><strong>Include:</strong> <code>&lt;stdlib.h&gt;</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>wchar_t</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> A type that holds a wide character value.</td>
</tr>
<tr>
<td><strong>Include:</strong> <code>&lt;stdlib.h&gt;</code></td>
</tr>
</tbody>
</table>

**EXIT_FAILURE**

| **Description:** Reports unsuccessful termination. |
| **Include:** `<stdlib.h>` |
| **Remarks:** `EXIT_FAILURE` is a value for the exit function to return an unsuccessful termination status |
| **Example:** See `exit` for example of use. |

**EXIT_SUCCESS**

| **Description:** Reports successful termination |
| **Include:** `<stdlib.h>` |
| **Remarks:** `EXIT_SUCCESS` is a value for the exit function to return a successful termination status. |
| **Example:** See `exit` for example of use. |
### MB_CUR_MAX

**Description:** Maximum number of characters in a multibyte character

**Include:** `<stdlib.h>`

**Value:** 1

---

### NULL

**Description:** The value of a null pointer constant

**Include:** `<stdlib.h>`

---

### RAND_MAX

**Description:** Maximum value capable of being returned by the `rand` function

**Include:** `<stdlib.h>`

**Value:** 32767

---

### abort

**Description:** Aborts the current process.

**Include:** `<stdlib.h>`

**Prototype:**

```c
void abort(void);
```

**Remarks:** abort will cause the processor to reset.

**Example:**

```c
#include <stdio.h> /* for fopen, fclose, */
/* printf, FILE, NULL */
#include <stdlib.h> /* for abort          */

int main(void)
{
    FILE *myfile;

    if ((myfile = fopen("samp.fil", "r")) == NULL)
    {
        printf("Cannot open samp.fil\n");
        abort();
    } else
    {
        printf("Success opening samp.fil\n");

        fclose(myfile);
    }
}
```

**Output:**

```
Cannot open samp.fil
ABRT
```
abs

Description: Calculates the absolute value.

Include: `<stdlib.h>`

Prototype: `int abs(int i);`

Argument: `i` integer value

Return Value: Returns the absolute value of `i`.

Remarks: A negative number is returned as positive; a positive number is unchanged.

Example:
```c
#include <stdio.h>  /* for printf */
#include <stdlib.h> /* for abs    */

int main(void)
{
    int i;

    i = 12;
    printf("The absolute value of  %d is  %d\n", i, abs(i));

    i = -2;
    printf("The absolute value of  %d is   %d\n", i, abs(i));

    i = 0;
    printf("The absolute value of   %d is   %d\n", i, abs(i));
}
```

Output:
The absolute value of  12 is  12
The absolute value of  -2 is   2
The absolute value of   0 is   0

atexit

Description: Registers the specified function to be called when the program terminates normally.

Include: `<stdlib.h>`

Prototype: `int atexit(void(*func)(void));`

Argument: `func` function to be called

Return Value: Returns a zero if successful; otherwise, returns a non-zero value.

Remarks: For the registered functions to be called, the program must terminate with the `exit` function call.

Example:
```c
#include <stdio.h>  /* for scanf, printf */
#include <stdlib.h> /* for atexit, exit */

void good_msg(void);
void bad_msg(void);
void end_msg(void);
```
int main(void)
{
    int number;

    atexit(end_msg);
    printf("Enter your favorite number:");
    scanf("%d", &number);
    printf(" %d\n", number);
    if (number == 5)
    {
        printf("Good Choice\n");
        atexit(good_msg);
        exit(0);
    }
    else
    {
        printf("%d!?\n", number);
        atexit(bad_msg);
        exit(0);
    }

}

void good_msg(void)
{
    printf("That's an excellent number\n");
}

void bad_msg(void)
{
    printf("That's an awful number\n");
}

void end_msg(void)
{
    printf("Now go count something\n");
}

Input:
With contents of UartIn.txt (used as stdin input for simulator):
5
Output:
Enter your favorite number: 5
Good Choice
That's an excellent number
Now go count something

Input:
With contents of UartIn.txt (used as stdin input for simulator):
42
Output:
Enter your favorite number: 42
42!?
That's an awful number
Now go count something
atof

Description: Converts a string to a double precision floating-point value.

Include: <stdlib.h>

Prototype: double atof(const char *s);

Argument: s pointer to the string to be converted

Return Value: Returns the converted value if successful; otherwise, returns 0.

Remarks: The number may consist of the following:

- [whitespace] [sign] digits [.digits]
- [ { e | E }[sign]digits] optional whitespace, followed by an optional sign then a sequence of one or more digits with an optional decimal point, followed by one or more optional digits and an optional e or E followed by an optional signed exponent. The conversion stops when the first unrecognized character is reached. The conversion is the same as *strtof*(s,0,0) except it does no error checking so *errno* will not be set.

Example:

```c
#include <stdio.h>  /* for printf */
#include <stdlib.h> /* for atof   */

int main(void)
{
    char a[] = " 1.28";
    char b[] = "27.835e2";
    char c[] = "Number1";
    double x;

    x = atof(a);
    printf("String = \"%s\" float = %f\n", a, x);

    x = atof(b);
    printf("String = \"%s\" float = %f\n", b, x);

    x = atof(c);
    printf("String = \"%s\"  float = %f\n", c, x);
}
```

Output:

```
String = "1.28" float = 1.280000
String = "27.835:e2" float = 2783.500000
String = "Number1" float = 0.000000
```
**atoi**

**Description:** Converts a string to an integer.

**Include:** `<stdlib.h>`

**Prototype:**

```c
int atoi(const char *s);
```

**Argument:** `s` string to be converted

**Return Value:** Returns the converted integer if successful; otherwise, returns 0.

**Remarks:**

The number may consist of the following:

- Optional whitespace, followed by an optional sign then a sequence of one or more digits. The conversion stops when the first unrecognized character is reached. The conversion is equivalent to `(int)strtol(s,0,10)` except it does no error checking so `errno` will not be set.

**Example:**

```c
#include <stdio.h> /* for printf */
#include <stdlib.h> /* for atoi   */

int main(void)
{
    char a[] = " -127";
    char b[] = "Number1";
    int x;

    x = atoi(a);
    printf("String = \\
           \"%s\\\tint = %d\n", a, x);

    x = atoi(b);
    printf("String = \\
           \"%s\\\tint = %d\n", b, x);
}
```

**Output:**

```
String = " -127"  int = -127
String = "Number1"  int = 0
```

**atol**

**Description:** Converts a string to a long integer.

**Include:** `<stdlib.h>`

**Prototype:**

```c
long atol(const char *s);
```

**Argument:** `s` string to be converted

**Return Value:** Returns the converted long integer if successful; otherwise, returns 0.

**Remarks:**

The number may consist of the following:

- Optional whitespace, followed by an optional sign then a sequence of one or more digits. The conversion stops when the first unrecognized character is reached. The conversion is equivalent to `(int)strtol(s,0,10)` except it does no error checking so `errno` will not be set.
### atol (Continued)

#### Example:
```c
#include <stdio.h>  /* for printf */
#include <stdlib.h> /* for atol   */

int main(void)
{
    char a[] = " -123456";
    char b[] = "2Number";
    long x;

    x = atol(a);
    printf("String = \"%s\"  int = %ld\n", a, x);

    x = atol(b);
    printf("String = \"%s\"  int = %ld\n", b, x);
}
```

#### Output:
```
String = " -123456"     int = -123456
String = "2Number"      int = 2
```

---

### bsearch

#### Description:
Performs a binary search

#### Include:
`<stdlib.h>`

#### Prototype:
```
void *bsearch(const void *key, const void *base, size_t nelem, size_t size,
              int (*cmp)(const void *ck, const void *ce));
```

#### Arguments:
- **key**: object to search for
- **base**: pointer to the start of the search data
- **nelem**: number of elements
- **size**: size of elements
- **cmp**: pointer to the comparison function
- **ck**: pointer to the key for the search
- **ce**: pointer to the element being compared with the key.

#### Return Value:
Returns a pointer to the object being searched for if found; otherwise, returns NULL.

#### Remarks:
The value returned by the compare function is <0 if `ck` is less than `ce`, 0 if `ck` is equal to `ce`, or >0 if `ck` is greater than `ce.` In the following example, `qsort` is used to sort the list before `bsearch` is called. `bsearch` requires the list to be sorted according to the comparison function. This `cmp` uses ascending order.
Example:

```c
#include <stdlib.h> /* for bsearch, qsort */
#include <stdio.h>  /* for printf, sizeof */

#define NUM 7

int comp(const void *e1, const void *e2);

int main(void)
{
  int list[NUM] = {35, 47, 63, 25, 93, 16, 52};
  int x, y;
  int *r;

  qsort(list, NUM, sizeof(int), comp);

  printf("Sorted List: ");
  for (x = 0; x < NUM; x++)
    printf("%d  ", list[x]);

  y = 25;
  r = bsearch(&y, list, NUM, sizeof(int), comp);
  if (r)
    printf("\nThe value %d was found\n", y);
  else
    printf("\nThe value %d was not found\n", y);

  y = 75;
  r = bsearch(&y, list, NUM, sizeof(int), comp);
  if (r)
    printf("\nThe value %d was found\n", y);
  else
    printf("\nThe value %d was not found\n", y);
}

int comp(const void *e1, const void *e2)
{
  const int * a1 = e1;
  const int * a2 = e2;

  if (*a1 < *a2)
    return -1;
  else if (*a1 == *a2)
    return 0;
  else
    return 1;
}
```

Output:
```
Sorted List: 16 25 35 47 52 63 93
The value 25 was found

The value 75 was not found
```
**calloc**

**Description:** Allocates an array in memory and initializes the elements to 0.

**Include:** `<stdlib.h>`

**Prototype:**

```c
void *calloc(size_t nelem, size_t size);
```

**Arguments:**

- `nelem`: number of elements
- `size`: length of each element

**Return Value:** Returns a pointer to the allocated space if successful; otherwise, returns a null pointer.

**Remarks:** Memory returned by calloc is aligned correctly for any size data element and is initialized to zero.

**Example:**

```c
/* This program allocates memory for the array 'i' of long integers and initializes them to zero. */

#include <stdio.h>  /* for printf, NULL */
#include <stdlib.h> /* for calloc, free */

int main(void)
{
    int x;
    long *i;

    i = (long *)calloc(5, sizeof(long));
    if (i != NULL)
    {
        for (x = 0; x < 5; x++)
            printf("i[%d] = %ld\n", x, i[x]);
        free(i);
    }
    else
        printf("Cannot allocate memory\n");
}
```

**Output:**

```
i[0] = 0
i[1] = 0
i[2] = 0
i[3] = 0
i[4] = 0
```

---

**div**

**Description:** Calculates the quotient and remainder of two numbers

**Include:** `<stdlib.h>`

**Prototype:**

```c
div_t div(int numer, int denom);
```

**Arguments:**

- `numer`: numerator
- `denom`: denominator

**Return Value:** Returns the quotient and the remainder.

**Remarks:** The returned quotient will have the same sign as the numerator divided by the denominator. The sign for the remainder will be such that the quotient times the denominator plus the remainder will equal the numerator (`quot * denom + rem = numer`). Division by zero will invoke the math exception error, which by default, will cause a reset. Write a math error handler to do something else.
Example:

```c
#include <stdlib.h> /* for div, div_t */
#include <stdio.h> /* for printf */

void __attribute__((__interrupt__)) _MathError(void)
{
    printf("Illegal instruction executed\n");
    abort();
}

int main(void)
{
    int x, y;
    div_t z;

    x = 7;
    y = 3;
    printf("For div(%d, %d)\n", x, y);
    z = div(x, y);
    printf("The quotient is %d and the "
            "remainder is %d\n", z.quot, z.rem);

    x = 7;
    y = -3;
    printf("For div(%d, %d)\n", x, y);
    z = div(x, y);
    printf("The quotient is %d and the "
            "remainder is %d\n", z.quot, z.rem);

    x = -5;
    y = 3;
    printf("For div(%d, %d)\n", x, y);
    z = div(x, y);
    printf("The quotient is %d and the "
            "remainder is %d\n", z.quot, z.rem);

    x = 7;
    y = 7;
    printf("For div(%d, %d)\n", x, y);
    z = div(x, y);
    printf("The quotient is %d and the "
            "remainder is %d\n", z.quot, z.rem);
}
```
**div (Continued)**

Output:
For div(7, 3)
The quotient is 2 and the remainder is 1

For div(7, -3)
The quotient is -2 and the remainder is 1

For div(-5, 3)
The quotient is -1 and the remainder is -2

For div(7, 7)
The quotient is 1 and the remainder is 0

For div(7, 0)
Illegal instruction executed
ABRT

---

**exit**

**Description:** Terminates program after clean up.

**Include:** `<stdlib.h>`

**Prototype:**
```c
void exit(int status);
```

**Argument:**
`status` exit status

**Remarks:**
exit calls any functions registered by `atexit` in reverse order of registration, flushes buffers, closes stream, closes any temporary files created with `tmpfile`, and resets the processor. This function is customizable. See `pic30-libs`.

**Example:**
```c
#include <stdio.h>  /* for fopen, printf, */
/* FILE, NULL         */
#include <stdlib.h> /* for exit           */

int main(void)
{
    FILE *myfile;
    
    if ((myfile = fopen("samp.fil", "r" )) == NULL)
    {
        printf("Cannot open samp.fil\n");
        exit(EXIT_FAILURE);
    }
    else
    {
        printf("Success opening samp.fil\n");
        exit(EXIT_SUCCESS);
    }
    printf("This will not be printed");
}
```

**Output:**
Cannot open samp.fil
free

Description: Frees memory.
Include: <stdlib.h>
Prototype: void free(void *ptr);
Argument: ptr points to memory to be freed
Remarks: Frees memory previously allocated with calloc, malloc, or realloc. If free is used on space that has already been deallocated (by a previous call to free or by realloc) or on space not allocated with calloc, malloc, or realloc, the behavior is undefined.
Example:
```c
#include <stdio.h> /* for printf, sizeof, */
/* NULL */
#include <stdlib.h> /* for malloc, free */

int main(void)
{
    long *i;

    if ((i = (long *)malloc(50 * sizeof(long))) == NULL)
        printf("Cannot allocate memory\n");
    else
    {
        printf("Memory allocated\n");
        free(i);
        printf("Memory freed\n");
    }
}
```
Output:
Memory allocated
Memory freed

getenv

Description: Get a value for an environment variable.
Include: <stdlib.h>
Prototype: char *getenv(const char *name);
Argument: name name of environment variable
Return Value: Returns a pointer to the value of the environment variable if successful; otherwise, returns a null pointer.
Remarks: This function must be customized to be used as described (see pic30-libs). By default there are no entries in the environment list for getenv to find.
getenv (Continued)

Example:

```c
#include <stdio.h> /* for printf, NULL */
#include <stdlib.h> /* for getenv */

int main(void)
{
    char *incvar;

    incvar = getenv("INCLUDE");
    if (incvar != NULL)
        printf("INCLUDE environment variable = %s\n", incvar);
    else
        printf("Cannot find environment variable "
                "INCLUDE ");
}
```

Output:

```
Cannot find environment variable INCLUDE
```

labs

Description: Calculates the absolute value of a long integer.

Include: `<stdlib.h>`

Prototype: `long labs(long i);`

Argument: `i` long integer value

Return Value: Returns the absolute value of `i`.

Remarks: A negative number is returned as positive; a positive number is unchanged.

Example:

```c
#include <stdio.h> /* for printf */
#include <stdlib.h> /* for labs */

int main(void)
{
    long i;

    i = 123456;
    printf("The absolute value of %ld is %ld\n", i, labs(i));

    i = -246834;
    printf("The absolute value of %ld is %ld\n", i, labs(i));

    i = 0;
    printf("The absolute value of %ld is %ld\n", i, labs(i));
}
```

Output:

```
The absolute value of 123456 is 123456
The absolute value of -246834 is 246834
The absolute value of 0 is 0
```
ldiv

Description: Calculates the quotient and remainder of two long integers.

Include: `<stdlib.h>`

Prototype: `ldiv_t ldiv(long numer, long denom);`

Arguments:
- `numer` numerator
- `denom` denominator

Return Value: Returns the quotient and the remainder.

Remarks: The returned quotient will have the same sign as the numerator divided by the denominator. The sign for the remainder will be such that the quotient times the denominator plus the remainder will equal the numerator `(quot * denom + rem = numer)`. If the denominator is zero, the behavior is undefined.

Example:
```c
#include <stdlib.h> /* for ldiv, ldiv_t */
#include <stdio.h>  /* for printf       */

int main(void)
{
    long x,y;
    ldiv_t z;

    x = 7;
    y = 3;
    printf("For ldiv(%ld, %ld)\n", x, y);
    z = ldiv(x, y);
    printf("The quotient is %ld and the "
           "remainder is %ld\n\n", z.quot, z.rem);

    x = 7;
    y = -3;
    printf("For ldiv(%ld, %ld)\n", x, y);
    z = ldiv(x, y);
    printf("The quotient is %ld and the "
           "remainder is %ld\n\n", z.quot, z.rem);

    x = -5;
    y = 3;
    printf("For ldiv(%ld, %ld)\n", x, y);
    z = ldiv(x, y);
    printf("The quotient is %ld and the "
           "remainder is %ld\n\n", z.quot, z.rem);

    x = 7;
    y = 7;
    printf("For ldiv(%ld, %ld)\n", x, y);
    z = ldiv(x, y);
    printf("The quotient is %ld and the "
           "remainder is %ld\n\n", z.quot, z.rem);

    x = 7;
    y = 0;
    printf("For ldiv(%ld, %ld)\n", x, y);
    z = ldiv(x, y);
    printf("The quotient is %ld and the "
           "remainder is %ld\n\n", z.quot, z.rem);
}
```
Idiv (Continued)

Output:

For ldiv(7, 3)
The quotient is 2 and the remainder is 1

For ldiv(7, -3)
The quotient is -2 and the remainder is 1

For ldiv(-5, 3)
The quotient is -1 and the remainder is -2

For ldiv(7, 7)
The quotient is 1 and the remainder is 0

For ldiv(7, 0)
The quotient is -1 and the remainder is 7

Explanation:
In the last example (ldiv(7, 0)) the denominator is zero, the behavior is undefined.

malloc
Description: Allocates memory.
Include: <stdlib.h>
Prototype: void *malloc(size_t size);
Argument: size number of characters to allocate
Return Value: Returns a pointer to the allocated space if successful; otherwise, returns a null pointer.
Remarks: malloc does not initialize memory it returns.
Example:
#include <stdio.h> /* for printf, sizeof, */
/* NULL */
#include <stdlib.h> /* for malloc, free */
int main(void)
{
  long *i;

  if ((i = (long *)malloc(50 * sizeof(long))) ==
      NULL)
    printf("Cannot allocate memory\n");
  else
  {
    printf("Memory allocated\n");
    free(i);
    printf("Memory freed\n");
  }
}

Output:
Memory allocated
Memory freed
**mblen**

**Description:** Gets the length of a multibyte character. (See Remarks.)

**Include:** `<stdlib.h>`

**Prototype:**

```c
int mblen(const char *s, size_t n);
```

**Arguments:**
- `s` points to the multibyte character
- `n` number of bytes to check

**Return Value:** Returns zero if `s` points to a null character; otherwise, returns 1.

**Remarks:** MPLAB C30 does not support multibyte characters with length greater than 1 byte.

---

**mbstowcs**

**Description:** Converts a multibyte string to a wide character string. (See Remarks.)

**Include:** `<stdlib.h>`

**Prototype:**

```c
size_t mbstowcs(wchar_t *wcs, const char *s, size_t n);
```

**Arguments:**
- `wcs` points to the wide character string
- `s` points to the multibyte string
- `n` the number of wide characters to convert.

**Return Value:** Returns the number of wide characters stored excluding the null character.

**Remarks:** `mbstowcs` converts `n` number of wide characters unless it encounters a null wide character first. MPLAB C30 does not support multibyte characters with length greater than 1 byte.

---

**mbtowc**

**Description:** Converts a multibyte character to a wide character. (See Remarks.)

**Include:** `<stdlib.h>`

**Prototype:**

```c
int mbtowc(wchar_t *pwc, const char *s, size_t n);
```

**Arguments:**
- `pwc` points to the wide character
- `s` points to the multibyte character
- `n` number of bytes to check

**Return Value:** Returns zero if `s` points to a null character; otherwise, returns 1

**Remarks:** The resulting wide character will be stored at `pwc`. MPLAB C30 does not support multibyte characters with length greater than 1 byte.
**qsort**

**Description:** Performs a quick sort.

**Include:** `<stdlib.h>`

**Prototype:**

```c
void qsort(void *base, size_t nelem, size_t size, 
           int (*cmp)(const void *e1, const void *e2));
```

**Arguments:**
- `base` pointer to the start of the array
- `nelem` number of elements
- `size` size of the elements
- `cmp` pointer to the comparison function
- `e1` pointer to the key for the search
- `e2` pointer to the element being compared with the key

**Remarks:** `qsort` overwrites the array with the sorted array. The comparison function is supplied by the user. In the following example, the list is sorted according to the comparison function. This `comp` uses ascending order.

**Example:**

```c
#include <stdlib.h> /* for qsort */
#include <stdio.h>  /* for printf */

#define NUM 7

int comp(const void *e1, const void *e2);

int main(void)
{
    int list[NUM] = {35, 47, 63, 25, 93, 16, 52};
    int x;

    printf("Unsorted List: ");
    for (x = 0; x < NUM; x++)
        printf("%d  ", list[x]);
    qsort(list, NUM, sizeof(int), comp);
    printf("\n");
    printf("Sorted List:   ");
    for (x = 0; x < NUM; x++)
        printf("%d  ", list[x]);
}

int comp(const void *e1, const void *e2)
{
    const int * a1 = e1;
    const int * a2 = e2;

    if (*a1 < *a2)
        return -1;
    else if (*a1 == *a2)
        return 0;
    else
        return 1;
}
```

**Output:**

Unsorted List: 35 47 63 25 93 16 52
Sorted List: 16 25 35 47 52 63 93
### rand

**Description:** Generates a pseudo-random integer.

**Include:** `<stdlib.h>`

**Prototype:**

```
int rand(void);
```

**Return Value:** Returns an integer between 0 and `RAND_MAX`.

**Remarks:** Calls to this function return pseudo-random integer values in the range [0,`RAND_MAX`]. To use this function effectively, you must seed the random number generator using the `srand` function. This function will always return the same sequence of integers when no seeds are used (as in the example below) or when identical seed values are used. (See `srand` for seed example.)

**Example:**

```c
#include <stdio.h>  /* for printf */
#include <stdlib.h> /* for rand */

int main(void)
{
    int x;

    for (x = 0; x < 5; x++)
        printf("Number = %d\n", rand());
}
```

**Output:**

```
Number = 21422
Number = 2061
Number = 16443
Number = 11617
Number = 9125
```

Notice if the program is run a second time, the numbers are the same. See the example for `srand` to seed the random number generator.

### realloc

**Description:** Reallocates memory to allow a size change.

**Include:** `<stdlib.h>`

**Prototype:**

```
void *realloc(void *ptr, size_t size);
```

**Arguments:**

- `ptr` points to previously allocated memory
- `size` new size to allocate to

**Return Value:** Returns a pointer to the allocated space if successful; otherwise, returns a null pointer.

**Remarks:** If the existing object is smaller than the new object, the entire existing object is copied to the new object and the remainder of the new object is indeterminate. If the existing object is larger than the new object, the function copies as much of the existing object as will fit in the new object. If `realloc` succeeds in allocating a new object, the existing object will be deallocated; otherwise, the existing object is left unchanged. Keep a temporary pointer to the existing object since `realloc` will return a null pointer on failure.
realloc (Continued)

Example:

```c
#include <stdio.h> /* for printf, sizeof, NULL */
#include <stdlib.h> /* for realloc, malloc, free */

int main(void)
{
    long *i, *j;

    if ((i = (long *)malloc(50 * sizeof(long))) == NULL)
        printf("Cannot allocate memory\n");
    else
    {
        printf("Memory allocated\n");
        /* Temp pointer in case realloc() fails */
        j = i;

        if ((i = (long *)realloc(i, 25 * sizeof(long))) == NULL)
            { 
                printf("Cannot reallocate memory\n");
                /* j pointed to allocated memory */
                free(j);
            }
        else
        {
            printf("Memory reallocated\n");
            free(i);
        }
    }
}
```

Output:
Memory allocated
Memory reallocated
**srand**

**Description:**
Set the starting seed for the pseudo-random number sequence.

**Include:**
<stdio.h>

**Prototype:**
void srand(unsigned int seed);

**Argument:**
seed  starting value for the pseudo-random number sequence

**Return Value:**
None

**Remarks:**
This function sets the starting seed for the pseudo-random number sequence generated by the rand function. The rand function will always return the same sequence of integers when identical seed values are used. If rand is called with a seed value of 1, the sequence of numbers generated will be the same as if rand had been called without srand having been called first.

**Example:**
```c
#include <stdio.h>  /* for printf      */
#include <stdlib.h> /* for rand, srand */

int main(void)
{
    int x;

    srand(7);
    for (x = 0; x < 5; x++)
        printf("Number = %d\n", rand());
}
```

**Output:**
Number = 16327
Number = 5931
Number = 23117
Number = 30985
Number = 29612

---

**strtod**

**Description:**
Converts a partial string to a floating-point number of type double.

**Include:**
<stdio.h>

**Prototype:**
double strtod(const char *s, char **endptr);

**Arguments:**

- **s** string to be converted
- **endptr** pointer to the character at which the conversion stopped

**Return Value:**
Returns the converted number if successful; otherwise, returns 0.

**Remarks:**
The number may consist of the following:

- [whitespace] [sign] digits [.digits]
  - [ e | E ][sign]digits

optional whitespace, followed by an optional sign, then a sequence of one or more digits with an optional decimal point, followed by one or more optional digits and an optional e or E followed by an optional signed exponent.

strtod converts the string until it reaches a character that cannot be converted to a number. endptr will point to the remainder of the string starting with the first unconverted character.

If a range error occurs, errno will be set.
Example:

```c
#include <stdio.h>  /* for printf */
#include <stdlib.h> /* for strtod */

int main(void)
{
    char *end;
    char a[] = "1.28 inches";
    char b[] = "27.835e2i";
    char c[] = "Number1";
    double x;

    x = strtod(a, &end);
    printf("String = \"%s\"  float = %f\n", a, x);
    printf("Stopped at: %s\n\n", end);

    x = strtod(b, &end);
    printf("String = \"%s\" float = %f\n", b, x);
    printf("Stopped at: %s\n\n", end);

    x = strtod(c, &end);
    printf("String = \"%s\"   float = %f\n", c, x);
    printf("Stopped at: %s\n\n", end);
}
```

Output:

```
String = "1.28 inches"  float = 1.280000
Stopped at: inches

String = "27.835e2i"  float = 2783.500000
Stopped at: i

String = "Number1"   float = 0.000000
Stopped at: Number1
```
strtol

**Description:** Converts a partial string to a long integer.

**Include:** `<stdlib.h>`

**Prototype:**
```
long strtol(const char *s, char **endptr, int base);
```

**Arguments:**
- `s` string to be converted
- `endptr` pointer to the character at which the conversion stopped
- `base` number base to use in conversion

**Return Value:** Returns the converted number if successful; otherwise, returns 0.

**Remarks:** If `base` is zero, `strtol` attempts to determine the base automatically. It can be octal, determined by a leading zero, hexadecimal, determined by a leading 0x or 0X, or decimal in any other case. If base is specified `strtol` converts a sequence of digits and letters a-z (case insensitive), where a-z represents the numbers 10-36. Conversion stops when an out of base number is encountered. `endptr` will point to the remainder of the string starting with the first unconverted character. If a range error occurs, `errno` will be set.

**Example:**
```
#include <stdio.h>  /* for printf */
#include <stdlib.h> /* for strtol */

int main(void)
{
    char *end;
    char a[] = "-12BGEE";
    char b[] = "1234Number";
    long x;

    x = strtol(a, &end, 16);
    printf("String = \"%s\"  long = %ld\n", a, x);
    printf("Stopped at: %s\n", end);

    x = strtol(b, &end, 4);
    printf("String = \"%s\"  long = %ld\n", b, x);
    printf("Stopped at: %s\n", end);
}
```

**Output:**
```
String = "-12BGEE"  long = -299
Stopped at: GEE

String = "1234Number"  long = 27
Stopped at: 4Number
```
**strtoul**

**Description:** Converts a partial string to an unsigned long integer.

**Include:** `<stdlib.h>`

**Prototype:**

```c
unsigned long strtoul(const char *s, char **endptr, int base);
```

**Arguments:**
- `s`  string to be converted
- `endptr`  pointer to the character at which the conversion stopped
- `base`  number base to use in conversion

**Return Value:** Returns the converted number if successful; otherwise, returns 0.

**Remarks:**
- If `base` is zero, `strtoul` attempts to determine the base automatically. It can be octal, determined by a leading zero, hexadecimal, determined by a leading 0x or 0X, or decimal in any other case. If base is specified, `strtoul` converts a sequence of digits and letters a-z (case insensitive), where a-z represents the numbers 10-36. Conversion stops when an out of base number is encountered. `endptr` will point to the remainder of the string starting with the first unconverted character.
- If a range error occurs, `errno` will be set.

**Example:**
```c
#include <stdio.h>  /* for printf */
#include <stdlib.h> /* for strtoul */

int main(void)
{
    char *end;
    char a[] = "12BGET3";
    char b[] = "0x1234Number";
    char c[] = "-123abc";
    unsigned long x;

    x = strtoul(a, &end, 25);
    printf("String = "a" long = %lu\n", a, x );
    printf("Stopped at: %s\n", end );

    x = strtoul(b, &end, 0);
    printf("String = "%s" long = %lu\n", b, x );
    printf("Stopped at: %s\n", end );

    x = strtoul(c, &end, 0);
    printf("String = "")a" long = %lu\n", c, x );
    printf("Stopped at: %s\n", end );
}
```

**Output:**
- String = "12BGET3" long = 429164
  Stopped at: T3

- String = "0x1234Number" long = 4660
  Stopped at: Number

- String = "-123abc" long = 4294967173
  Stopped at: abc
**system**

**Description:** Execute a command.

**Include:** `<stdlib.h>`

**Prototype:**
```c
int system(const char *s);
```

**Argument:**
- `s` command to be executed

**Remarks:** This function must be customized to be used as described (see pic30-libs). By default, `system` will cause a reset if called with anything other than `NULL`. `system(NULL)` will do nothing.

**Example:**
```c
/* This program uses system */
/* to TYPE its source file. */

#include <stdlib.h> /* for system */

int main(void)
{
    system("type sampsystem.c");
}
```

**Output:**
System(type sampsystem.c) called: Aborting

---

**wctomb**

**Description:** Converts a wide character to a multibyte character. (See Remarks.)

**Include:** `<stdlib.h>`

**Prototype:**
```c
int wctomb(char *s, wchar_t wchar);
```

**Arguments:**
- `s` points to the multibyte character
- `wchar` the wide character to be converted

**Return Value:** Returns zero if `s` points to a null character; otherwise, returns 1.

**Remarks:** The resulting multibyte character is stored at `s`. MPLAB C30 does not support multibyte characters with length greater than 1 character.

---

**wcstombs**

**Description:** Converts a wide character string to a multibyte string. (See Remarks.)

**Include:** `<stdlib.h>`

**Prototype:**
```c
size_t wcstombs(char *s, const wchar_t *wcs, size_t n);
```

**Arguments:**
- `s` points to the multibyte string
- `wcs` points to the wide character string
- `n` the number of characters to convert

**Return Value:** Returns the number of characters stored excluding the null character.

**Remarks:** `wcstombs` converts `n` number of multibyte characters unless it encounters a null character first. MPLAB C30 does not support multibyte characters with length greater than 1 character.
4.15 `<STRING.H>` STRING FUNCTIONS

The header file `string.h` consists of types, macros and functions that provide tools to manipulate strings.

<table>
<thead>
<tr>
<th><strong>size_t</strong></th>
<th>Description:</th>
<th>The type of the result of the <code>sizeof</code> operator.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td><code>&lt;string.h&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>NULL</strong></th>
<th>Description:</th>
<th>The value of a null pointer constant.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td><code>&lt;string.h&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>memchr</strong></th>
<th>Description:</th>
<th>Locates a character in a buffer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td><code>&lt;string.h&gt;</code></td>
<td></td>
</tr>
<tr>
<td>Prototype:</td>
<td><code>void *memchr(const void *s, int c, size_t n);</code></td>
<td></td>
</tr>
<tr>
<td>Arguments:</td>
<td><code>s</code> pointer to the buffer</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>c</code> character to search for</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>n</code> number of characters to check</td>
<td></td>
</tr>
<tr>
<td>Return Value:</td>
<td>Returns a pointer to the location of the match if successful; otherwise, returns null.</td>
<td></td>
</tr>
<tr>
<td>Remarks:</td>
<td><code>memchr</code> stops when it finds the first occurrence of <code>c</code> or after searching <code>n</code> number of characters.</td>
<td></td>
</tr>
<tr>
<td>Example:</td>
<td><code>#include &lt;string.h&gt;</code> /* for memchr, NULL */</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>#include &lt;stdio.h&gt;</code> /* for printf */</td>
<td></td>
</tr>
</tbody>
</table>

```c
int main(void)
{
    char buf1[50] = "What time is it?";
    char ch1 = 'i', ch2 = 'y';
    char *ptr;
    int res;

    printf("buf1 : %s\n\n", buf1);

    ptr = memchr(buf1, ch1, 50);
    if (ptr != NULL)
    {
        res = ptr - buf1 + 1;
        printf("%c found at position %d\n", ch1, res);
    }
    else
        printf("%c not found\n", ch1);
```
printf("
");

ptr = memchr(buf1, ch2, 50);
if (ptr != NULL)
{
    res = ptr - buf1 + 1;
    printf("%c found at position %d\n", ch2, res);
}
else
    printf("%c not found\n", ch2);
}

Output:
buf1 : What time is it?
i found at position 7
y not found

### memcmp

**Description:** Compare the contents of two buffers.

**Include:** `<string.h>`

**Prototype:**

```c
int memcmp(const void *s1, const void *s2, size_t n);
```

**Arguments:**

- `s1` first buffer
- `s2` second buffer
- `n` number of characters to compare

**Return Value:**
Returns a positive number if `s1` is greater than `s2`, zero if `s1` is equal to `s2`, or a negative number if `s1` is less than `s2`.

**Remarks:** This function compares the first `n` characters in `s1` to the first `n` characters in `s2` and returns a value indicating whether the buffers are less than, equal to or greater than each other.

**Example:**

```c
#include <string.h> /* memcmp */
#include <stdio.h>  /* for printf */

int main(void)
{
    char buf1[50] = "Where is the time?";
    char buf2[50] = "Where did they go?";
    char buf3[50] = "Why?";
    int res;

    printf("buf1 : %s\n", buf1);
    printf("buf2 : %s\n", buf2);
    printf("buf3 : %s\n", buf3);

    res = memcmp(buf1, buf2, 6);
    if (res < 0)
        printf("buf1 comes before buf2\n");
    else if (res == 0)
        printf("6 characters of buf1 and buf2 "
            "are equal\n");
    else
        printf("buf2 comes before buf1\n");

    return 0;
}
```
printf("\n");

res = memcmp(buf1, buf2, 20);
if (res < 0)
    printf("buf1 comes before buf2\n");
else if (res == 0)
    printf("20 characters of buf1 and buf2 "
    "are equal\n");
else
    printf("buf2 comes before buf1\n");

printf("\n");

res = memcmp(buf1, buf3, 20);
if (res < 0)
    printf("buf1 comes before buf3\n");
else if (res == 0)
    printf("20 characters of buf1 and buf3 "
    "are equal\n");
else
    printf("buf3 comes before buf1\n");
}

**Output:**
buf1 : Where is the time?
buf2 : Where did they go?
buf3 : Why?

6 characters of buf1 and buf2 are equal
buf2 comes before buf1
buf1 comes before buf3
memcpy

Description: Copies characters from one buffer to another.

Include: <string.h>

Prototype: void *memcpy(void *dst, const void *src, size_t n);

Arguments: dst buffer to copy characters to
src buffer to copy characters from
n number of characters to copy

Return Value: Returns dst.

Remarks: memcpy copies n characters from the source buffer src to the destination buffer dst. If the buffers overlap, the behavior is undefined.

Example:

```c
#include <string.h> /* memcpy */
#include <stdio.h>  /* for printf */

int main(void)
{
    char buf1[50] = "";
    char buf2[50] = "Where is the time?";
    char buf3[50] = "Why";

    printf("buf1 : %s\n", buf1);
    printf("buf2 : %s\n", buf2);
    printf("buf3 : %s\n", buf3);

    memcpy(buf1, buf2, 6);
    printf("buf1 after memcpy of 6 chars of "
            "buf2: \n\t%s\n", buf1);

    printf("\n");

    memcpy(buf1, buf3, 5);
    printf("buf1 after memcpy of 5 chars of "
            "buf3: \n\t%s\n", buf1);
}
```

Output:
buf1 :
buf2 : Where is the time?
buf3 : Why?

buf1 after memcpy of 6 chars of buf2:
    Where

buf1 after memcpy of 5 chars of buf3:
    Why?
memmove

Description: Copies n characters of the source buffer into the destination buffer, even if the regions overlap.

Include: `<string.h>

Prototype: `void *memmove(void *s1, const void *s2, size_t n);

Arguments: s1 buffer to copy characters to (destination) s2 buffer to copy characters from (source) n number of characters to copy from s2 to s1

Return Value: Returns a pointer to the destination buffer

Remarks: If the buffers overlap, the effect is as if the characters are read first from s2 then written to s1 so the buffer is not corrupted.

Example:
```c
#include <string.h> /* for memmove */
#include <stdio.h>  /* for printf */

int main(void)
{
    char buf1[50] = "When time marches on";
    char buf2[50] = "Where is the time?"
    char buf3[50] = "Why?"

    printf("buf1 : %s\n", buf1);
    printf("buf2 : %s\n", buf2);
    printf("buf3 : %s\n", buf3);

    memmove(buf1, buf2, 6);
    printf("buf1 after memmove of 6 chars of "
            "buf2: \n\t%s\n", buf1);
    printf("\n");

    memmove(buf1, buf3, 5);
    printf("buf1 after memmove of 5 chars of "
            "buf3: \n\t%s\n", buf1);
}
```

Output:
```
buf1 : When time marches on
buf2 : Where is the time?
buf3 : Why?

buf1 after memmove of 6 chars of buf2:
    Where ime marches on

buf1 after memmove of 5 chars of buf3:
    Why?
```
memset

Description: Copies the specified character into the destination buffer.

Include: <string.h>

Prototype: void *memset(void *s, int c, size_t n);

Arguments:
- s: buffer
- c: character to put in buffer
- n: number of times

Return Value: Returns the buffer with characters written to it.

Remarks: The character c is written to the buffer n times.

Example:

```c
#include <string.h> /* for memset */
#include <stdio.h>  /* for printf */

int main(void)
{
    char buf1[20] = "What time is it?";
    char buf2[20] = "";
    char ch1 = '?', ch2 = 'y';
    char *ptr;
    int res;

    printf("memset("%s", \'%c\',4);\n", buf1, ch1);
    memset(buf1, ch1, 4);
    printf("buf1 after memset: %s\n", buf1);

    printf("\n");
    printf("memset("%s", \'%c\',10);\n", buf2, ch2);
    memset(buf2, ch2, 10);
    printf("buf2 after memset: %s\n", buf2);
}
```

Output:

- `memset("What time is it?", '?',4);`
- `buf1 after memset: ????? time is it?`
- `memset("", 'y',10);`
- `buf2 after memset: yyyyyyyyyy`
### strcat

**Description:** Appends a copy of the source string to the end of the destination string.

**Include:**

```c
#include <string.h>
```

**Prototype:**

```c
char *strcat(char *s1, const char *s2);
```

**Arguments:**

- `s1` - null terminated destination string to copy to
- `s2` - null terminated source string to be copied

**Return Value:** Returns a pointer to the destination string.

**Remarks:** This function appends the source string (including the terminating null character) to the end of the destination string. The initial character of the source string overwrites the null character at the end of the destination string. If the buffers overlap, the behavior is undefined.

**Example:**

```c
#include <string.h> /* for strcat, strlen */
#include <stdio.h> /* for printf */

int main(void)
{
    char buf1[50] = "We're here";
    char buf2[50] = "Where is the time?";

    printf("buf1 : %s\n", buf1);
    printf("\t(%d characters)\n", strlen(buf1));
    printf("buf2 : %s\n", buf2);
    printf("\t(%d characters)\n", strlen(buf2));

    strcat(buf1, buf2);
    printf("buf1 after strcat of buf2: \n\t%s\n", buf1);
    printf("\t(%d characters)\n", strlen(buf1));

    printf("\n");

    strcat(buf1, "Why?");
    printf("buf1 after strcat of "Why?": \n\t%s\n", buf1);
    printf("\t(%d characters)\n", strlen(buf1));
}
```

**Output:**

- `buf1 : We're here
  (10 characters)`
- `buf2 : Where is the time?
  (18 characters)`
- `buf1 after strcat of buf2:
  We're here
  Where is the time?
  (28 characters)`
- `buf1 after strcat of "Why?":
  We're here
  Where is the time?
  Why?
  (32 characters)}`
### strchr

**Description:**
Locates the first occurrence of a specified character in a string.

**Include:**
<string.h>

**Prototype:**
char *strchr(const char *s, int c);

**Arguments:**
- s: pointer to the string
- c: character to search for

**Return Value:**
Returns a pointer to the location of the match if successful; otherwise, returns a null pointer.

**Remarks:**
This function searches the string `s` to find the first occurrence of the character `c`.

**Example:**
```c
#include <string.h> /* for strchr, NULL */
#include <stdio.h>  /* for printf       */

int main(void)
{
    char buf1[50] = "What time is it?";
    char ch1 = 'm', ch2 = 'y';
    char *ptr;
    int res;

    printf("buf1 : %s\n\n", buf1);
    ptr = strchr(buf1, ch1);
    if (ptr != NULL)
    {
        res = ptr - buf1 + 1;
        printf("%c found at position %d\n", ch1, res);
    }
    else
    {
        printf("%c not found\n", ch1);
    }

    printf("\n");
    ptr = strchr(buf1, ch2);
    if (ptr != NULL)
    {
        res = ptr - buf1 + 1;
        printf("%c found at position %d\n", ch2, res);
    }
    else
    {
        printf("%c not found\n", ch2);
    }

    Output:
    buf1 : What time is it?
    m found at position 8
    y not found
```
`strcmp`

**Description:** Compares two strings.

**Include:**

```c
#include <string.h>
```

**Prototype:**

```c
int strcmp(const char *s1, const char *s2);
```

**Arguments:**

- `s1`  first string
- `s2`  second string

**Return Value:** Returns a positive number if `s1` is greater than `s2`, zero if `s1` is equal to `s2`, or a negative number if `s1` is less than `s2`.

**Remarks:** This function compares successive characters from `s1` and `s2` until they are not equal or the null terminator is reached.

**Example:**

```c
#include <string.h> /* for strcmp */
#include <stdio.h>  /* for printf */

int main(void)
{
    char buf1[50] = "Where is the time?";
    char buf2[50] = "Where did they go?";
    char buf3[50] = "Why?";
    int res;

    printf("buf1 : %s\n", buf1);
    printf("buf2 : %s\n", buf2);
    printf("buf3 : %s\n", buf3);

    res = strcmp(buf1, buf2);
    if (res < 0)
        printf("buf1 comes before buf2\n");
    else if (res == 0)
        printf("buf1 and buf2 are equal\n");
    else
        printf("buf2 comes before buf1\n");

    printf("\n");

    res = strcmp(buf1, buf3);
    if (res < 0)
        printf("buf1 comes before buf3\n");
    else if (res == 0)
        printf("buf1 and buf3 are equal\n");
    else
        printf("buf3 comes before buf1\n");

    printf("\n");

    res = strcmp("Why?", buf3);
    if (res < 0)
        printf("\"Why?\" comes before buf3\n");
    else if (res == 0)
        printf("\"Why?\" and buf3 are equal\n");
    else
        printf("buf3 comes before \"Why?\"\n");
}
```
strcoll

Description:  Compares one string to another. (See Remarks.)
Include:    <string.h>
Prototype:  int strcoll(const char *s1, const char *s2);
Arguments:  s1  first string
            s2  second string
Return Value:  Using the locale-dependent rules, it returns a positive number if s1 is
greater than s2, zero if s1 is equal to s2, or a negative number if s1 is
less than s2.
Remarks:    Since MPLAB C30 does not support alternate locales, this function is
equivalent to strcmp.

strcpy

Description:  Copy the source string into the destination string.
Include:    <string.h>
Prototype:  char *strcpy(char *s1, const char *s2);
Arguments:  s1  destination string to copy to
            s2  source string to copy from
Return Value:  Returns a pointer to the destination string.
Remarks:    All characters of s2 are copied, including the null terminating character.
            If the strings overlap, the behavior is undefined.
Example:  #include <string.h> /* for strcpy, strlen */
          #include <stdio.h>  /* for printf         */
          int main(void)
          {
            char buf1[50] = "We're here";
            char buf2[50] = "Where is the time?";
            char buf3[50] = "Why?";
            printf("buf1 : %s\n", buf1);
            printf("buf2 : %s\n", buf2);
            printf("buf3 : %s\n", buf3);
            strcpy(buf1, buf2);
            printf("buf1 after strcpy of buf2: \n\t%s\n\n", buf1);
strcpy (Continued)

```c
strcpy(buf1, buf3);
printf("buf1 after strcpy of buf3: \n\t%s\n", buf1);
}
```

**Output:**

```
buf1 : We're here
buf2 : Where is the time?
buf3 : Why?

buf1 after strcpy of buf2:
  Where is the time?

buf1 after strcpy of buf3:
  Why?
```

---

### strcspn

<table>
<thead>
<tr>
<th>Description:</th>
<th>Calculate the number of consecutive characters at the beginning of a string that are not contained in a set of characters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td><code>&lt;string.h&gt;</code></td>
</tr>
<tr>
<td>Prototype:</td>
<td><code>size_t strcspn(const char *s1, const char *s2);</code></td>
</tr>
</tbody>
</table>
| Arguments:            | `s1` pointer to the string to be searched  
|                       | `s2` pointer to characters to search for                                                                            |
| Return Value:         | Returns the length of the segment in `s1` not containing characters found in `s2`.                                    |
| Remarks:              | This function will determine the number of consecutive characters from the beginning of `s1` that are not contained in `s2`. |
| Example:              | `#include <string.h> /* for strcspn */  
|                       | `#include <stdio.h>  /* for printf  */`                                                                               |

```c
int main(void)
{
   char str1[20] = "hello";
   char str2[20] = "aeiou";
   char str3[20] = "animal";
   char str4[20] = "xyz";
   int res;

   res = strcspn(str1, str2);
   printf("strcspn("%s\", "%s\") = %d\n", str1, str2, res);

   res = strcspn(str3, str2);
   printf("strcspn("%s\", "%s\") = %d\n", str3, str2, res);

   res = strcspn(str3, str4);
   printf("strcspn("%s\", "%s\") = %d\n", str3, str4, res);
}
```

**Output:**

```
strcspn("hello", "aeiou") = 1  
strcspn("animal", "aeiou") = 0  
strcspn("animal", "xyz") = 6
```
Explanation:
In the first result, e is in s2 so it stops counting after h.
In the second result, a is in s2.
In the third result, none of the characters of s1 are in s2 so all characters are counted.

**strerror**

**Description:** Gets an internal error message.

**Include:** `<string.h>`

**Prototype:** `char *strerror(int errcode);`

**Argument:**
- `errcode` number of the error code

**Return Value:** Returns a pointer to an internal error message string corresponding to the specified error code `errcode`.

**Remarks:** The array pointed to by `strerror` may be overwritten by a subsequent call to this function.

**Example:**
```c
#include <stdio.h>  /* for fopen, fclose, */
    /* printf, FILE, NULL */
#include <string.h> /* for strerror       */
#include <errno.h>  /* for errno          */

int main(void)
{
    FILE *myfile;
    if ((myfile = fopen("samp.fil", "r+")) == NULL)
        printf("Cannot open samp.fil: %s\n",
                strerror(errno));
    else
        printf("Success opening samp.fil\n");
    fclose(myfile);
}
```

**Output:**
Cannot open samp.fil: file open error

**strlen**

**Description:** Finds the length of a string.

**Include:** `<string.h>`

**Prototype:** `size_t strlen(const char *s);`

**Argument:**
- `s` the string

**Return Value:** Returns the length of a string.

**Remarks:** This function determines the length of the string, not including the terminating null character.
Example:

```c
#include <string.h> /* for strlen */
#include <stdio.h>  /* for printf */

int main(void)
{
    char str1[20] = "We are here";
    char str2[20] = "";
    char str3[20] = "Why me?";

    printf("str1 : %s\n", str1);
    printf("\t(string length = %d characters)\n", strlen(str1));
    printf("str2 : %s\n", str2);
    printf("\t(string length = %d characters)\n", strlen(str2));
    printf("str3 : %s\n", str3);
    printf("\t(string length = %d characters)\n", strlen(str3));
}
```

Output:

```
str1 : We are here
    (string length = 11 characters)
str2 :
    (string length = 0 characters)
str3 : Why me?
    (string length = 7 characters)
```

strncat

Description: Append a specified number of characters from the source string to the destination string.

Include: `<string.h>`

Prototype: `char *strncat(char *s1, const char *s2, size_t n);`

Arguments:

- `s1` destination string to copy to
- `s2` source string to copy from
- `n` number of characters to append

Return Value: Returns a pointer to the destination string.

Remarks: This function appends up to `n` characters (a null character and characters that follow it are not appended) from the source string to the end of the destination string. If a null character is not encountered, then a terminating null character is appended to the result. If the strings overlap, the behavior is undefined.

Example:

```c
#include <string.h> /* for strncat, strlen */
#include <stdio.h>  /* for printf */

int main(void)
{
    char buf1[50] = "We're here";
    char buf2[50] = "Where is the time?";
    char buf3[50] = "Why?";
```
# strncat (Continued)

```c
strncat(buf1, buf2, 6);
printf("buf1 after strncat of 6 characters of buf2: 
	%s
", buf1);
printf("(%d characters)\n\n", strlen(buf1));

strncat(buf1, buf2, 25);
printf("buf1 after strncat of 25 characters of buf2: 
	%s
", buf1);
printf("(%d characters)\n\n", strlen(buf1));

strncat(buf1, buf3, 4);
printf("buf1 after strncat of 4 characters of buf3: 
	%s
", buf1);
printf("(%d characters)\n\n", strlen(buf1));
}
```

**Output:**

buf1 : We're here

(10 characters)

buf2 : Where is the time?

(18 characters)

buf3 : Why?

(4 characters)

buf1 after strncat of 6 characters of buf2: We're hereWhere

(16 characters)

buf1 after strncat of 25 characters of buf2: We're hereWhere Where is the time?

(34 characters)

buf1 after strncat of 4 characters of buf3: We're hereWhere Where is the time?Why?

(38 characters)
strncmp

Description: Compare two strings, up to a specified number of characters.

Include: <string.h>

Prototype: int strncmp(const char *s1, const char *s2, size_t n);

Arguments: s1 first string
s2 second string
n number of characters to compare

Return Value: Returns a positive number if s1 is greater than s2, zero if s1 is equal to s2, or a negative number if s1 is less than s2.

Remarks: strncmp returns a value based on the first character that differs between s1 and s2. Characters that follow a null character are not compared.

Example:
#include <string.h> /* for strncmp */
#include <stdio.h>  /* for printf */

int main(void)
{
    char buf1[50] = "Where is the time?";
    char buf2[50] = "Where did they go?";
    char buf3[50] = "Why?";
    int res;

    printf("buf1 : %s\n", buf1);
    printf("buf2 : %s\n", buf2);
    printf("buf3 : %s\n\n", buf3);

    res = strncmp(buf1, buf2, 6);
    if (res < 0)
        printf("buf1 comes before buf2\n");
    else if (res == 0)
        printf("6 characters of buf1 and buf2 "
               "are equal\n");
    else
        printf("buf2 comes before buf1\n");

    printf("\n");

    res = strncmp(buf1, buf2, 20);
    if (res < 0)
        printf("buf1 comes before buf2\n");
    else if (res == 0)
        printf("20 characters of buf1 and buf2 "
               "are equal\n");
    else
        printf("buf2 comes before buf1\n");
}
# strncmp (Continued)

```c
printf("\n");

res = strncmp(buf1, buf3, 20);
if (res < 0)
    printf("buf1 comes before buf3\n");
else if (res == 0)
    printf("20 characters of buf1 and buf3 "
        "are equal\n");
else
    printf("buf3 comes before buf1\n");
}

Output:
buf1 : Where is the time?
buf2 : Where did they go?
buf3 : Why?

6 characters of buf1 and buf2 are equal
buf2 comes before buf1
buf1 comes before buf3
```

---

**strncpy**

**Description:** Copy characters from the source string into the destination string, up to the specified number of characters.

**Include:** `<string.h>`

**Prototype:**
```
char *strncpy(char *s1, const char *s2, size_t n);
```

**Arguments:**
- `s1` destination string to copy to
- `s2` source string to copy from
- `n` number of characters to copy

**Return Value:** Returns a pointer to the destination string.

**Remarks:** Copies `n` characters from the source string to the destination string. If the source string is less than `n` characters, the destination is filled with null characters to total `n` characters. If `n` characters were copied and no null character was found then the destination string will not be null-terminated. If the strings overlap, the behavior is undefined.

**Example:**
```
#include <string.h> /* for strncpy, strlen */
#include <stdio.h>  /* for printf          */

int main(void)
{
    char buf1[50] = "We're here";
    char buf2[50] = "Where is the time?";
    char buf3[50] = "Why?";
    char buf4[7] = "Where?";

    printf("buf1 : %s\n", buf1);
    printf("buf2 : %s\n", buf2);
    printf("buf3 : %s\n", buf3);
    printf("buf4 : %s\n", buf4);
```
strncpy (Continued)

strncpy(buf1, buf2, 6);
printf("buf1 after strncpy of 6 characters "
    "of buf2: \n\t%s\n", buf1);
printf("\t( %d characters)\n", strlen(buf1));

printf("\n");

strncpy(buf1, buf2, 18);
printf("buf1 after strncpy of 18 characters "
    "of buf2: \n\t%s\n", buf1);
printf("\t( %d characters)\n", strlen(buf1));

printf("\n");

strncpy(buf1, buf3, 5);
printf("buf1 after strncpy of 5 characters "
    "of buf3: \n\t%s\n", buf1);
printf("\t( %d characters)\n", strlen(buf1));

printf("\n");

strncpy(buf1, buf4, 9);
printf("buf1 after strncpy of 9 characters "
    "of buf4: \n\t%s\n", buf1);
printf("\t( %d characters)\n", strlen(buf1));
}

Output:
buf1 : We're here
buf2 : Where is the time?
buf3 : Why?
buf4 : Where?
buf1 after strncpy of 6 characters of buf2:
    Where here
        ( 10 characters)

buf1 after strncpy of 18 characters of buf2:
    Where is the time?
        ( 18 characters)

buf1 after strncpy of 5 characters of buf3:
    Why?
        ( 4 characters)

buf1 after strncpy of 9 characters of buf4:
    Where?
        ( 6 characters)
**strncpy (Continued)**

**Explanation:**
Each buffer contains the string shown, followed by null characters for a length of 50. Using `strlen` will find the length of the string up to but not including the first null character.

In the first example, 6 characters of `buf2` ("Where ") replace the first 6 characters of `buf1` ("We're ") and the rest of `buf1` remains the same ("here" plus null characters).

In the second example, 18 characters replace the first 18 characters of `buf1` and the rest remain null characters.

In the third example, 5 characters of `buf3` ("Why?" plus a null terminating character) replace the first 5 characters of `buf1`. `buf1` now actually contains ("Why?", 1 null character, " is the time?", 32 null characters). `strlen` shows 4 characters because it stops when it reaches the first null character.

In the fourth example, since `buf4` is only 7 characters `strncpy` uses 2 additional null characters to replace the first 9 characters of `buf1`. The result of `buf1` is 6 characters ("Where?") followed by 3 null characters, followed by 9 characters ("the time?"), followed by 32 null characters.

---

**strpbrk**

**Description:** Search a string for the first occurrence of a character from a specified set of characters.

**Include:** `<string.h>`

**Prototype:**
```c
char *strpbrk(const char *s1, const char *s2);
```

**Arguments:**
- `s1` pointer to the string to be searched
- `s2` pointer to characters to search for

**Return Value:** Returns a pointer to the matched character in `s1` if found; otherwise, returns a null pointer.

**Remarks:** This function will search `s1` for the first occurrence of a character contained in `s2`.

**Example:**
```c
#include <string.h> /* for strpbrk, NULL */
#include <stdio.h>  /* for printf        */

int main(void)
{
    char str1[20] = "What time is it?";
    char str2[20] = "xyz";
    char str3[20] = "eou?";
    char *ptr;
    int res;

    printf("strpbrk(""%s"", "%s")\n", str1, str2);
    ptr = strpbrk(str1, str2);
    if (ptr != NULL)
    {
        res = ptr - str1 + 1;
        printf("match found at position %d\n", res);
    }
    else
        printf("match not found\n");
}
```
strpbrk (Continued)

    printf("\n");
    printf("strpbrk("%s", "%s")\n", str1, str3);
    ptr = strpbrk(str1, str3);
    if (ptr != NULL)
    {
        res = ptr - str1 + 1;
        printf("match found at position %d\n", res);
    }
    else
    {
        printf("match not found\n");
    }

Output:
strpbrk("What time is it?", "xyz")
match not found
strpbrk("What time is it?", "eou?")
match found at position 9

strrchr

Description: Search for the last occurrence of a specified character in a string.
Include: <string.h>
Prototype: char *strrchr(const char *s, int c);
Arguments: s pointer to the string to be searched
            c character to search for
Return Value: Returns a pointer to the character if found; otherwise, returns a null pointer.
Remarks: The function searches the string s, including the terminating null character, to find the last occurrence of character c.
Example:
#include <string.h> /* for strrchr, NULL */
#include <stdio.h>  /* for printf        */
int main(void)
{
    char buf1[50] = "What time is it?";
    char ch1 = 'm', ch2 = 'y';
    char *ptr;
    int res;

    printf("buf1 : %s\n\n", buf1);
    ptr = strrchr(buf1, ch1);
    if (ptr != NULL)
    {
        res = ptr - buf1 + 1;
        printf("%c found at position %d\n", ch1, res);
    }
    else
    {
        printf("%c not found\n", ch1);
**strrchr (Continued)**

```c
printf("
");

ptr = strrchr(buf1, ch2);
if (ptr != NULL)
{
    res = ptr - buf1 + 1;
    printf("%c found at position %d\n", ch2, res);
}
else
    printf("%c not found\n", ch2);
```

**Output:**

buf1 : What time is it?

m found at position 8

y not found

---

### strspn

**Description:** Calculate the number of consecutive characters at the beginning of a string that are contained in a set of characters.

**Include:** `<string.h>`

**Prototype:**

```c
size_t strspn(const char *s1, const char *s2);
```

**Arguments:**

- `s1` pointer to the string to be searched
- `s2` pointer to characters to search for

**Return Value:** Returns the number of consecutive characters from the beginning of `s1` that are contained in `s2`.

**Remarks:** This function stops searching when a character from `s1` is not in `s2`.

**Example:**

```c
#include <string.h> /* for strspn */
#include <stdio.h>  /* for printf */

int main(void)
{
    char str1[20] = "animal";
    char str2[20] = "aeiounm";
    char str3[20] = "aimnl";
    char str4[20] = "xyz";
    int res;

    res = strspn(str1, str2);
    printf("strspn("%s", "%s") = %d\n", str1, str2, res);

    res = strspn(str1, str3);
    printf("strspn("%s", "%s") = %d\n", str1, str3, res);

    res = strspn(str1, str4);
    printf("strspn("%s", "%s") = %d\n", str1, str4, res);
}
```
### strspn (Continued)

**Output:**
- `strspn("animal", "aeiounm") = 5`
- `strspn("animal", "aimnl") = 6`
- `strspn("animal", "xyz") = 0`

**Explanation:**
- In the first result, l is not in s2.
- In the second result, the terminating null is not in s2.
- In the third result, a is not in s2, so the comparison stops.

### strstr

**Description:** Search for the first occurrence of a string inside another string.

**Include:** `<string.h>`

**Prototype:**
```
char *strstr(const char *s1, const char *s2);
```

**Arguments:**
- `s1` pointer to the string to be searched
- `s2` pointer to substring to be searched for

**Return Value:** Returns the address of the first element that matches the substring if found; otherwise, returns a null pointer.

**Remarks:** This function will find the first occurrence of the string s2 (excluding the null terminator) within the string s1. If s2 points to a zero length string, s1 is returned.

**Example:**
```c
#include <string.h> /* for strstr, NULL */
#include <stdio.h>  /* for printf       */

int main(void)
{
    char str1[20] = "What time is it?";
    char str2[20] = "is";
    char str3[20] = "xyz";
    char *ptr;
    int res;

    printf("str1 : %s
", str1);
    printf("str2 : %s
", str2);
    printf("str3 : %s
", str3);

    ptr = strstr(str1, str2);
    if (ptr != NULL)
    {
        res = ptr - str1 + 1;
        printf("\"s\" found at position %d\n", res);
    }
    else
        printf("\"s\" not found\n", str2);
    return 0;
}
```
```c
printf("\n");

ptr = strstr(str1, str3);
if (ptr != NULL)
{
    res = ptr - str1 + 1;
    printf(""%s" found at position %d\n",
            str3, res);
}
else
    printf(""%s" not found\n", str3);
}
```

**Output:**

- str1 : What time is it?
- str2 : is
- str3 : xyz

- "is" found at position 11
- "xyz" not found

---

### strtok

**Description:** Break a string into substrings, or tokens, by inserting null characters in place of specified delimiters.

**Include:**

```c
<string.h>
```

**Prototype:**

```c
char *strtok(char *s1, const char *s2);
```

**Arguments:**

- **s1** pointer to the null terminated string to be searched
- **s2** pointer to characters to be searched for (used as delimiters)

**Return Value:** Returns a pointer to the first character of a token (the first character in *s1* that does not appear in the set of characters of *s2*). If no token is found, the null pointer is returned.

**Remarks:** A sequence of calls to this function can be used to split up a string into substrings (or tokens) by replacing specified characters with null characters. The first time this function is invoked on a particular string, that string should be passed in *s1*. After the first time, this function can continue parsing the string from the last delimiter by invoking it with a null value passed in *s1*.

It skips all leading characters that appear in the string *s2* (delimiters), then skips all characters not appearing in *s2* (this segment of characters is the token), and then overwrites the next character with a null character, terminating the current token. The function *strtok* then saves a pointer to the character that follows, from which the next search will start. If *strtok* finds the end of the string before it finds a delimiter, the current token extends to the end of the string pointed to by *s1*. If this is the first call to *strtok*, it does not modify the string (no null characters are written to *s1*). The set of characters that is passed in *s2* need not be the same for each call to *strtok*.

If *strtok* is called with a non-null parameter for *s1* after the initial call, the string becomes the new string to search. The old string previously searched will be lost.
strtok (Continued)

Example:

```c
#include <string.h> /* for strtok, NULL */
#include <stdio.h> / * for printf       */

int main(void)
{
    char str1[30] = "Here, on top of the world!";
    char *word;
    int x;

    printf("str1 : %s\n", str1);
    x = 1;
    word = strtok(str1,delim);
    while (word != NULL)
    {
        printf("word %d: %s\n", x++, word);
        word = strtok(NULL, delim);
    }
}
```

Output:
str1 : Here, on top of the world!
word 1: Here
word 2: on
word 3: top
word 4: of
word 5: the
word 6: world!

strxfrm

Description: Transforms a string using the locale-dependent rules. (See Remarks.)

Include: ```<string.h>```

Prototype: ```size_t strxfrm(char *s1, const char *s2, size_t n);```

Arguments: ```s1``` destination string
```s2``` source string to be transformed
```n``` number of characters to transform

Return Value: Returns the length of the transformed string not including the terminating null character. If `n` is zero, the string is not transformed (```s1``` may be a point null in this case) and the length of ```s2``` is returned.

Remarks: If the return value is greater than or equal to `n`, the content of ```s1``` is indeterminate. Since MPLAB C30 does not support alternate locales, the transformation is equivalent to ```strcpy```, except that the length of the destination string is bounded by `n-1`. 
### 4.16 `<TIME.H>` DATE AND TIME FUNCTIONS

The header file `time.h` consists of types, macros and functions that manipulate time.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Include</th>
<th>Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>clock_t</strong></td>
<td>Stores processor time values.</td>
<td><code>&lt;time.h&gt;</code></td>
<td><code>typedef long clock_t</code></td>
</tr>
<tr>
<td><strong>size_t</strong></td>
<td>The type of the result of the <code>sizeof</code> operator.</td>
<td><code>&lt;time.h&gt;</code></td>
<td></td>
</tr>
<tr>
<td><strong>struct tm</strong></td>
<td>Structure used to hold the time and date (calendar time).</td>
<td><code>&lt;time.h&gt;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>struct tm {</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>int tm_sec; /* seconds after the minute (0 to 61) */</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>int tm_min; /* minutes after the hour (0 to 59) */</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>int tm_hour; /* hours since midnight (0 to 23) */</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>int tm_mday; /* day of month (1 to 31) */</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>int tm_mon; /* month (0 to 11 where January = 0) */</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>int tm_year; /* years since 1900 */</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>int tm_wday; /* day of week (0 to 6 where Sunday = 0) */</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>int tm_yday; /* day of year (0 to 365 where January 1 = 0) */</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>int tm_isdst; /* Daylight Savings Time flag */</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>}</code></td>
<td></td>
</tr>
<tr>
<td>Remarks</td>
<td>If <code>tm_isdst</code> is a positive value, Daylight Savings is in effect. If it is zero, Daylight Saving time is not in effect. If it is a negative value, the status of Daylight Saving Time is not known.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| **time_t** | Represents calendar time values. | `<time.h>` | `typedef long time_t` |

| **CLOCKS_PER_SEC** | Number of processor clocks per second. | `<time.h>` | `#define CLOCKS_PER_SEC 1` |
| Remarks | MPLAB C30 returns clock ticks (instruction cycles) not actual time. | |
**NULL**

**Description:** The value of a null pointer constant.

**Include:** `<time.h>`

---

**asctime**

**Description:** Converts the time structure to a character string.

**Include:** `<time.h>`

**Prototype:**

```c
char *asctime(const struct tm *tptr);
```

**Argument:**

- `tptr` time/date structure

**Return Value:**

Returns a pointer to a character string of the following format:

`DDD MMM dd hh:mm:ss YYYY`

- `DDD` is day of the week
- `MMM` is month of the year
- `dd` is day of the month
- `hh` is hour
- `mm` is minute
- `ss` is second
- `YYYY` is year

**Example:**

```c
#include <time.h>  /* for asctime, tm */
#include <stdio.h> /* for printf */

volatile int i;

int main(void)
{
    struct tm when;
    time_t whattime;

    when.tm_sec = 30;
    when.tm_min = 30;
    when.tm_hour = 2;
    when.tm_mday = 1;
    when.tm_mon = 1;
    when.tm_year = 103;

    whattime = mktime(&when);
    printf("Day and time is %s\n", asctime(&when));
}
```

**Output:**

`Day and time is Sat Feb 1 02:30:30 2003`

---

**clock**

**Description:** Calculates the processor time.

**Include:** `<time.h>`

**Prototype:**

```c
clock_t clock(void);
```

**Return Value:**

Returns the number of clock ticks of elapsed processor time.

**Remarks:**

If the target environment cannot measure elapsed processor time, the function returns -1, cast as a `clock_t` (i.e. `(clock_t)-1`). By default, MPLAB C30 returns the time as instruction cycles.
clock (Continued)

Example:
```c
#include <time.h> /* for clock */
#include <stdio.h> /* for printf */

volatile int i;

int main(void)
{
    clock_t start, stop;
    int ct;

    start = clock();
    for (i = 0; i < 10; i++)
        stop = clock();
    printf("start = %ld\n", start);
    printf("stop = %ld\n", stop);
}
```

Output:
start = 0
stop = 317

ctime

Description: Converts calendar time to a string representation of local time.
Include: `<time.h>`
Prototype: `char *ctime(const time_t *tod);`
Argument: `tod` pointer to stored time
Return Value: Returns the address of a string that represents the local time of the parameter passed.
Remarks: This function is equivalent to `asctime(localtime(tod));`.
Example:
```c
#include <time.h> /* for mktime, tm, ctime */
#include <stdio.h> /* for printf */

int main(void)
{
    time_t whattime;
    struct tm nowtime;

    nowtime.tm_sec = 30;
    nowtime.tm_min = 30;
    nowtime.tm_hour = 2;
    nowtime.tm_mday = 1;
    nowtime.tm_mon = 1;
    nowtime.tm_year = 103;

    whattime = mktime(&nowtime);
    printf("Day and time %s\n", ctime(&whattime));
}
```

Output:
Day and time Sat Feb  1 02:30:30 2003
### difftime

**Description:** Find the difference between two times.

**Include:** `<time.h>`

**Prototype:**

```c
double difftime(time_t t1, time_t t0);
```

**Arguments:**
- `t1` ending time
- `t0` beginning time

**Return Value:** Returns the number of seconds between `t1` and `t0`.

**Remarks:** By default, MPLAB C30 returns the time as instruction cycles so `difftime` returns the number of ticks between `t1` and `t0`.

**Example:**

```c
#include <time.h> /* for clock, difftime */
#include <stdio.h> /* for printf */

volatile int i;

int main(void)
{
    clock_t start, stop;
    double elapsed;

    start = clock();
    for (i = 0; i < 10; i++)
        stop = clock();
    printf("start = %ld\n", start);
    printf("stop = %ld\n", stop);
    elapsed = difftime(stop, start);
    printf("Elapsed time = %.0f\n", elapsed);
}
```

**Output:**

```
start = 0
stop = 317
Elapsed time = 317
```

### gmtime

**Description:** Converts calendar time to time structure expressed as Universal Time Coordinated (UTC) also known as Greenwich Mean Time (GMT).

**Include:** `<time.h>`

**Prototype:**

```c
struct tm *gmtime(const time_t *tod);
```

**Argument:** `tod` pointer to stored time

**Return Value:** Returns the address of the time structure.

**Remarks:** This function breaks down the `tod` value into the time structure of type `tm`. By default, MPLAB C30 returns the time as instruction cycles. With this default, `gmtime` and `localtime` will be equivalent except `gmtime` will return `tm_isdst` (Daylight Savings Time flag) as zero to indicate that Daylight Savings Time is not in effect.
Example:
```c
#include <time.h> /* for gmtime, asctime, */
   /* time_t, tm */
#include <stdio.h> /* for printf */

int main(void)
{
    time_t timer;
    struct tm *newtime;

    timer = 1066668182; /* Mon Oct 20 16:43:02 2003 */

    newtime = gmtime(&timer);
    printf("UTC time = %s\n", asctime(newtime));
}
Output:
UTC time = Mon Oct 20 16:43:02 2003
```

localtime

Description: Converts a value to the local time.
Include: `<time.h>`
Prototype: `struct tm *localtime(const time_t *tod);`
Argument: `tod` pointer to stored time
Return Value: Returns the address of the time structure.
Remarks: By default, MPLAB C30 returns the time as instruction cycles. With this default localtime and gmtime will be equivalent except localtime will return tm_isdst (Daylight Savings Time flag) as -1 to indicate that the status of Daylight Savings Time is not known.
Example:
```c
#include <time.h> /* for localtime, */
   /* asctime, time_t, tm */
#include <stdio.h> /* for printf */

int main(void)
{
    time_t timer;
    struct tm *newtime;

    timer = 1066668182; /* Mon Oct 20 16:43:02 2003 */

    newtime = localtime(&timer);
    printf("Local time = %s\n", asctime(newtime));
}
Output:
Local time = Mon Oct 20 16:43:02 2003
```
### mktime

**Description:** Converts local time to a calendar value.

**Include:**

```c
#include <time.h>
```

**Prototype:**

```c
time_t mktime(struct tm *tptr);
```

**Argument:**

- `tptr`  a pointer to the time structure

**Return Value:** Returns the calendar time encoded as a value of `time_t`.

**Remarks:**
If the calendar time cannot be represented, the function returns -1, cast as a `time_t` (i.e. `(time_t)-1`).

**Example:**

```c
#include <time.h>  /* for localtime, asctime, mktime, time_t, tm       */
#include <stdio.h> /* for printf       */

int main(void)
{
    time_t timer, whattime;
    struct tm *newtime;

    timer = 1066668182; /* Mon Oct 20 16:43:02 2003 */
    /* localtime allocates space for struct tm */
    newtime = localtime(&timer);
    printf("Local time = %s", asctime(newtime));

    whattime = mktime(newtime);
    printf("Calendar time as time_t = %ld\n", whattime);
}
```

**Output:**

```
Local time = Mon Oct 20 16:43:02 2003
Calendar time as time_t = 1066668182
```

### strftime

**Description:** Formats the time structure to a string based on the format parameter.

**Include:**

```c
#include <time.h>
```

**Prototype:**

```c
size_t strftime(char *s, size_t n, const char *format, const struct tm *tptr);
```

**Arguments:**

- `s`  output string
- `n`  maximum length of string
- `format`  format-control string
- `tptr`  pointer to tm data structure

**Return Value:** Returns the number of characters placed in the array `s` if the total including the terminating null is not greater than `n`. Otherwise, the function returns 0 and the contents of array `s` are indeterminate.

**Remarks:**
The format parameters follow:

- `%a`  abbreviated weekday name
- `%A`  full weekday name
- `%b`  abbreviated month name
- `%B`  full month name
- `%c`  appropriate date and time representation
- `%d`  day of the month (01-31)
- `%H`  hour of the day (00-23)
Example:

```c
#include <time.h> /* for strftime, */
    /* localtime,    */
    /* time_t, tm    */
#include <stdio.h> /* for printf    */

int main(void)
{
    time_t timer, whattime;
    struct tm *newtime;
    char buf[128];

    timer = 10666668182; /* Mon Oct 20 16:43:02 2003 */
    /* localtime allocates space for structure */
    newtime = localtime(&timer);

    strftime(buf, 128, "It was a %A, %d days into the "
             "month of %B in the year %Y.\n", newtime);
    printf(buf);

    strftime(buf, 128, "It was %W weeks into the year "
             "or %j days into the year.\n", newtime);
    printf(buf);
}
```

Output:

It was a Monday, 20 days into the month of October in the year 2003.
It was 42 weeks into the year or 293 days into the year.
time

Description: Calculates the current calendar time.

Include: <time.h>

Prototype: time_t time(time_t *tod);

Argument: tod pointer to storage location for time

Return Value: Returns the calendar time encoded as a value of time_t.

Remarks: If the target environment cannot determine the time, the function returns -1, cast as a time_t. By default, MPLAB C30 returns the time as instruction cycles. This function is customizable. See pic30-libs.

Example:
#include <time.h> /* for time */
#include <stdio.h> /* for printf */

volatile int i;

int main(void)
{
    time_t ticks;
    time(0); /* start time */
    for (i = 0; i < 10; i++) /* waste time */
        time(&ticks); /* get time */
    printf("Time = %ld\n", ticks);
}

Output:
Time = 256
4.17  <MATH.H> MATHEMATICAL FUNCTIONS

The header file math.h consists of a macro and various functions that calculate common mathematical operations. Error conditions may be handled with a domain error or range error (see errno.h).

A domain error occurs when the input argument is outside the domain over which the function is defined. The error is reported by storing the value of EDOM in errno and returning a particular value defined for each function.

A range error occurs when the result is too large or too small to be represented in the target precision. The error is reported by storing the value of ERANGE in errno and returning HUGE_VAL if the result overflowed (return value was too large) or a zero if the result underflowed (return value is too small).

Responses to special values, such as NaNs, zeros, and infinities, may vary depending upon the function. Each function description includes a definition of the function’s response to such values.

### HUGE_VAL

**Description:** HUGE_VAL is returned by a function on a range error (e.g., the function tries to return a value too large to be represented in the target precision).

**Include:** `<math.h>`

**Remarks:** -HUGE_VAL is returned if a function result is negative and is too large (in magnitude) to be represented in the target precision. When the printed result is +/- HUGE_VAL, it will be represented by +/- inf.

### acos

**Description:** Calculates the trigonometric arc cosine function of a double precision floating-point value.

**Include:** `<math.h>`

**Prototype:**

```c
double acos (double x);
```

**Argument:** x  value between -1 and 1 for which to return the arc cosine

**Return Value:** Returns the arc cosine in radians in the range of 0 to pi (inclusive).

**Remarks:** A domain error occurs if x is less than -1 or greater than 1.

**Example:**

```c
#include <math.h>  /* for acos           */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    double x,y;

    errno = 0;
    x = -2.0;
    y = acos (x);
    if (errno)
        perror("Error");
    printf("The arccosine of \%f is \%f\n\n", x, y);
}```
acos (Continued)

errno = 0;
x = 0.10;
y = acos (x);
if (errno)
    perror("Error");
    printf("The arccosine of %f is %f\n\n", x, y);
}

Output:
Error: domain error
The arccosine of -2.000000 is nan
The arccosine of 0.100000 is 1.470629

acosf

Description: Calculates the trigonometric arc cosine function of a single precision floating-point value.

Include: <math.h>

Prototype: float acosf (float x);

Argument: x  value between -1 and 1

Return Value: Returns the arc cosine in radians in the range of 0 to pi (inclusive).

Remarks: A domain error occurs if x is less than -1 or greater than 1.

Example:
#include <math.h> /* for acosf          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    float x, y;

    errno = 0;
    x = 2.0F;
y = acosf (x);
    if (errno)
        perror("Error");
        printf("The arccosine of %f is %f\n\n", x, y);

    errno = 0;
    x = 2.0F;
y = acosf (x);
    if (errno)
        perror("Error");
        printf("The arccosine of %f is %f\n\n", x, y);
}

Output:
Error: domain error
The arccosine of 2.000000 is nan
The arccosine of 0.000000 is 1.570796
## asin

**Description:** Calculates the trigonometric arc sine function of a double precision floating-point value.

**Include:**
```c
#include <math.h>
```

**Prototype:**
```c
double asin (double x);
```

**Argument:**
- `x` value between -1 and 1 for which to return the arc sine

**Return Value:** Returns the arc sine in radians in the range of -pi/2 to +pi/2 (inclusive).

**Remarks:** A domain error occurs if `x` is less than -1 or greater than 1.

**Example:**
```c
#include <math.h>  /* for asin           */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    double x, y;

    errno = 0;
    x = 2.0;
    y = asin (x);
    if (errno)
    {
        perror("Error");
        printf("The arcsine of %f is %f\n\n", x, y);
    }
    errno = 0;
    x = 0.0;
    y = asin (x);
    if (errno)
    {
        perror("Error");
        printf("The arcsine of %f is %f\n\n", x, y);
    }
}
```

**Output:**
- Error: domain error
- The arcsine of 2.000000 is nan
- The arcsine of 0.000000 is 0.000000

## asinf

**Description:** Calculates the trigonometric arc sine function of a single precision floating-point value.

**Include:**
```c
#include <math.h>
```

**Prototype:**
```c
float asinf (float x);
```

**Argument:**
- `x` value between -1 and 1

**Return Value:** Returns the arc sine in radians in the range of -pi/2 to +pi/2 (inclusive).

**Remarks:** A domain error occurs if `x` is less than -1 or greater than 1.

**Example:**
```c
#include <math.h>  /* for asinf          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    float x, y;
```
asinf (Continued)

```c
errno = 0;
x = 2.0F;
y = asinf(x);
if (errno)
    perror("Error");
printf("The arcsine of %f is %f\n\n", x, y);

errno = 0;
x = 0.0F;
y = asinf(x);
if (errno)
    perror("Error");
printf("The arcsine of %f is %f\n\n", x, y);
```

Output:
Error: domain error
The arcsine of 2.000000 is nan
The arcsine of 0.000000 is 0.000000

atan

**Description:** Calculates the trigonometric arc tangent function of a double precision floating-point value.

**Include:** `<math.h>`

**Prototype:**
```c
double atan (double x);
```

**Argument:** `x` value for which to return the arc tangent

**Return Value:** Returns the arc tangent in radians in the range of -pi/2 to +pi/2 (inclusive).

**Remarks:** No domain or range error will occur.

**Example:**
```c
#include <math.h> /* for atan   */
#include <stdio.h> /* for printf */

int main(void)
{
    double x, y;
    x = 2.0;
y = atan (x);
    printf("The arctangent of %f is %f\n\n", x, y);

    x = -1.0;
y = atan (x);
    printf("The arctangent of %f is %f\n\n", x, y);
}
```

Output:
The arctangent of 2.000000 is 1.107149
The arctangent of -1.000000 is -0.785398
atanf

Description: Calculates the trigonometric arc tangent function of a single precision floating-point value.

Include: <math.h>

Prototype: float atanf (float x);

Argument: x value for which to return the arc tangent

Return Value: Returns the arc tangent in radians in the range of -pi/2 to +pi/2 (inclusive).

Remarks: No domain or range error will occur.

Example:

```c
#include <math.h>  /* for atanf */
#include <stdio.h> /* for printf */

int main(void)
{
    float x, y;
    x = 2.0F;
    y = atanf (x);
    printf("The arctangent of %f is %f\n\n", x, y);
    x = -1.0F;
    y = atanf (x);
    printf("The arctangent of %f is %f\n\n", x, y);
}
```

Output:
The arctangent of 2.000000 is 1.107149
The arctangent of -1.000000 is -0.785398

atan2

Description: Calculates the trigonometric arc tangent function of y/x.

Include: <math.h>

Prototype: double atan2 (double y, double x);

Arguments: y y value for which to return the arc tangent
            x x value for which to return the arc tangent

Return Value: Returns the arc tangent in radians in the range of -pi to pi (inclusive) with the quadrant determined by the signs of both parameters.

Remarks: A domain error occurs if both x and y are zero or both x and y are +/- infinity.

Example:

```c
#include <math.h>  /* for atan2 */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    double x, y, z;
```
atan2 (Continued)

errno = 0;
x = 0.0;
y = 2.0;
z = atan2(y, x);
if (errno)
    perror("Error");
printf("The arctangent of %f/%f is %f\n\n",
       y, x, z);

errno = 0;
x = -1.0;
y = 0.0;
z = atan2(y, x);
if (errno)
    perror("Error");
printf("The arctangent of %f/%f is %f\n\n",
       y, x, z);

errno = 0;
x = 0.0;
y = 0.0;
z = atan2(y, x);
if (errno)
    perror("Error");
printf("The arctangent of %f/%f is %f\n\n",
       y, x, z);
}

Output:
The arctangent of 2.000000/0.000000 is 1.570796
The arctangent of 0.000000/-1.000000 is 3.141593
Error: domain error
The arctangent of 0.000000/0.000000 is nan
atan2f

Description: Calculates the trigonometric arc tangent function of y/x.

Include: `<math.h>

Prototype: float atan2f (float y, float x);

Arguments: y y value for which to return the arc tangent
x x value for which to return the arc tangent

Return Value: Returns the arc tangent in radians in the range of -pi to pi with the quadrant determined by the signs of both parameters.

Remarks: A domain error occurs if both x and y are zero or both x and y are +/- infinity.

Example:
```c
#include <math.h> /* for atan2f */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    float x, y, z;
    errno = 0;
    x = 2.0F;
    y = 0.0F;
    z = atan2f (y, x);
    if (errno)
        perror("Error");
    printf("The arctangent of %.4f/%.4f is %.6f\n", y, x, z);

    errno = 0;
    x = 0.0F;
    y = -1.0F;
    z = atan2f (y, x);
    if (errno)
        perror("Error");
    printf("The arctangent of %.4f/%.4f is %.6f\n", y, x, z);

    errno = 0;
    x = 0.0F;
    y = 0.0F;
    z = atan2f (y, x);
    if (errno)
        perror("Error");
    printf("The arctangent of %.4f/%.4f is nan\n", y, x, z);
}
```

Output:
The arctangent of 2.000000/0.000000 is 1.570796

The arctangent of 0.000000/-1.000000 is 3.141593

Error: domain error
The arctangent of 0.000000/0.000000 is nan
ceil

Description: Calculates the ceiling of a value.

Include: `<math.h>

Prototype: double ceil(double x);

Argument: x a floating-point value for which to return the ceiling.

Return Value: Returns the smallest integer value greater than or equal to x.

Remarks: No domain or range error will occur. See floor.

Example:
```c
#include <math.h> /* for ceil */
#include <stdio.h> /* for printf */

int main(void)
{
    double x[8] = {2.0, 1.75, 1.5, 1.25, -2.0, -1.75, -1.5, -1.25};
    double y;
    int i;

    for (i=0; i<8; i++)
    {
        y = ceil (x[i]);
        printf("The ceiling for %f is %f\n", x[i], y);
    }
}
```

Output:
The ceiling for 2.000000 is 2.000000
The ceiling for 1.750000 is 2.000000
The ceiling for 1.500000 is 2.000000
The ceiling for 1.250000 is 2.000000
The ceiling for -2.000000 is -2.000000
The ceiling for -1.750000 is -1.000000
The ceiling for -1.500000 is -1.000000
The ceiling for -1.250000 is -1.000000
ceilf

Description: Calculates the ceiling of a value.
Include: <math.h>
Prototype: float ceilf(float x);
Argument: x floating-point value.
Return Value: Returns the smallest integer value greater than or equal to x.
Remarks: No domain or range error will occur. See floorf.
Example:

```c
#include <math.h> /* for ceilf */
#include <stdio.h> /* for printf */

int main(void)
{
    float x[8] = {2.0F, 1.75F, 1.5F, 1.25F,
                  -2.0F, -1.75F, -1.5F, -1.25F};
    float y;
    int i;

    for (i=0; i<8; i++)
    {
        y = ceilf (x[i]);
        printf("The ceiling for \%f is \%f
", x[i], y);
    }
}

Output:
The ceiling for 2.000000 is 2.000000
The ceiling for 1.750000 is 2.000000
The ceiling for 1.500000 is 2.000000
The ceiling for 1.250000 is 2.000000
The ceiling for -2.000000 is -2.000000
The ceiling for -1.750000 is -1.000000
The ceiling for -1.500000 is -1.000000
The ceiling for -1.250000 is -1.000000
```

COS

Description: Calculates the trigonometric cosine function of a double precision floating-point value.
Include: <math.h>
Prototype: double cos (double x);
Argument: x value for which to return the cosine
Return Value: Returns the cosine of x in radians in the ranges of -1 to 1 inclusive.
Remarks: A domain error will occur if x is a NaN or infinity.
Example:

```c
#include <math.h> /* for cos */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    double x,y;

    errno = 0;
    x = -1.0;
    y = cos (x);
    if (errno)
        perror("Error");
    printf("The cosine of \%f is \%f\n", x, y);
}
```
```c
errno = 0;
x = 0.0;
y = cosf (x);
if (errno)
    perror("Error");
    printf("The cosine of %f is %f\n\n", x, y);
}
```

Output:
The cosine of -1.000000 is 0.540302

The cosine of 0.000000 is 1.000000

---

**cos**

Description: Calculates the trigonometric cosine function of a single precision floating-point value.

Include: `<math.h>`

Prototype: `float cosf (float x);`

Argument: `x` value for which to return the cosine

Return Value: Returns the cosine of `x` in radians in the ranges of -1 to 1 inclusive.

Remarks: A domain error will occur if `x` is a NaN or infinity.

Example:
```
#include <math.h> /* for cosf           */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    float x, y;

    errno = 0;
x = -1.0F;
y = cosf (x);
    if (errno)
        perror("Error");
        printf("The cosine of %f is %f\n\n", x, y);

    errno = 0;
x = 0.0F;
y = cosf (x);
    if (errno)
        perror("Error");
        printf("The cosine of %f is %f\n\n", x, y);
}
```

Output:
The cosine of -1.000000 is 0.540302

The cosine of 0.000000 is 1.000000
### cosh

**Description:** Calculates the hyperbolic cosine function of a double precision floating-point value.

**Include:**

```c
#include <math.h>
```

**Prototype:**

```c
double cosh (double x);
```

**Argument:**

`x` value for which to return the hyperbolic cosine

**Return Value:**

Returns the hyperbolic cosine of `x`

**Remarks:** A range error will occur if the magnitude of `x` is too large.

**Example:**

```c
#include <math.h> /* for cosh           */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    double x, y;

    errno = 0;
    x = -1.5;
    y = cosh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of %f is %f\n", x, y);

    errno = 0;
    x = 0.0;
    y = cosh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of %f is %f\n", x, y);

    errno = 0;
    x = 720.0;
    y = cosh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of %f is %f\n", x, y);
}
```

**Output:**

The hyperbolic cosine of -1.500000 is 2.352410

The hyperbolic cosine of 0.000000 is 1.000000

Error: range error
The hyperbolic cosine of 720.000000 is inf
coshf

Description: Calculates the hyperbolic cosine function of a single precision floating-point value.

Include: <math.h>

Prototype: float coshf (float x);

Argument: x value for which to return the hyperbolic cosine

Return Value: Returns the hyperbolic cosine of x

Remarks: A range error will occur if the magnitude of x is too large.

Example:

```c
#include <math.h> /* for coshf          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    float x, y;
    
    errno = 0;
    x = -1.0F;
    y = coshf (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of \%f is \%f\n\n", x, y);
    
    errno = 0;
    x = 0.0F;
    y = coshf (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of \%f is \%f\n\n", x, y);
    
    errno = 0;
    x = 720.0F;
    y = coshf (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of \%f is \%f\n\n", x, y);
}
```

Output:
The hyperbolic cosine of -1.000000 is 1.543081
The hyperbolic cosine of 0.000000 is 1.000000
Error: range error
The hyperbolic cosine of 720.000000 is inf
exp

Description: Calculates the exponential function of x (e raised to the power x where x is a double precision floating-point value).

Include: 
<math.h>

Prototype: 
double exp (double x);

Argument: 
x value for which to return the exponential

Return Value: 
Returns the exponential of x. On an overflow, exp returns inf and on an underflow exp returns 0.

Remarks: 
A range error occurs if the magnitude of x is too large.

Example: 
#include <math.h> /* for exp */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    double x, y;

    errno = 0;
    x = 1.0;
    y = exp (x);
    if (errno)
        perror("Error");
    printf("The exponential of %f is %f\n", x, y);

    errno = 0;
    x = 1E3;
    y = exp (x);
    if (errno)
        perror("Error");
    printf("The exponential of %f is %f\n", x, y);

    errno = 0;
    x = -1E3;
    y = exp (x);
    if (errno)
        perror("Error");
    printf("The exponential of %f is %f\n", x, y);
}

Output: 
The exponential of 1.000000 is 2.718282

Error: range error
The exponential of 1000.000000 is inf

Error: range error
The exponential of -1000.000000 is 0.000000
expf

Description: Calculates the exponential function of x (e raised to the power x where x is a single precision floating-point value).

Include: <math.h>

Prototype: float expf (float x);

Argument: x floating-point value for which to return the exponential

Return Value: Returns the exponential of x. On an overflow, expf returns inf and on an underflow exp returns 0.

Remarks: A range error occurs if the magnitude of x is too large.

Example:
#include <math.h> /* for expf */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    float x, y;

    errno = 0;
    x = 1.0F;
    y = expf (x);
    if (errno)
        perror("Error");
    printf("The exponential of \%f is \%f\n\n", x, y);

    errno = 0;
    x = 1.0E3F;
    y = expf (x);
    if (errno)
        perror("Error");
    printf("The exponential of \%f is \%f\n\n", x, y);

    errno = 0;
    x = -1.0E3F;
    y = expf (x);
    if (errno)
        perror("Error");
    printf("The exponential of \%f is \%f\n\n", x, y);
}

Output:
The exponential of 1.000000 is 2.718282

Error: range error
The exponential of 1000.000000 is inf

Error: range error
The exponential of -1000.000000 is 0.000000
fabs
Description: Calculates the absolute value of a double precision floating-point value.
Include: <math.h>
Prototype: double fabs(double x);
Argument: x floating-point value for which to return the absolute value
Return Value: Returns the absolute value of x. (A negative number is returned as positive, a positive number is unchanged.)
Remarks: No domain or range error will occur.
Example:
```c
#include <math.h> /* for fabs   */
#include <stdio.h> /* for printf */

int main(void)
{
    double x, y;

    x = 1.75;
    y = fabs (x);
    printf("The absolute value of %f is %f\n", x, y);

    x = -1.5;
    y = fabs (x);
    printf("The absolute value of %f is %f\n", x, y);
}
```
Output:
The absolute value of 1.750000 is 1.750000
The absolute value of -1.500000 is 1.500000

fabsf
Description: Calculates the absolute value of a single precision floating-point value.
Include: <math.h>
Prototype: float fabsf(float x);
Argument: x floating-point value for which to return the absolute value
Return Value: Returns the absolute value of x. (A negative number is returned as positive, a positive number is unchanged.)
Remarks: No domain or range error will occur.
Example:
```c
#include <math.h> /* for fabsf */
#include <stdio.h> /* for printf */

int main(void)
{
    float x,y;

    x = 1.75F;
    y = fabsf (x);
    printf("The absolute value of %f is %f\n", x, y);

    x = -1.5F;
    y = fabsf (x);
    printf("The absolute value of %f is %f\n", x, y);
}
```
Output:
The absolute value of 1.750000 is 1.750000
The absolute value of -1.500000 is 1.500000
**floor**

**Description:** Calculates the floor of a double precision floating-point value.

**Include:**

```c
#include <math.h>
```

**Prototype:**

```c
double floor (double x);
```

**Argument:**

\( x \) floating-point value for which to return the floor.

**Return Value:** Returns the largest integer value less than or equal to \( x \).

**Remarks:** No domain or range error will occur. See `ceil`.

**Example:**

```c
#include <math.h> /* for floor */
#include <stdio.h> /* for printf */

int main(void)
{
    double x[8] = {2.0, 1.75, 1.5, 1.25, -2.0, -1.75, -1.5, -1.25};
    double y;
    int i;

    for (i=0; i<8; i++)
    {
        y = floor (x[i]);
        printf("The floor for \%f is \%f\n", x[i], y);
    }
}
```

**Output:**

```
The floor for 2.000000 is 2.000000
The floor for 1.750000 is 1.000000
The floor for 1.500000 is 1.000000
The floor for 1.250000 is 1.000000
The floor for -2.000000 is -2.000000
The floor for -1.750000 is -2.000000
The floor for -1.500000 is -2.000000
The floor for -1.250000 is -2.000000
```

**floorf**

**Description:** Calculates the floor of a single precision floating-point value.

**Include:**

```c
#include <math.h>
```

**Prototype:**

```c
float floorf(float x);
```

**Argument:**

\( x \) floating-point value.

**Return Value:** Returns the largest integer value less than or equal to \( x \).

**Remarks:** No domain or range error will occur. See `ceilf`. 
Example:

```c
#include <math.h> /* for floorf */
#include <stdio.h> /* for printf */

int main(void)
{
    float x[8] = {2.0F, 1.75F, 1.5F, 1.25F,
                  -2.0F, -1.75F, -1.5F, -1.25F};
    float y;
    int i;

    for (i=0; i<8; i++)
    {
        y = floorf(x[i]);
        printf("The floor for %f is %f\n", x[i], y);
    }
}
```

Output:
The floor for 2.000000 is 2.000000
The floor for 1.750000 is 1.000000
The floor for 1.500000 is 1.000000
The floor for 1.250000 is 1.000000
The floor for -2.000000 is -2.000000
The floor for -1.750000 is -2.000000
The floor for -1.500000 is -2.000000
The floor for -1.250000 is -2.000000

**fmod**

**Description:** Calculates the remainder of x/y as a double precision value.

**Include:**
```c
#include <math.h>
```

**Prototype:**
```c
double fmod(double x, double y);
```

**Arguments:**
- `x`: a double precision floating-point value.
- `y`: a double precision floating-point value.

**Return Value:** Returns the remainder of `x` divided by `y`.

**Remarks:** If `y` = 0, a domain error occurs. If `y` is non-zero, the result will have the same sign as `x` and the magnitude of the result will be less than the magnitude of `y`.

**Example:**

```c
#include <math.h> /* for fmod */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    double x, y, z;

    errno = 0;
    x = 7.0;
    y = 3.0;
    z = fmod(x, y);
    if (errno)
        perror("Error");
    printf("For fmod(%f, %f) the remainder is %f\n\n", x, y, z);
```

floorf (Continued)
fmod (Continued)

erno = 0;
x = 7.0;
y = 7.0;
z = fmod(x, y);
if (errno)
    perror("Error");
printf("For fmod(%f, %f) the remainder is %f\n\n",
    x, y, z);

errno = 0;
x = -5.0;
y = 3.0;
z = fmod(x, y);
if (errno)
    perror("Error");
printf("For fmod(%f, %f) the remainder is %f\n\n",
    x, y, z);

errno = 0;
x = 5.0;
y = -3.0;
z = fmod(x, y);
if (errno)
    perror("Error");
printf("For fmod(%f, %f) the remainder is %f\n\n",
    x, y, z);

errno = 0;
x = -5.0;
y = -5.0;
z = fmod(x, y);
if (errno)
    perror("Error");
printf("For fmod(%f, %f) the remainder is %f\n\n",
    x, y, z);

errno = 0;
x = 7.0;
y = 0.0;
z = fmod(x, y);
if (errno)
    perror("Error");
printf("For fmod(%f, %f) the remainder is %f\n\n",
    x, y, z);
}

Output:
For fmod(7.000000, 3.000000) the remainder is 1.000000
For fmod(7.000000, 7.000000) the remainder is 0.000000
For fmod(-5.000000, 3.000000) the remainder is -2.000000
For fmod(5.000000, -3.000000) the remainder is 2.000000
For fmod(-5.000000, -5.000000) the remainder is -0.000000
Error: domain error
For fmod(7.000000, 0.000000) the remainder is nan
**fmodf**

**Description:** Calculates the remainder of x/y as a single precision value.

**Include:**

`<math.h>`

**Prototype:**

`float fmodf(float x, float y);`

**Arguments:**

- `x` a single precision floating-point value
- `y` a single precision floating-point value

**Return Value:** Returns the remainder of `x` divided by `y`.

**Remarks:** If `y = 0`, a domain error occurs. If `y` is non-zero, the result will have the same sign as `x` and the magnitude of the result will be less than the magnitude of `y`.

**Example:**

```c
#include <math.h> /* for fmodf */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    float x, y, z;

    errno = 0;
    x = 7.0F;
    y = 3.0F;
    z = fmodf(x, y);
    if (errno)
        perror("Error");
    printf("For fmodf (%f, %f) the remainder is" " %f\n"
"     , x, y, z);

    errno = 0;
    x = -5.0F;
    y = 3.0F;
    z = fmodf(x, y);
    if (errno)
        perror("Error");
    printf("For fmodf (%f, %f) the remainder is" " %f\n"
"     , x, y, z);

    errno = 0;
    x = 5.0F;
    y = -3.0F;
    z = fmodf(x, y);
    if (errno)
        perror("Error");
    printf("For fmodf (%f, %f) the remainder is" " %f\n"
"     , x, y, z);

    errno = 0;
    x = 5.0F;
    y = -5.0F;
    z = fmodf(x, y);
    if (errno)
        perror("Error");
    printf("For fmodf (%f, %f) the remainder is" " %f\n"
"     , x, y, z);
}
```
fmodf (Continued)

    errno = 0;
    x = 7.0F;
    y = 0.0F;
    z = fmodf (x, y);
    if (errno)
        perror("Error");
    printf("For fmodf (%f, %f) the remainder is"
           " %f\n\n", x, y, z);

    errno = 0;
    x = 7.0F;
    y = 7.0F;
    z = fmodf (x, y);
    if (errno)
        perror("Error");
    printf("For fmodf (%f, %f) the remainder is"
           " %f\n\n", x, y, z);
}

Output:
For fmodf (7.000000, 3.000000) the remainder is 1.000000
For fmodf (-5.000000, 3.000000) the remainder is -2.000000
For fmodf (5.000000, -3.000000) the remainder is 2.000000
For fmodf (5.000000, -5.000000) the remainder is 0.000000
Error: domain error
For fmodf (7.000000, 0.000000) the remainder is nan
For fmodf (7.000000, 7.000000) the remainder is 0.000000

frexp

Description: Gets the fraction and the exponent of a double precision floating-point number.

Include: <math.h>

Prototype: double frexp (double x, int *exp);

Arguments: x floating-point value for which to return the fraction and exponent
            *exp pointer to a stored integer exponent

Return Value: Returns the fraction, exp points to the exponent. If x is 0, the function returns 0 for both the fraction and exponent.

Remarks: The absolute value of the fraction is in the range of 1/2 (inclusive) to 1 (exclusive). No domain or range error will occur.

Example:
    #include <math.h> /* for frexp */
    #include <stdio.h> /* for printf */

    int main(void)
    {
        double x,y;
        int n;
frexp (Continued)

```c
x = 50.0;
y = frexp (x, &n);
printf("For frexp of %f\n the fraction is %f\n ",
        x, y);
printf(" and the exponent is %d\n", n);

x = -2.5;
y = frexp (x, &n);
printf("For frexp of %f\n the fraction is %f\n ",
        x, y);
printf(" and the exponent is %d\n", n);

x = 0.0;
y = frexp (x, &n);
printf("For frexp of %f\n the fraction is %f\n ",
        x, y);
printf(" and the exponent is %d\n", n);
```

Output:

For frexp of 50.000000
  the fraction is 0.781250
  and the exponent is 6

For frexp of -2.500000
  the fraction is -0.625000
  and the exponent is 2

For frexp of 0.000000
  the fraction is 0.000000
  and the exponent is 0

frexpf

**Description:** Gets the fraction and the exponent of a single precision floating-point number.

**Include:** `<math.h>`

**Prototype:**

```c
float frexpf (float x, int *exp);
```

**Arguments:**

- `x` floating-point value for which to return the fraction and exponent
- `*exp` pointer to a stored integer exponent

**Return Value:** Returns the fraction, `exp` points to the exponent. If `x` is 0, the function returns 0 for both the fraction and exponent.

**Remarks:** The absolute value of the fraction is in the range of 1/2 (inclusive) to 1 (exclusive). No domain or range error will occur.

**Example:**

```c
#include <math.h> /* for frexpf */
#include <stdio.h> /* for printf */

int main(void)
{
    float x,y;
    int n;
```
frexp (Continued)

```c
x = 0.15F;
y = frexp (x, &n);
printf("For frexp of %f\n the fraction is %f\n ",
       x, y);
printf(" and the exponent is %d\n", n);
```

```c
x = -2.5F;
y = frexp (x, &n);
printf("For frexp of %f\n the fraction is %f\n ",
       x, y);
printf(" and the exponent is %d\n", n);
```

```c
x = 0.0F;
y = frexp (x, &n);
printf("For frexp of %f\n the fraction is %f\n ",
       x, y);
printf(" and the exponent is %d\n", n);
```}

**Output:**

For frexp of 0.150000
the fraction is 0.600000
and the exponent is -2

For frexp of -2.500000
the fraction is -0.625000
and the exponent is 2

For frexp of 0.000000
the fraction is 0.000000
and the exponent is 0

### ldexp

**Description:** Calculates the result of a double precision floating-point number multiplied by an exponent of 2.

**Include:** `<math.h>`

**Prototype:**

```c
double ldexp(double x, int ex);
```

**Arguments:**

- `x` floating-point value
- `ex` integer exponent

**Return Value:** Returns `x * 2^ex`. On an overflow, `ldexp` returns `inf` and on an underflow, `ldexp` returns 0.

**Remarks:** A range error will occur on overflow or underflow.

**Example:**

```c
#include <math.h> /* for ldexp          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    double x,y;
    int n;
```
Idexp (Continued)

```
errno = 0;
x = -0.625;
n = 2;
y = ldexp (x, n);
if (errno)
    perror("Error");
printf("For a number = %f and an exponent = %d\n", 
x, n);
printf("  ldexp(%f, %d) = %f\n\n", 
x, n, y);

errno = 0;
x = 2.5;
n = 3;
y = ldexp (x, n);
if (errno)
    perror("Error");
printf("For a number = %f and an exponent = %d\n", 
x, n);
printf("  ldexp(%f, %d) = %f\n\n", 
x, n, y);

errno = 0;
x = 15.0;
n = 10000;
y = ldexp (x, n);
if (errno)
    perror("Error");
printf("For a number = %f and an exponent = %d\n", 
x, n);
printf("  ldexp(%f, %d) = %f\n\n", 
x, n, y);
}
```

Output:
For a number = -0.625000 and an exponent = 2
ldexp(-0.625000, 2) = -2.500000

For a number = 2.500000 and an exponent = 3
ldexp(2.500000, 3) = 20.000000

Error: range error
For a number = 15.000000 and an exponent = 10000
ldexp(15.000000, 10000) = inf

Idexpf

Description: Calculates the result of a single precision floating-point number multiplied by an exponent of 2.

Include: <math.h>

Prototype: float ldexpf(float x, int ex);

Arguments: 
x floating-point value
ex integer exponent

Return Value: Returns \(x \times 2^{ex}\). On an overflow, ldexp returns inf and on an underflow, ldexp returns 0.
Idexpf (Continued)

Remarks: A range error will occur on overflow or underflow.

Example:
#include <math.h> /* for ldexpf */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    float x, y;
    int n;

    errno = 0;
    x = -0.625F;
    n = 2;
    y = ldexpf (x, n);
    if (errno)
        perror("Error");
    printf("For a number = %f and an exponent = %d\n", 
            x, n);
    printf("  ldexpf(%f, %d) = %f\n
", 
            x, n, y);

    errno = 0;
    x = 2.5F;
    n = 3;
    y = ldexpf (x, n);
    if (errno)
        perror("Error");
    printf("For a number = %f and an exponent = %d\n", 
            x, n);
    printf("  ldexpf(%f, %d) = %f\n
", 
            x, n, y);

    errno = 0;
    x = 15.0F;
    n = 10000;
    y = ldexpf (x, n);
    if (errno)
        perror("Error");
    printf("For a number = %f and an exponent = %d\n", 
            x, n);
    printf("  ldexpf(%f, %d) = %f\n
", 
            x, n, y);
}

Output:
For a number = -0.625000 and an exponent = 2
  ldexpf(-0.625000, 2) = -2.500000

For a number = 2.500000 and an exponent = 3
  ldexpf(2.500000, 3) = 20.000000

Error: range error
For a number = 15.000000 and an exponent = 10000
  ldexpf(15.000000, 10000) = inf
### log

**Description:** Calculates the natural logarithm of a double precision floating-point value.

**Include:**

```
<math.h>
```

**Prototype:**

```
double log(double x);
```

**Argument:**

`x` — any positive value for which to return the log

**Return Value:**

Returns the natural logarithm of `x`. `-inf` is returned if `x` is 0 and NaN is returned if `x` is a negative number.

**Remarks:**

A domain error occurs if `x ≤ 0`.

**Example:**

```c
#include <math.h>  /* for log            */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    double x, y;

    errno = 0;
    x = 2.0;
    y = log (x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of \f is %f\n\n", 
            x, y);

    errno = 0;
    x = 0.0;
    y = log (x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of \f is %f\n\n", 
            x, y);

    errno = 0;
    x = -2.0;
    y = log (x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of \f is %f\n\n", 
            x, y);
}
```

**Output:**

```
The natural logarithm of 2.000000 is 0.693147

The natural logarithm of 0.000000 is -inf

Error: domain error
The natural logarithm of -2.000000 is nan
```
log10

Description: Calculates the base-10 logarithm of a double precision floating-point value.

Include: `<math.h>`

Prototype: `double log10(double x);`

Argument: `x` any double precision floating-point positive number

Return Value: Returns the base-10 logarithm of `x`. `-inf` is returned if `x` is 0 and `NaN` is returned if `x` is a negative number.

Remarks: A domain error occurs if `x` ≤ 0.

Example:

```c
#include <math.h> /* for log10          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    double x, y;

    errno = 0;
    x = 2.0;
    y = log10 (x);
    if (errno)
        perror("Error");
    printf("The base-10 logarithm of %f is %f\n", x, y);

    errno = 0;
    x = 0.0;
    y = log10 (x);
    if (errno)
        perror("Error");
    printf("The base-10 logarithm of %f is %f\n", x, y);

    errno = 0;
    x = -2.0;
    y = log10 (x);
    if (errno)
        perror("Error");
    printf("The base-10 logarithm of %f is %f\n", x, y);
}
```

Output:
The base-10 logarithm of 2.000000 is 0.301030

The base-10 logarithm of 0.000000 is -inf

Error: domain error
The base-10 logarithm of -2.000000 is nan
**log10f**

**Description:** Calculates the base-10 logarithm of a single precision floating-point value.

**Include:** `<math.h>`

**Prototype:**

```c
float log10f(float x);
```

**Argument:**

- `x` any single precision floating-point positive number

**Return Value:** Returns the base-10 logarithm of `x`. `-inf` is returned if `x` is 0 and `NaN` is returned if `x` is a negative number.

**Remarks:** A domain error occurs if `x` ≤ 0.

**Example:**

```c
#include <math.h> /* for log10f         */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    float x, y;

    errno = 0;
    x = 2.0F;
    y = log10f(x);
    if (errno)
        perror("Error");
    printf("The base-10 logarithm of %f is %f\n\n", x, y);

    errno = 0;
    x = 0.0F;
    y = log10f(x);
    if (errno)
        perror("Error");
    printf("The base-10 logarithm of %f is %f\n\n", x, y);

    errno = 0;
    x = -2.0F;
    y = log10f(x);
    if (errno)
        perror("Error");
    printf("The base-10 logarithm of %f is %f\n\n", x, y);
}
```

**Output:**

```
The base-10 logarithm of 2.000000 is 0.301030

Error: domain error
The base-10 logarithm of 0.000000 is -inf

Error: domain error
The base-10 logarithm of -2.000000 is nan
```
**logf**

**Description:** Calculates the natural logarithm of a single precision floating-point value.

**Include:**  
#include <math.h>

**Prototype:**  
float logf(float x);

**Argument:**  
x any positive value for which to return the log

**Return Value:**  
Returns the natural logarithm of x. -inf is returned if x is 0 and NaN is returned if x is a negative number.

**Remarks:**  
A domain error occurs if x ≤ 0.

**Example:**  
```
#include <math.h> /* for logf           */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    float x, y;

    errno = 0;
    x = 2.0F;
    y = logf (x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of %f is %f\n", x, y);

    errno = 0;
    x = 0.0F;
    y = logf (x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of %f is %f\n", x, y);

    errno = 0;
    x = -2.0F;
    y = logf (x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of %f is %f\n", x, y);
}
```

**Output:**
The natural logarithm of 2.000000 is 0.693147

The natural logarithm of 0.000000 is -inf

Error: domain error
The natural logarithm of -2.000000 is nan
### modf

**Description:** Splits a double precision floating-point value into fractional and integer parts.

**Include:**

```c
#include <math.h>
```

**Prototype:**

```c
double modf(double x, double *pint);
```

**Arguments:**

- `x` double precision floating-point value
- `pint` pointer to a stored the integer part

**Return Value:** Returns the signed fractional part and `pint` points to the integer part.

**Remarks:** The absolute value of the fractional part is in the range of 0 (inclusive) to 1 (exclusive). No domain or range error will occur.

**Example:**

```c
#include <math.h> /* for modf */
#include <stdio.h> /* for printf */

int main(void)
{
    double x, y, n;

    x = 0.707;
    y = modf (x, &n);
    printf("For %.1f the fraction is %f \n ", x, y);
    printf(" and the integer is %0.1f \n ", n);

    x = -15.2121;
    y = modf (x, &n);
    printf("For %.1f the fraction is %f \n ", x, y);
    printf(" and the integer is %0.1f \n ", n);
}
```

**Output:**

For 0.707000 the fraction is 0.707000
and the integer is 0

For -15.212100 the fraction is -0.212100
and the integer is -15
modff

Description:  Splits a single precision floating-point value into fractional and integer parts.

Include:     <math.h>

Prototype:   float modff(float x, float *pint);

Arguments:   x       single precision floating-point value
             pint    pointer to stored integer part

Return Value: Returns the signed fractional part and pint points to the integer part.

Remarks:     The absolute value of the fractional part is in the range of 0 (inclusive) to 1 (exclusive). No domain or range error will occur.

Example:

```c
#include <math.h>  /* for modff */
#include <stdio.h> /* for printf */

int main(void)
{
    float x,y,n;
    x = 0.707F;
    y = modff (x, &n);
    printf("For %f the fraction is %f\n " , x, y);
    printf(" and the integer is %0.f\n", n);

    x = -15.2121F;
    y = modff (x, &n);
    printf("For %f the fraction is %f\n ", x, y);
    printf(" and the integer is %0.f\n", n);
}
```

Output:

For 0.707000 the fraction is 0.707000
    and the integer is 0

For -15.212100 the fraction is -0.212100
    and the integer is -15
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Include</th>
<th>Prototype</th>
<th>Arguments</th>
<th>Return Value</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pow</code></td>
<td>Calculates x raised to the power y.</td>
<td><code>&lt;math.h&gt;</code></td>
<td><code>double pow(double x, double y);</code></td>
<td>x: the base, y: the exponent</td>
<td>Returns x raised to the power y ($x^y$).</td>
<td>If y is 0, <code>pow</code> returns 1. If x is 0.0 and y is less than 0, <code>pow</code> returns <code>inf</code> and a domain error occurs. If the result overflows or underflows, a range error occurs.</td>
</tr>
</tbody>
</table>

**Example:**
```c
#include <math.h>  /* for pow            */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    double x,y,z;
    errno = 0;
    x = -2.0;
    y = 3.0;
    z = pow (x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n\n", x, y, z);

    errno = 0;
    x = 3.0;
    y = -0.5;
    z = pow (x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n\n", x, y, z);

    errno = 0;
    x = 4.0;
    y = 0.0;
    z = pow (x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n\n", x, y, z);

    errno = 0;
    x = 0.0;
    y = -3.0;
    z = pow (x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n\n", x, y, z);
}
```

**Output:**
- `-2.000000 raised to 3.000000 is -8.000000`
- `3.000000 raised to -0.500000 is 0.577350`
- `4.000000 raised to 0.000000 is 1.000000`
- `Error: domain error`
- `0.000000 raised to -3.000000 is inf`
### powf

**Description:** Calculates $x$ raised to the power $y$.

**Include:**
```c
#include <math.h>
```

**Prototype:**
```c
float powf(float x, float y);
```

**Arguments:**
- $x$: base
- $y$: exponent

**Return Value:** Returns $x$ raised to the power $y$ ($x^y$).

**Remarks:**
- If $y$ is 0, `powf` returns 1. If $x$ is 0.0 and $y$ is less than 0, `powf` returns inf and a domain error occurs. If the result overflows or underflows, a range error occurs.

**Example:**
```c
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */
#include <math.h> /* for powf */

int main(void)
{
    float x, y, z;
    errno = 0;
    x = -2.0F;
    y = 3.0F;
    z = powf (x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n", x, y, z);

    errno = 0;
    x = 3.0F;
    y = -0.5F;
    z = powf (x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n", x, y, z);

    errno = 0;
    x = 0.0F;
    y = -3.0F;
    z = powf (x, y);
    if (errno)
        perror("Error");
    printf("%f raised to %f is %f\n", x, y, z);
}
```

**Output:**
- `-2.000000 raised to 3.000000 is -8.000000`
- `3.000000 raised to -0.500000 is 0.577350`
- `Error: domain error`
- `0.000000 raised to -3.000000 is inf`
sin

Description: Calculates the trigonometric sine function of a double precision floating-point value.

Include: <math.h>

Prototype: double sin (double x);

Argument: x value for which to return the sine

Return Value: Returns the sine of x in radians in the ranges of -1 to 1 inclusive.

Remarks: A domain error will occur if t x is a NaN or infinity.

Example:

```c
#include <math.h> /* for sin            */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    double x, y;

    errno = 0;
    x = -1.0;
    y = sin (x);
    if (errno)
        perror("Error");
    printf("The sine of %f is %f\n", x, y);

    errno = 0;
    x = 0.0;
    y = sin (x);
    if (errno)
        perror("Error");
    printf("The sine of %f is %f\n", x, y);
}
```

Output:
The sine of -1.000000 is -0.841471

The sine of 0.000000 is 0.000000
sinf

Description: Calculates the trigonometric sine function of a single precision floating-point value.

Include: <math.h>

Prototype: float sinf (float x);

Argument: x value for which to return the sine

Return Value: Returns the sin of x in radians in the ranges of -1 to 1 inclusive.

Remarks: A domain error will occur if x is a NaN or infinity.

Example:
```c
#include <math.h> /* for sinf           */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    float x, y;

    errno = 0;
    x = -1.0F;
    y = sinf (x);
    if (errno)
        perror("Error");
    printf("The sine of %f is %f\n", x, y);

    errno = 0;
    x = 0.0F;
    y = sinf (x);
    if (errno)
        perror("Error");
    printf("The sine of %f is %f\n", x, y);
}
```

Output:
The sine of -1.000000 is -0.841471
The sine of 0.000000 is 0.000000
**sinh**

**Description:** Calculates the hyperbolic sine function of a double precision floating-point value.

**Include:** `<math.h>`

**Prototype:**

```c
double sinh (double x);
```

**Argument:** `x` value for which to return the hyperbolic sine

**Return Value:** Returns the hyperbolic sine of `x`

**Remarks:** A range error will occur if the magnitude of `x` is too large.

**Example:**

```c
#include <math.h> /* for sinh           */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    double x, y;

    errno = 0;
    x = -1.5;
    y = sinh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic sine of %f is %f\n\n", x, y);

    errno = 0;
    x = 0.0;
    y = sinh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic sine of %f is %f\n\n", x, y);

    errno = 0;
    x = 720.0;
    y = sinh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic sine of %f is %f\n\n", x, y);
}
```

**Output:**

The hyperbolic sine of -1.500000 is -2.129279

The hyperbolic sine of 0.000000 is 0.000000

Error: range error
The hyperbolic sine of 720.000000 is inf
### sinhf

**Description:** Calculates the hyperbolic sine function of a single precision floating-point value.

**Include:**<math.h>

**Prototype:**

```c
float sinhf (float x);
```

**Argument:**

x  value for which to return the hyperbolic sine

**Return Value:**

Returns the hyperbolic sine of x

**Remarks:**

A range error will occur if the magnitude of x is too large.

**Example:**

```c
#include <math.h> /* for sinhf          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    float x, y;

    errno = 0;
    x = -1.0F;
    y = sinhf (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic sine of %f is %f\n\n", 
           x, y);

    errno = 0;
    x = 0.0F;
    y = sinhf (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic sine of %f is %f\n\n", 
           x, y);
}
```

**Output:**
The hyperbolic sine of -1.000000 is -1.175201

The hyperbolic sine of 0.000000 is 0.000000
sqrt

**Description:**
Calculates the square root of a double precision floating-point value.

**Include:**
`<math.h>`

**Prototype:**
`double sqrt(double x);`

**Argument:**
`x` a non-negative floating-point value

**Return Value:**
Returns the non-negative square root of `x`.

**Remarks:**
If `x` is negative, a domain error occurs.

**Example:**
```
#include <math.h> /* for sqrt           */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    double x, y;
    errno = 0;
    x = 0.0;
    y = sqrt(x);
    if (errno)
        perror("Error");
    printf("The square root of %f is %f\n\n", x, y);

    errno = 0;
    x = 9.5;
    y = sqrt(x);
    if (errno)
        perror("Error");
    printf("The square root of %f is %f\n\n", x, y);

    errno = 0;
    x = -25.0;
    y = sqrt(x);
    if (errno)
        perror("Error");
    printf("The square root of %f is %f\n\n", x, y);
}
```

**Output:**
The square root of 0.000000 is 0.000000

The square root of 9.500000 is 3.082207

Error: domain error
The square root of -25.000000 is nan
**sqrtf**

**Description:** Calculates the square root of a single precision floating-point value.

**Include:** `<math.h>`

**Prototype:**

```c
float sqrtf(float x);
```

**Argument:** x  non-negative floating-point value

**Return Value:** Returns the non-negative square root of x.

**Remarks:** If x is negative, a domain error occurs.

**Example:**

```c
#include <math.h> /* for sqrtf          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    double x;

    errno = 0;
    x = sqrtf (0.0F);
    if (errno)
        perror("Error");
    printf("The square root of 0.0F is \%f\n\n", x);

    errno = 0;
    x = sqrtf (9.5F);
    if (errno)
        perror("Error");
    printf("The square root of 9.5F is \%f\n\n", x);

    errno = 0;
    x = sqrtf (-25.0F);
    if (errno)
        perror("Error");
    printf("The square root of -25F is \%f\n", x);
}
```

**Output:**

The square root of 0.0F is 0.000000

The square root of 9.5F is 3.082207

Error: domain error
The square root of -25F is nan
tan

Description: Calculates the trigonometric tangent function of a double precision floating-point value.

Include: <math.h>

Prototype: double tan (double x);

Argument: x value for which to return the tangent

Return Value: Returns the tangent of x in radians.

Remarks: A domain error will occur if x is a NaN or infinity.

Example:
```c
#include <math.h>  /* for tan            */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    double x, y;

    errno = 0;
    x = -1.0;
    y = tan (x);
    if (errno)
        perror("Error");
    printf("The tangent of %f is %f\n\n", x, y);

    errno = 0;
    x = 0.0;
    y = tan (x);
    if (errno)
        perror("Error");
    printf("The tangent of %f is %f\n\n", x, y);
}
```

Output:
The tangent of -1.000000 is -1.557408

The tangent of 0.000000 is 0.000000

tanf

Description: Calculates the trigonometric tangent function of a single precision floating-point value.

Include: <math.h>

Prototype: float tanf (float x);

Argument: x value for which to return the tangent

Return Value: Returns the tangent of x

Remarks: A domain error will occur if x is a NaN or infinity.

Example:
```c
#include <math.h>  /* for tanf          */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno          */

int main(void)
{
    float x, y;
```
errno = 0;
x = -1.0F;
y = tanf (x);
if (errno)
    perror("Error");
printf("The tangent of %f is %f\n\n", x, y);

errno = 0;
x = 0.0F;
y = tanf (x);
if (errno)
    perror("Error");
printf("The tangent of %f is %f\n", x, y);
}

Output:
The tangent of -1.000000 is -1.557408
The tangent of 0.000000 is 0.000000

tanh

Description:
Calculates the hyperbolic tangent function of a double precision floating-point value.

Include:
<math.h>

Prototype:
double tanh (double x);

Argument:
x value for which to return the hyperbolic tangent

Return Value:
Returns the hyperbolic tangent of x in the ranges of -1 to 1 inclusive.

Remarks:
No domain or range error will occur.

Example:
#include <math.h> /* for tanh   */
#include <stdio.h> /* for printf */

int main(void)
{
    double x, y;
    
x = -1.0;
y = tanh (x);
    printf("The hyperbolic tangent of %f is %f\n\n", x, y);
    
x = 2.0;
y = tanh (x);
    printf("The hyperbolic tangent of %f is %f\n\n", x, y);
}

Output:
The hyperbolic tangent of -1.000000 is -0.761594
The hyperbolic tangent of 2.000000 is 0.964028
tanhf

**Description:** Calculates the hyperbolic tangent function of a single precision floating-point value.

**Include:**

```c
#include <math.h>
```

**Prototype:**

```c
float tanhf (float x);
```

**Argument:**

- `x` value for which to return the hyperbolic tangent

**Return Value:** Returns the hyperbolic tangent of `x` in the ranges of -1 to 1 inclusive.

**Remarks:** No domain or range error will occur.

**Example:**

```c
#include <math.h>  /* for tanhf  */
#include <stdio.h> /* for printf */

int main(void)
{
    float x, y;
    
    x = -1.0F;
    y = tanhf (x);
    printf("The hyperbolic tangent of %f is %f\n\n", x, y);

    x = 0.0F;
    y = tanhf (x);
    printf("The hyperbolic tangent of %f is %f\n\n", x, y);
}
```

**Output:**

The hyperbolic tangent of -1.000000 is -0.761594

The hyperbolic tangent of 0.000000 is 0.000000
4.18 PIC30-LIBS

The following functions are standard C library helper functions:

- _exit: terminate program execution
- brk: set the end of the process's data space
- close: close a file
- lseek: move a file pointer to a specified location
- open: open a file
- read: read data from a file
- sbrk: extend the process's data space by a given increment
- write: write data to a file

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 MPLAB C30) is available in the `src\pic30` folder.

Additionally, several standard C library functions must also be modified for the target application. They are:

- getenv: get a value for an environment variable
- remove: remove a file
- rename: rename a file or directory
- system: execute a command
- time: get the system time

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 MPLAB C30) is available in the `src\pic30` folder. These modules are not distributed as part of `libc-omf.a`.

4.18.1 Rebuilding the libpic30-omf.a library

By default, the helper functions listed in this chapter 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.
4.18.2 Function Descriptions

This section describes the functions that must be customized for correct operation of the Standard C Library in your 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.

_exit

Description: Terminate program execution.
Include: None
Prototype: void _exit (int status);
Argument: status exit status
Remarks: This is a helper function called by the exit() Standard C Library function.
Default Behavior: As distributed, this function flushes stdout and terminates. The parameter status is the same as that passed to the exit() standard C library function.
File: _exit.c

brk

Description: Set the end of the process's data space.
Include: None
Prototype: int brk(void *endds)
Argument: endds pointer to the end of the data segment
Return Value: Returns '0' if successful, '-1' if not.
Remarks: brk() is used to dynamically change the amount of space allocated for the calling process's data segment. The change is made by resetting the process's break value and allocating the appropriate amount of space. The break value is the address of the first location beyond the end of the data segment. The amount of allocated space increases as the break value increases.
Newly allocated space is uninitialized.
This helper function is used by the Standard C Library function malloc().
Default Behavior: If the argument `endds` is zero, the function sets the global variable `__curbrk` to the address of the start of the heap, and returns zero. If the argument `endds` is non-zero, and has a value less than the address of the end of the heap, the function sets the global variable `__curbrk` to the value of `endds` and returns zero. Otherwise, the global variable `__curbrk` is unchanged, and the function returns -1. The argument `endds` must be within the heap range (see data space memory map below).

Notice that, since the stack is located immediately above the heap, using `brk()` or `sbrk()` has little effect on the size of the dynamic memory pool. The `brk()` and `sbrk()` functions are primarily intended for use in run-time environments where the stack grows downward and the heap grows upward.

The linker allocates a block of memory for the heap if the `-Wl,--heap=n` option is specified, where `n` is the desired heap size in characters. The starting and ending addresses of the heap are reported in variables `_heap` and `_eheap`, respectively.

For MPLAB C30, using the linker's heap size option is the standard way of controlling heap size, rather than relying on `brk()` and `sbrk()`.

File: `brk.c`

---

**close**

**Description:** Close a file.

**Include:** None

**Prototype:**

```c
int close(int handle);
```

**Argument:** `handle` handle referring to an opened file

**Return Value:** Returns ‘0’ if the file is successfully closed. A return value of ‘-1’ indicates an error.

**Remarks:** This helper function is called by the `fclose()` Standard C Library function.

**Default Behavior:** As distributed, this function passes the file handle to the simulator, which issues a close in the host file system.

**File:** `close.c`
### getenv

**Description:** Get a value for an environment variable

**Include:** `<stdlib.h>`

**Prototype:**

```c
char *getenv(const char *s);
```

**Argument:**

- `s` name of environment variable

**Return Value:**

Returns a pointer to the value of the environment variable if successful; otherwise, returns a null pointer.

**Default Behavior:** As distributed, this function returns a null pointer. There is no support for environment variables.

**File:** `getenv.c`

### lseek

**Description:** Move a file pointer to a specified location.

**Include:** None

**Prototype:**

```c
long lseek(int handle, long offset, int origin);
```

**Argument:**

- `handle` refers to an opened file
- `offset` the number of characters from the origin
- `origin` the position from which to start the seek. `origin` may be one of the following values (as defined in `stdio.h`):
  - `SEEK_SET` – Beginning of file.
  - `SEEK_CUR` – Current position of file pointer.

**Return Value:**

Returns the offset, in characters, of the new position from the beginning of the file. A return value of `-1L` indicates an error.

**Remarks:** This helper function is called by the Standard C Library functions `fgetpos()`, `ftell()`, `fseek()`, `fsetpos`, and `rewind()`.

**Default Behavior:** As distributed, the parameters are passed to the host file system through the simulator. The return value is the value returned by the host file system.

**File:** `lseek.c`
### open

**Description:** Open a file.

**Include:** None

**Prototype:**
```c
int open(const char *name, int access, int mode);
```

**Argument:**
- `name` name of the file to be opened
- `access` access method to open file
- `mode` type of access permitted

**Return Value:** If successful, the function returns a file handle, a small positive integer. This handle is then used on subsequent low-level file I/O operations. A return value of `-1` indicates an error.

**Remarks:**
The access flag is a union of one of the following access methods and zero or more access qualifiers:
- `0` – Open a file for reading.
- `1` – Open a file for writing.
- `2` – Open a file for both reading and writing.

The following access qualifiers must be supported:
- `0x0008` – Move file pointer to end-of-file before every write operation.
- `0x0100` – Create and open a new file for writing.
- `0x0200` – Open the file and truncate it to zero length.
- `0x4000` – Open the file in text (translated) mode.
- `0x8000` – Open the file in binary (untranslated) mode.

The mode parameter may be one of the following:
- `0x0100` – Reading only permitted.
- `0x0080` – Writing permitted (implies reading permitted).

This helper function is called by the Standard C Library functions `fopen()` and `freopen()`.

**Default Behavior:** As distributed, the parameters are passed to the host file system through the simulator. The return value is the value returned by the host file system. If the host system returns a value of `-1`, the global variable `errno` is set to the value of the symbolic constant `EOPEN` defined in `<errno.h>`.

**File:** `open.c`
read

Description: Read data from a file.
Include: None
Prototype: int read(int handle, void * buffer,
    unsigned int len);
Argument: handle handle referring to an opened file
        buffer points to the storage location for read data
        len the maximum number of characters to read
Return Value: Returns the number of characters read, which may be less than len if
    there are fewer than len characters left in the file or if the file was
    opened in text mode, in which case each carriage return-linefeed
    (CR-LF) pair is replaced with a single linefeed character. Only the
    single linefeed character is counted in the return value. The
    replacement does not affect the file pointer. If the function tries to read
    at end-of-file, it returns '0'. If the handle is invalid, or the file is not open
    for reading, or the file is locked, the function returns '-1'.
Remarks: This helper function is called by the Standard C Library functions
        fgetc(), fgets(), fread(), and gets().
Default Behavior: As distributed, the parameters are passed to the host file system
    through the simulator. The return value is the value returned by the host
    file system.
File: read.c

remove

Description: Remove a file.
Include: <stdio.h>
Prototype: int remove(const char *filename);
Argument: filename file to be removed
Return Value: Returns '0' if successful, '-1' if unsuccessful.
Default Behavior: As distributed, the parameters are passed to the host file system
    through the simulator. The return value is the value returned by the host
    file system.
File: remove.c

rename

Description: Rename a file or directory.
Include: <stdio.h>
Prototype: int rename(const char *oldname, const char
    *newname);
Argument: oldname pointer to the old name
        newname pointer to the new name
Return Value: Returns '0' if it is successful. On an error, the function returns a
    non-zero value.
Default Behavior: As distributed, the parameters are passed to the host file system
    through the simulator. The return value is the value returned by the host
    file system.
File: rename.c
sbrk

Description: Extend the process's data space by a given increment.
Include: None
Prototype: void * sbrk(int incr);
Argument: incr number of characters to increment/decrement
Return Value: Return the start of the new space allocated, or '-1' for errors.
Remarks: sbrk() adds incr characters to the break value and changes the allocated space accordingly. incr can be negative, in which case the amount of allocated space is decreased.
sbrk() is used to dynamically change the amount of space allocated for the calling process's data segment. The change is made by resetting the process's break value and allocating the appropriate amount of space. The break value is the address of the first location beyond the end of the data segment. The amount of allocated space increases as the break value increases.
This is a helper function called by the Standard C Library function malloc().
Default Behavior: If the global variable __curbrk is zero, the function calls brk() to initialize the break value. If brk() returns -1, so does this function. If the incr is zero, the current value of the global variable __curbrk is returned.
If the incr is non-zero, the function checks that the address (__curbrk + incr) is less than the end address of the heap. If it is less, the global variable __curbrk is updated to that value, and the function returns the unsigned value of __curbrk.
Otherwise, the function returns -1.
See the description of brk().
File: sbrk.c

system

Description: Execute a command.
Include: <stdlib.h>
Prototype: int system(const char *s);
Argument: s command to be executed.
Default Behavior: As distributed, this function acts as a stub or placeholder for your function. If s is not NULL, an error message is written to stdout and the program will reset; otherwise, a value of -1 is returned.
File: system.c
### time

**Description:** Get the system time.

**Include:** `<time.h>`

**Prototype:**
```
time_t time(time_t *timer);
```

**Argument:** `timer` points to a storage location for time

**Return Value:** Returns the elapse time in seconds. There is no error return.

**Default Behavior:** As distributed, if timer2 is not enabled, it is enabled in 32-bit mode. The return value is the current value of the 32-bit timer2 register. Except in very rare cases, this return value is not the elapsed time in seconds.

**File:** `time.c`

### write

**Description:** Write data to a file.

**Include:** None

**Prototype:**
```
int write(int handle, void *buffer, unsigned int count);
```

**Argument:**
- `handle` refers to an opened file
- `buffer` points to the storage location of data to be written
- `count` the number of characters to write.

**Return Value:** If successful, write returns the number of characters actually written. A return value of `-1` indicates an error.

**Remarks:** If the actual space remaining on the disk is less than the size of the buffer the function is trying to write to the disk, write fails and does not flush any of the buffer's contents to the disk. If the file is opened in text mode, each linefeed character is replaced with a carriage return – linefeed pair in the output. The replacement does not affect the return value.

This is a helper function called by the Standard C Library function `fflush()`.

**Default Behavior:** As distributed, the parameters are passed to the host file system through the simulator. The return value is the value returned by the host file system.

**File:** `write.c`
Chapter 5. MPLAB C30 Built-in Functions

5.1 INTRODUCTION

This chapter describes the MPLAB C30 built-in functions that are specific to 16-bit devices.

Built-in functions give the C programmer access to assembler operators or machine instructions that are currently only accessible using inline assembly, but are sufficiently useful that they are applicable to a broad range of applications. Built-in functions are coded in C source files syntactically like function calls, but they are compiled to assembly code that directly implements the function, and do not involve function calls or library routines.

There are a number of reasons why providing built-in functions is preferable to requiring programmers to use inline assembly. They include the following:

1. Providing built-in functions for specific purposes simplifies coding.
2. Certain optimizations are disabled when inline assembly is used. This is not the case for built-in functions.
3. For machine instructions that use dedicated registers, coding inline assembly while avoiding register allocation errors can require considerable care. The built-in functions make this process simpler as you do not need to be concerned with the particular register requirements for each individual machine instruction.

This chapter is organized as follows:

• Built-In Function List
5.2 BUILT-IN FUNCTION LIST

This section describes the programmer interface to the MPLAB C30 C compiler built-in functions. Since the functions are "built in", there are no header files associated with them. Similarly, there are no command-line switches associated with the built-in functions – they are always available. The built-in function names are chosen such that they belong to the compiler’s namespace (they all have the prefix __builtin_), so they will not conflict with function or variable names in the programmer’s namespace.

__builtin_addab

**Description:** Add accumulators A and B with the result written back to the specified accumulator. For example:

```c
register int result asm("A");
result = __builtin_addab();
```

will generate:

```
add A
```

**Prototype:**

```
int __builtin_addab(void);
```

**Argument:** None

**Return Value:** Returns the addition result to an accumulator.

**Assembler Operator / Machine Instruction:**

```
addad
```

**Error Messages**

An error message will be displayed if the result is not an accumulator register.

__builtin_add

**Description:** Add value to the accumulator specified by result with a shift specified by literal shift. For example:

```c
register int result asm("A");
int value;
result = __builtin_add(value,0);
```

If value is held in w0, the following will be generated:

```
add w0, #0, A
```

**Prototype:**

```
int __builtin_add(int value, const int shift);
```

**Argument:**

- `value` Integer number to add to accumulator value.
- `shift` Amount to shift resultant accumulator value.

**Return Value:** Returns the shifted addition result to an accumulator.

**Assembler Operator / Machine Instruction:**

```
add
```

**Error Messages**

An error message will be displayed if:

- the result is not an accumulator register
- the shift value is not a literal within range
**__builtin_btg**

**Description:** This function will generate a btg machine instruction. Some examples include:

```c
int i; /* near by default */
int l __attribute__((far));

struct foo {
    int bit1:1;
} barbits;

int bar;

void some_bittoggles() {
    register int j asm("w9");
    int k;

    k = i;

    __builtin_btg(&bar,barbits.bit1);
    __builtin_btg(&i,1);
    __builtin_btg(&j,3);
    __builtin_btg(&k,4);
    __builtin_btg(&l,11);

    return j+k;
}
```

Note that taking the address of a variable in a register will produce warning by the compiler and cause the register to be saved onto the stack (so that its address may be taken); this form is not recommended. This caution only applies to variables explicitly placed in registers by the programmer.

**Prototype:**
```c
void __builtin_btg(unsigned int *, unsigned int n);
```

**Argument:**
- `*` A pointer to the data item for which a bit should be toggled.
- `n` A literal value in the range of 0 to 15. As a convenience, it is possible to pass a bit-field name as this argument. The builtin will substitute the bit position of the identified field for the argument and toggle the appropriate bit.

**Return Value:** Returns a btg machine instruction.

**Assembler Operator / Machine Instruction:** btg

**Error Messages** An error message will be displayed if the parameter values are not within range

---

**__builtin_clr**

**Description:** Clear the specified accumulator. For example:

```c
register int result asm("A");
result = __builtin_clr();
```

will generate:
```asm
clr A
```

**Prototype:**
```c
int __builtin_clr(void);
```

**Argument:** None
Return Value: Returns the cleared value result to an accumulator.

Assembler Operator / Machine Instruction: clr

Error Messages An error message will be displayed if the result is not an accumulator register.

__builtin_clr_prefetch

Description: Clear an accumulator and prefetch data ready for a future MAC operation.

xmmr may be null to signify no X prefetch to be performed, in which case the values of xincr and xval are ignored, but required.

yptr may be null to signify no Y prefetch to be performed, in which case the values of yincr and yval are ignored, but required.

xval and yval nominate the address of a C variable where the prefetched value will be stored.

xincr and yincr may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

If AWB is non null, the other accumulator will be written back into the referenced variable.

For example:

```c
register int result asm("A");
int x_memory_buffer[256]
__attribute__((space(xmemory)));
int y_memory_buffer[256]
__attribute__((space(ymemory)));
int *xmemory;
int *ymemory;
int awb;
int xVal, yVal;

xmemory = x_memory_buffer;
ymemory = y_memory_buffer;
result = __builtin_clr(&xmemory, &xVal, 2, &ymemory, &yVal, 2, &awb);
```

might generate:

```
clr A, [w8]+=2, w4, [w10]+=2, w5, w13
```

The compiler may need to spill w13 to ensure that it is available for the write-back. It may be recommended to users that the register be claimed for this purpose.

After this instruction:

- result will be cleared
- xVal will contain x_memory_buffer[0]
- yVal will contain y_memory_buffer[0]
- xmemory and ymemory will be incremented by 2, ready for the next MAC operation

Prototype:

```c
int __builtin_clr_prefetch(
    int **xptr, int *xval, int xincr,
    int **yptr, int *yval, int yincr, int *AWB);
```
__builtin_clr_prefetch (Continued)

Argument:
- xptr: Integer pointer to x prefetch.
- xval: Integer value of x prefetch.
- xincr: Integer increment value of x prefetch.
- yptr: Integer pointer to y prefetch.
- yval: Integer value of y prefetch.
- yincr: Integer increment value of y prefetch.
- AWB: Accumulator selection.

Return Value: Returns the cleared value result to an accumulator.

Assembler
- clr

Error Messages
- An error message will be displayed if:
  - the result is not an accumulator register
  - xval is a null value but xptr is not null
  - yval is a null value but yptr is not null

__builtin_divsd

Description: The function computes the quotient \( \frac{num}{den} \). A math error exception occurs if \( den \) is zero. Function arguments are signed, as is the function result. The command-line option `-Wconversions` can be used to detect unexpected sign conversions.

Prototype:
```c
int __builtin_divsd(const long num, const int den);
```

Argument:
- num: numerator
- den: denominator

Return Value: Returns the signed integer value of the quotient \( \frac{num}{den} \).

Assembler
- div.sd

__builtin_divud

Description: The function computes the quotient \( \frac{num}{den} \). A math error exception occurs if \( den \) is zero. Function arguments are unsigned, as is the function result. The command-line option `-Wconversions` can be used to detect unexpected sign conversions.

Prototype:
```c
unsigned int __builtin_divud(const unsigned long num, const unsigned int den);
```

Argument:
- num: numerator
- den: denominator

Return Value: Returns the unsigned integer value of the quotient \( \frac{num}{den} \).

Assembler
- div.ud
### __builtin_dmaoffset

**Description:** Obtain the offset of a symbol within DMA memory.

For example:

```c
int result;
char buffer[256] __attribute__((space(dma)));

result = __builtin_dmaoffset(buffer);
```

 Might generate:

```assembly
mov #dmaoffset(buffer), w0
```

**Prototype:**

```c
int __builtin_dmaoffset(int buffer);
```

**Argument:**

- `buffer` DMA address value

**Return Value:**

Returns the offset to an accumulator.

**Assembler Operator / Machine Instruction:**

```
dmaoffset
```

**Error Messages**

An error message will be displayed if the result is not an accumulator register.

### __builtin_ed

**Description:** Square `sqr`, returning it as the result. Also prefetch data for future square operation by computing `**xptr - **yptr` and storing the result in `*distance`.

`xincr` and `yincr` may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

For example:

```c
register int result asm("A");
int *xmemory, *ymemory;
int distance;

result = __builtin_ed(distance,
                        &xmemory, 2,
                        &ymemory, 2,
                        &distance);
```

 Might generate:

```assembly
ed w4*w4, A, [w8]+=2, [W10]+=2, w4
```

**Prototype:**

```c
int __builtin_ed(int sqr, int **xptr, int xincr,
                 int **yptr, int yincr, int *distance);
```

**Argument:**

- `sqr` Integer squared value.
- `xptr` Integer pointer to pointer to x prefetch.
- `xincr` Integer increment value of x prefetch.
- `yptr` Integer pointer to pointer to y prefetch.
- `yincr` Integer increment value of y prefetch.
- `distance` Integer pointer to distance.

**Return Value:**

Returns the squared result to an accumulator.

**Assembler Operator / Machine Instruction:**

```
ed
```
## __builtin_ed (Continued)

### Error Messages
An error message will be displayed if:
- the result is not an accumulator register
- xptr is null
- yptr is null
- distance is null

## __builtin_edac

### Description:
Square `sqr` and sum with the nominated accumulator register, returning it as the result. Also prefetch data for future square operation by computing `**xptr - **yptr` and storing the result in `*distance`. `xincr` and `yincr` may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

For example:
```c
register int result asm("A");
int *xmemory, *ymemory;
int distance;

result = __builtin_edac(distance,
                        &xmemory, 2,
                        &ymemory, 2,
                        &distance);
```

might generate:
```
edac w4*w4, A, [w8]+=2, [W10]+=2, w4
```

### Prototype:
```c
int __builtin_edac(int sqr, int **xptr, int xincr,
                    int **yptr, int yincr, int *distance);
```

### Argument:
- `sqr` : Integer squared value.
- `xptr` : Integer pointer to pointer to x prefetch.
- `xincr` : Integer increment value of x prefetch.
- `yptr` : Integer pointer to pointer to y prefetch.
- `yincr` : Integer increment value of y prefetch.
- `distance` : Integer pointer to distance.

### Return Value:
Returns the squared result to specified accumulator.

### Assembler Operator / Machine Instruction:
edac

### Error Messages
An error message will be displayed if:
- the result is not an accumulator register
- xptr is null
- yptr is null
- distance is null

## __builtin_fbcl

### Description:
Find the first bit change from left in value. This function is useful for dynamic scaling of fixed-point data. For example:
```c
int result, value;
result = __builtin_fbcl(value);
```

might generate:
```
fbc w4, w5
```

### Prototype:
```c
int __builtin_fbcl(int value);
```
### __builtin_fbcl (Continued) __

<table>
<thead>
<tr>
<th>Argument:</th>
<th>value</th>
<th>Integer number of first bit change.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return Value:</td>
<td>Returns the shifted addition result to an accumulator.</td>
<td></td>
</tr>
<tr>
<td>Assembler Operator / Machine Instruction:</td>
<td>fbcl</td>
<td></td>
</tr>
<tr>
<td>Error Messages</td>
<td>An error message will be displayed if the result is not an accumulator register.</td>
<td></td>
</tr>
</tbody>
</table>

---

### __builtin_lac __

**Description:** Shift value by `shift` (a literal between -8 and 7) and return the value to be stored into the accumulator register. For example:

```c
register int result asm("A");
int value;
result = __builtin_lac(value,3);
```

Might generate:

```
lac w4, #3, A
```

**Prototype:**

```c
int __builtin_lac(int value, int shift);
```

**Argument:**

- `value` Integer number to be shifted.
- `shift` Literal amount to shift.

**Return Value:** Returns the shifted addition result to an accumulator.

**Assembler Operator / Machine Instruction:**

- `lac`

**Error Messages:**

- An error message will be displayed if:
  - the result is not an accumulator register
  - the shift value is not a literal within range
__builtin_mac

**Description:** Compute $a \times b$ and sum with accumulator; also prefetch data ready for a future MAC operation.

- $xptr$ may be null to signify no X prefetch to be performed, in which case the values of $xincr$ and $xval$ are ignored, but required.
- $yptr$ may be null to signify no Y prefetch to be performed, in which case the values of $yincr$ and $yval$ are ignored, but required.
- $xval$ and $yval$ nominate the address of a C variable where the prefetched value will be stored.
- $xincr$ and $yincr$ may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

If $AWB$ is non null, the other accumulator will be written back into the referenced variable.

For example:

```c
register int result asm("A");
int *xmemory;
int *ymemory;
int xVal, yVal;

result = __builtin_mac(xVal, yVal, 
                        &xmemory, &xVal, 2, 
                        &ymemory, &yVal, 2, 0);
```

might generate:

mac w4*w5, A, [w8]+=2, w4, [w10]+=2, w5

**Prototype:**

```c
int __builtin_mac(int a, int b, 
                   int **xptr, int *xval, int xincr, 
                   int **yptr, int *yval, int yincr, int *AWB);
```

**Argument:**

- $a$ Integer multiplicand.
- $b$ Integer multiplier.
- $xptr$ Integer pointer to pointer to x prefetch.
- $xval$ Integer pointer to value of x prefetch.
- $xincr$ Integer increment value of x prefetch.
- $yptr$ Integer pointer to pointer to y prefetch.
- $yval$ Integer pointer to value of y prefetch.
- $yincr$ Integer increment value of y prefetch.
- $AWB$ Integer pointer to accumulator selection.

**Return Value:** Returns the cleared value result to an accumulator.

**Assembler Operator / Machine Instruction:**

mac

**Error Messages**

An error message will be displayed if:

- the result is not an accumulator register
- $xval$ is a null value but $xptr$ is not null
- $yval$ is a null value but $yptr$ is not null
__builtin_movsac

**Description:**
Compute nothing, but prefetch data ready for a future MAC operation.

- `xptr` may be null to signify no X prefetch to be performed, in which case the values of `xincr` and `xval` are ignored, but required.
- `yptr` may be null to signify no Y prefetch to be performed, in which case the values of `yincr` and `yval` are ignored, but required.
- `xval` and `yval` nominate the address of a C variable where the prefetched value will be stored.
- `xincr` and `yincr` may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

If `AWB` is non null, the other accumulator will be written back into the referenced variable.

**Prototype:**
```
int __builtin_movsac(
    int **xptr, int *xval, int xincr,  
    int **yptr, int *yval, int yincr, int *AWB);
```

**Argument:**
- `xptr` Integer pointer to pointer to x prefetch.
- `xval` Integer pointer to value of x prefetch.
- `xincr` Integer increment value of x prefetch.
- `yptr` Integer pointer to pointer to y prefetch.
- `yval` Integer pointer to value of y prefetch.
- `yincr` Integer increment value of y prefetch.
- `AWB` Integer pointer to accumulator selection.

**Return Value:**
Returns prefetch data.

**Assembler Operator / Machine Instruction:**
movsac

**Error Messages**
- An error message will be displayed if:
  - the result is not an accumulator register
  - `xval` is a null value but `xptr` is not null
  - `yval` is a null value but `yptr` is not null
__builtin_mpy

**Description:** compute \( a \times b \); also prefetch data ready for a future MAC operation.

\( xptr \) may be null to signify no \( X \) prefetch to be performed, in which case the values of \( xincr \) and \( xval \) are ignored, but required.

\( yptr \) may be null to signify no \( Y \) prefetch to be performed, in which case the values of \( yincr \) and \( yval \) are ignored, but required.

\( xval \) and \( yval \) nominate the address of a \( C \) variable where the prefetched value will be stored.

\( xincr \) and \( yincr \) may be the literal values: \(-6, -4, -2, 0, 2, 4, 6\) or an integer value.

For example:

```c
register int result asm("A");
int *xmemory;
int *ymemory;
int xVal, yVal;

result = __builtin_mpy(xVal, yVal, 
                      &xmemory, &xVal, 2, 
                      &ymemory, &yVal, 2);
```

might generate:

```assembly
mac w4*w5, A, [w8]+=2, w4, [w10]+=2, w5
```

**Prototype:**

```c
int __builtin_mpy(int a, int b, 
                   int **xptr, int *xval, int xincr, 
                   int **yptr, int *yval, int yincr);
```

**Argument:**

- \( a \)  Integer multiplicand.
- \( b \)  Integer multiplier.
- \( xptr \)  Integer pointer to pointer to \( X \) prefetch.
- \( xval \)  Integer pointer to value of \( X \) prefetch.
- \( xincr \)  Integer increment value of \( X \) prefetch.
- \( yptr \)  Integer pointer to pointer to \( Y \) prefetch.
- \( yval \)  Integer pointer to value of \( Y \) prefetch.
- \( yincr \)  Integer increment value of \( Y \) prefetch.
- \( AWB \)  Integer pointer to accumulator selection.

**Return Value:** Returns the cleared value result to an accumulator.

**Assembler Operator / Machine Instruction:**

- \( \text{mpy} \)

**Error Messages**

An error message will be displayed if:

- the result is not an accumulator register
- \( xval \) is a null value but \( xptr \) is not null
- \( yval \) is a null value but \( yptr \) is not null
__builtin_mpy

**Description:** compute \( axb \); also prefetch data ready for a future MAC operation. 

*\( xptr \)* may be null to signify no \( X \) prefetch to be performed, in which case the values of \( xincr \) and \( xval \) are ignored, but required.

*\( yptr \)* may be null to signify no \( Y \) prefetch to be performed, in which case the values of \( yincr \) and \( yval \) are ignored, but required.

\( xval \) and \( yval \) nominate the address of a \( C \) variable where the prefetched value will be stored.

\( xincr \) and \( yincr \) may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

For example:

```c
register int result asm("A");
int *xmemory;
int *ymemory;
int xVal, yVal;

result = __builtin_mpy(xVal, yVal,
 &xmemory, &xVal, 2,
 &ymemory, &yVal, 2);
```

might generate:

```asm
mac w4*w5, A, [w8]+=2, w4, [w10]+=2, w5
```

**Prototype:**

```c
int __builtin_mpy(int a, int b,
 int **xptr, int *xval, int xincr,
 int **yptr, int *yval, int yincr);
```

**Argument:**

- \( a \) Integer multiplicand.
- \( b \) Integer multiplier.
- \( xptr \) Integer pointer to pointer to \( x \) prefetch.
- \( xval \) Integer pointer to value of \( x \) prefetch.
- \( xincr \) Integer increment value of \( x \) prefetch.
- \( yptr \) Integer pointer to pointer to \( y \) prefetch.
- \( yval \) Integer pointer to value of \( y \) prefetch.
- \( yincr \) Integer increment value of \( y \) prefetch.
- \( AWB \) Integer pointer to accumulator selection.

**Return Value:**

Returns the cleared value result to an accumulator.

**Assembler Operator / Machine Instruction:**

`mpyn`

**Error Messages**

An error message will be displayed if:

- the result is not an accumulator register
- \( xval \) is a null value but \( xptr \) is not null
- \( yval \) is a null value but \( yptr \) is not null
__builtin_msc

Description: compute \( a \times b \) and subtract from accumulator; also prefetch data ready for a future MAC operation.

\( xptr \) may be null to signify no \( X \) prefetch to be performed, in which case the values of \( xincr \) and \( xval \) are ignored, but required.

\( yptr \) may be null to signify no \( Y \) prefetch to be performed, in which case the values of \( yincr \) and \( yval \) are ignored, but required.

\( xval \) and \( yval \) nominate the address of a C variable where the prefetched value will be stored.

\( xincr \) and \( yincr \) may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

If \( AWB \) is non null, the other accumulator will be written back into the referenced variable.

For example:

```c
register int result asm("A");
int *xmemory;
int *ymemory;
int xVal, yVal;

result = __builtin_msc(xVal, yVal, &xmemory, &xVal, 2, &ymemory, &yVal, 2, 0);
```

Prototype:
```
int __builtin_msc(int a, int b, int **xptr, int *xval, int xincr, int **yptr, int *yval, int yincr, int *AWB);
```

Argument:
- \( a \) Integer multiplicand.
- \( b \) Integer multiplier.
- \( xptr \) Integer pointer to pointer to \( X \) prefetch.
- \( xval \) Integer pointer to value of \( X \) prefetch.
- \( xincr \) Integer increment value of \( X \) prefetch.
- \( yptr \) Integer pointer to pointer to \( Y \) prefetch.
- \( yval \) Integer pointer to value of \( Y \) prefetch.
- \( yincr \) Integer increment value of \( Y \) prefetch.
- \( AWB \) Integer pointer to accumulator selection.

Return Value: Returns the cleared value result to an accumulator.

Assembler Operator / Machine Instruction:
```
msc
```

Error Messages
An error message will be displayed if:
- the result is not an accumulator register
- \( xval \) is a null value but \( xptr \) is not null
- \( yval \) is a null value but \( yptr \) is not null

__builtin_mulss

Description: The function computes the product \( p0 \times p1 \). Function arguments are signed integers, and the function result is a signed long integer. The command-line option `-Wconversions` can be used to detect unexpected sign conversions.

Prototype:
```
signed long __builtin_mulss(const signed int p0, const signed int p1);
```
Argument: \( p_0 \) multiplicand
\( p_1 \) multiplier

Return Value: Returns the signed long integer value of the product \( p_0 \times p_1 \).

Assembler
\texttt{mul.ss}

Operator / Machine Instruction:
\texttt{mul.su}

Description: The function computes the product \( p_0 \times p_1 \). Function arguments are integers with mixed signs, and the function result is a signed long integer. The command-line option \texttt{-Wconversions} can be used to detect unexpected sign conversions. This function supports the full range of addressing modes of the instruction, including immediate mode for operand \( p_1 \).

Prototype:
\begin{verbatim}
signed long __builtin_mulsu(const signed int p0,
const unsigned int p1);
\end{verbatim}

Argument:
\( p_0 \) multiplicand
\( p_1 \) multiplier

Return Value: Returns the signed long integer value of the product \( p_0 \times p_1 \).

Assembler
\texttt{mul.us}

Operator / Machine Instruction:
\texttt{mul.usu}

Description: The function computes the product \( p_0 \times p_1 \). Function arguments are unsigned integers, and the function result is an unsigned long integer. The command-line option \texttt{-Wconversions} can be used to detect unexpected sign conversions. This function supports the full range of addressing modes of the instruction, including immediate mode for operand \( p_1 \).

Prototype:
\begin{verbatim}
unsigned long __builtin_muluu(const unsigned int p0,
const unsigned int p1);
\end{verbatim}

Argument:
\( p_0 \) multiplicand
\( p_1 \) multiplier

Return Value: Returns the signed long integer value of the product \( p_0 \times p_1 \).

Assembler
\texttt{mul.usu}
## __builtin_muluu (Continued)

<table>
<thead>
<tr>
<th>Return Value:</th>
<th>Returns the signed long integer value of the product $p_0 \times p_1$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembler Operator / Machine Instruction:</td>
<td>mul.uu</td>
</tr>
</tbody>
</table>

## __builtin_nop

<table>
<thead>
<tr>
<th>Description:</th>
<th>This function will generate a nop instruction.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prototype:</td>
<td>void __builtin_nop(void);</td>
</tr>
<tr>
<td>Argument:</td>
<td>None.</td>
</tr>
<tr>
<td>Return Value:</td>
<td>Returns a no operation (nop).</td>
</tr>
<tr>
<td>Assembler Operator / Machine Instruction:</td>
<td>nop</td>
</tr>
</tbody>
</table>

## __builtin_psvpage

<table>
<thead>
<tr>
<th>Description:</th>
<th>The function returns the psv page number of the object whose address is given as a parameter. The argument $p$ must be the address of an object in an EE data, PSV or executable memory space; otherwise an error message is produced and the compilation fails. See the space attribute in the &quot;MPLAB® C30 C Compiler User's Guide&quot; (DS51284).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prototype:</td>
<td>unsigned int __builtin_psvpage(const void *p);</td>
</tr>
<tr>
<td>Argument:</td>
<td>$p$ object address</td>
</tr>
<tr>
<td>Return Value:</td>
<td>Returns the psv page number of the object whose address is given as a parameter.</td>
</tr>
<tr>
<td>Assembler Operator / Machine Instruction:</td>
<td>psvpage</td>
</tr>
</tbody>
</table>

### Error Messages

The following error message is produced when this function is used incorrectly:

"Argument to __builtin_psvpage() is not the address of an object in code, psv, or eedata section".

The argument must be an explicit object address.

For example, if obj is object in an executable or read-only section, the following syntax is valid:

```c
unsigned page = __builtin_psvpage(&obj);
```

## __builtin_psvoffset

<table>
<thead>
<tr>
<th>Description:</th>
<th>The function returns the psv page offset of the object whose address is given as a parameter. The argument $p$ must be the address of an object in an EE data, PSV or executable memory space; otherwise an error message is produced and the compilation fails. See the space attribute in the &quot;MPLAB® C30 C Compiler User's Guide&quot; (DS51284).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prototype:</td>
<td>unsigned int __builtin_psvoffset(const void *p);</td>
</tr>
<tr>
<td>Argument:</td>
<td>$p$ object address</td>
</tr>
<tr>
<td>Return Value:</td>
<td>Returns the psv page number offset of the object whose address is given as a parameter.</td>
</tr>
</tbody>
</table>
__builtin_psvoffset (Continued)

<table>
<thead>
<tr>
<th>Assembler</th>
<th>psvoffset</th>
</tr>
</thead>
</table>

**Operator / Machine Instruction:**

**Error Messages:**
The following error message is produced when this function is used incorrectly:

"Argument to __builtin_psvoffset() is not the address of an object in code, psv, or eedata section".
The argument must be an explicit object address.
For example, if `obj` is object in an executable or read-only section, the following syntax is valid:

```c
unsigned page = __builtin_psvoffset(&obj);
```

__builtin_return_address

**Description:**
This function returns the return address of the current function, or of one of its callers. For the `level` argument, a value of 0 yields the return address of the current function, a value of 1 yields the return address of the caller of the current function, and so forth. When `level` exceeds the current stack depth, 0 will be returned. This function should only be used with a non-zero argument for debugging purposes.

**Prototype:**
```c
int __builtin_return_address (const int level);
```

**Argument:**
- `level`: Number of frames to scan up the call stack.

**Return Value:**
Returns the return address of the current function, or of one of its callers.

**Assembler**
- `return_address`

__builtin_sac

**Description:**
Shift value by `shift` (a literal between -8 and 7) and return the value.

For example:
```c
register int value asm("A");
int result;

result = __builtin_sac(value, 3);
```

Might generate:
```asm
sac A, #3, w0
```

**Prototype:**
```c
int __builtin_sac(int value, int shift);
```

**Argument:**
- `value`: Integer number to be shifted.
- `shift`: Literal amount to shift.

**Return Value:**
Returns the shifted result to an accumulator.

**Assembler**
- `sac`

**Error Messages:**
An error message will be displayed if:
- the result is not an accumulator register
- the shift value is not a literal within range
### __builtin_sacr

**Description:** Shift value by *shift* (a literal between -8 and 7) and return the value which is rounded using the rounding mode determined by the CORCONbits.RND control bit.

For example:

```c
register int value asm("A");
int result;

result = __builtin_sac(value,3);
```

Might generate:

```
sac.r A, #3, w0
```

**Prototype:**

```c
int __builtin_sac(int value, int shift);
```

**Argument:**

- `value`: Integer number to be shifted.
- `shift`: Literal amount to shift.

**Return Value:** Returns the shifted result to CORCON register.

**Assembler Operator / Machine Instruction:**

`sacr`

**Error Messages**

An error message will be displayed if:
- the result is not an accumulator register
- the shift value is not a literal within range

### __builtin_sftac

**Description:** Shift accumulator by *shift*. The valid shift range is -16 to 16.

For example:

```c
register int result asm("A");
int i;

result = __builtin_sftac(i);
```

Might generate:

```
sftac A, w0
```

**Prototype:**

```c
int __builtin_sftac(int shift);
```

**Argument:**

- `shift`: Literal amount to shift.

**Return Value:** Returns the shifted result to an accumulator.

**Assembler Operator / Machine Instruction:**

`sftac`

**Error Messages**

An error message will be displayed if:
- the result is not an accumulator register
- the shift value is not a literal within range

### __builtin_subab

**Description:** Subtract accumulators A and B with the result written back to the specified accumulator. For example:

```c
register int result asm("A");
result = __builtin_subab();
```

will generate:

```
sub A
```

**Prototype:**

```c
int __builtin_subab(void);
```

**Argument:** None
__builtin_subab (Continued)

Return Value: Returns the subtraction result to an accumulator.
Assembler Operator / Machine Instruction: subad
Error Messages An error message will be displayed if the result is not an accumulator register.

__builtin_tblpage

Description: The function returns the table page number of the object whose address is given as a parameter. The argument \( p \) must be the address of an object in an EE data, PSV or executable memory space; otherwise an error message is produced and the compilation fails. See the space attribute in the "MPLAB® C30 C Compiler User’s Guide" (DS51284).
Prototype: unsigned int __builtin_tblpage(const void *\( p \));
Argument: \( p \) object address
Return Value: Returns the table page number of the object whose address is given as a parameter.
Assembler Operator / Machine Instruction: tblpage
Error Messages The following error message is produced when this function is used incorrectly:
"Argument to __builtin_tblpage() is not the address of an object in code, psv, or eedata section".
The argument must be an explicit object address.
For example, if \( \text{obj} \) is object in an executable or read-only section, the following syntax is valid:
unsigned page = __builtin_tblpage(&\( \text{obj} \));

__builtin_tbloffset

Description: The function returns the table page offset of the object whose address is given as a parameter. The argument \( p \) must be the address of an object in an EE data, PSV or executable memory space; otherwise an error message is produced and the compilation fails. See the space attribute in the "MPLAB® C30 C Compiler User’s Guide".
Prototype: unsigned int __builtin_tbloffset(const void *\( p \));
Argument: \( p \) object address
Return Value: Returns the table page number offset of the object whose address is given as a parameter.
Assembler Operator / Machine Instruction: tbloffset
Error Messages The following error message is produced when this function is used incorrectly:
"Argument to __builtin_tbloffset() is not the address of an object in code, psv, or eedata section".
The argument must be an explicit object address.
For example, if \( \text{obj} \) is object in an executable or read-only section, the following syntax is valid:
unsigned page = __builtin_tbloffset(&\( \text{obj} \));
### TABLE A-1: ASCII CHARACTER SET

<table>
<thead>
<tr>
<th>Hex</th>
<th>Most Significant Character</th>
<th>Least Significant Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>DLE</td>
</tr>
<tr>
<td></td>
<td>Space</td>
<td>@</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P</td>
</tr>
<tr>
<td></td>
<td></td>
<td>' p</td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>DC1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>!</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>q</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>DC2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r</td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>DC3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s</td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
<td>DC4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>NAK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>U</td>
</tr>
<tr>
<td></td>
<td></td>
<td>e</td>
</tr>
<tr>
<td></td>
<td></td>
<td>u</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>SYN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&amp;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f</td>
</tr>
<tr>
<td></td>
<td></td>
<td>v</td>
</tr>
<tr>
<td>7</td>
<td>Bell</td>
<td>ETB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>G</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W</td>
</tr>
<tr>
<td></td>
<td></td>
<td>g</td>
</tr>
<tr>
<td></td>
<td></td>
<td>w</td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>CAN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>H</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
<td>EM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y</td>
</tr>
<tr>
<td>A</td>
<td>LF</td>
<td>SUB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>J</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Z</td>
</tr>
<tr>
<td></td>
<td></td>
<td>j</td>
</tr>
<tr>
<td></td>
<td></td>
<td>z</td>
</tr>
<tr>
<td>B</td>
<td>VT</td>
<td>ESC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+</td>
</tr>
<tr>
<td></td>
<td></td>
<td>;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[</td>
</tr>
<tr>
<td></td>
<td></td>
<td>k</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{</td>
</tr>
<tr>
<td>C</td>
<td>FF</td>
<td>FS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\</td>
</tr>
<tr>
<td></td>
<td></td>
<td>l</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>CR</td>
<td>GS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>m</td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td>E</td>
<td>SO</td>
<td>RS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N</td>
</tr>
<tr>
<td></td>
<td></td>
<td>^</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n</td>
</tr>
<tr>
<td></td>
<td></td>
<td>~</td>
</tr>
<tr>
<td>F</td>
<td>SI</td>
<td>US</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/</td>
</tr>
<tr>
<td></td>
<td></td>
<td>?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>O</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DEL</td>
</tr>
</tbody>
</table>
Index

Symbols
#define ............................................. 10, 82, 84, 107, 114, 155, 171, 193
#if .......................................................... 219
#include.................................................. 10, 82, 202
% .............................................................. 231, 262, 267, 268, 331
\ ...................................................... 268
\l, Form Feed ............................................. 209
\n, Newline ................................................... 209, 233, 243, 248, 259, 260, 264
\r, Carriage Return ..................................... 209
\t, Horizontal Tab ........................................... 209
\v, Vertical Tab ........................................... 209
^, Caret ..................................................... 268
__builtin_add ........................................... 384
__builtin_addab ......................................... 384
__builtin_adds ........................................... 385
__builtin_addl ........................................... 385
__builtin_addp ........................................... 386
__builtin_div ........................................... 387
__builtin_divd ........................................... 387
__builtin_divud .......................................... 387
__builtin_dmaofsset ................................. 388
__builtin_ed ............................................. 388
__builtin_edac .......................................... 389
__builtin_fbc ........................................... 389
__builtin_fcl ........................................... 390
__builtin_lac ........................................... 390
__builtin_mac ........................................... 390
__builtin_mvsac ....................................... 391
__builtin_myp .......................................... 392
__builtin_mpy .......................................... 392
__builtin_myn .......................................... 393
__builtin_msc .......................................... 394
__builtin_mulss ........................................ 394
__builtin_mulsu ........................................ 395
__builtin_mulus ........................................ 395
__builtin_mulus ........................................ 395
__builtin_mului ........................................ 395
__builtin_nop .......................................... 396
__builtin_psoffset .................................... 396
__builtin_psvpage ..................................... 396
__builtin_return_address ......................... 397
__builtin_sac .......................................... 397
__builtin_sacr ......................................... 398
__builtin_sfac ......................................... 398
__builtin_subab ........................................ 398
__builtin_tbsloffset ................................ 399
__builtin_tbipage .................................... 399
FILE ...................................................... 203
_LINE ...................................................... 203
_exit .................................................... 375
_IOFBF ............................................... 234, 269, 270
_LOFBF ................................................. 234, 270
_HOFBF ................................................ 234, 269, 270
MathError ............................................... 221
_NSETJMP .............................................. 222

Numerics
0x ............................................................... 210, 261, 299, 300
16-Bit Peripheral Libraries .......................... 81
A
Abnormal Termination Signal ...................... 224
abort ....................................................... 203, 279
abs .......................................................... 280
Absolute Value
Double Floating Point ................................ 347
Integer .................................................... 280
Long Integer ........................................... 290
Single Floating Point ................................ 347
Absolute Value Function
abs .......................................................... 280
fabs ....................................................... 347
fabsf ...................................................... 347
labs ....................................................... 290
Access Mode
Binary .................................................... 244
Text ....................................................... 244
AckI2C .................................................... 192
acos ....................................................... 333
acosf ..................................................... 334
ADC, 10-Bit
Busy ....................................................... 110
Close ..................................................... 110
Configure Interrupt ................................ 110
Disable Interrupt Macro ......................... 115
Enable Interrupt Macro ........................... 115
Example of Use ....................................... 116
Open ....................................................... 111
Read ...................................................... 114
Set Channel ........................................... 114
Set Interrupt Priority Macro ................... 115
Start Conversion ..................................... 111
Stop Sampling ......................................... 114
ADC, 12-Bit
Busy ....................................................... 103
Close ..................................................... 103
Configure Interrupt ................................ 103
Disable Interrupt Macro ......................... 108
Enable Interrupt Macro ......................... 108
Example of Use ....................................... 109
Open ....................................................... 104
Read ...................................................... 107
Set Channel ........................................... 107
Set Interrupt Priority Macro ................... 108
Start Conversion ..................................... 104
Stop Sampling ......................................... 107
<table>
<thead>
<tr>
<th>Alphanumeric Character</th>
<th>Defined</th>
<th>204</th>
<th>Test for</th>
<th>204</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphabetic Character</td>
<td>Defined</td>
<td>204</td>
<td>Test for</td>
<td>204</td>
</tr>
<tr>
<td>AM/PM</td>
<td></td>
<td>331</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Append</td>
<td></td>
<td>308, 314</td>
<td></td>
<td></td>
</tr>
<tr>
<td>arcsin</td>
<td></td>
<td>334</td>
<td></td>
<td></td>
</tr>
<tr>
<td>arcsin</td>
<td>Double Floating Point</td>
<td>335</td>
<td></td>
<td></td>
</tr>
<tr>
<td>arcsin</td>
<td>Single Floating Point</td>
<td>335</td>
<td></td>
<td></td>
</tr>
<tr>
<td>arctangent</td>
<td></td>
<td>336</td>
<td></td>
<td></td>
</tr>
<tr>
<td>arctangent</td>
<td>Double Floating Point</td>
<td>337</td>
<td></td>
<td></td>
</tr>
<tr>
<td>arctangent</td>
<td>Single Floating Point</td>
<td>337</td>
<td></td>
<td></td>
</tr>
<tr>
<td>arctangent of y/x</td>
<td></td>
<td>337</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Argument List</td>
<td></td>
<td>229, 275, 276, 277</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASCII Character Set</td>
<td></td>
<td>401</td>
<td></td>
<td></td>
</tr>
<tr>
<td>asctime</td>
<td></td>
<td>326</td>
<td></td>
<td></td>
</tr>
<tr>
<td>asin</td>
<td></td>
<td>335</td>
<td></td>
<td></td>
</tr>
<tr>
<td>asinf</td>
<td></td>
<td>335</td>
<td></td>
<td></td>
</tr>
<tr>
<td>assert</td>
<td></td>
<td>203</td>
<td></td>
<td></td>
</tr>
<tr>
<td>assert.h</td>
<td></td>
<td>203</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assignment Suppression</td>
<td></td>
<td>267</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asterisk</td>
<td></td>
<td>261, 267</td>
<td></td>
<td></td>
</tr>
<tr>
<td>atan</td>
<td></td>
<td>336</td>
<td></td>
<td></td>
</tr>
<tr>
<td>atan2</td>
<td></td>
<td>337</td>
<td></td>
<td></td>
</tr>
<tr>
<td>atan2f</td>
<td></td>
<td>339</td>
<td></td>
<td></td>
</tr>
<tr>
<td>atanf</td>
<td></td>
<td>337</td>
<td></td>
<td></td>
</tr>
<tr>
<td>atexit</td>
<td></td>
<td>280, 288</td>
<td></td>
<td></td>
</tr>
<tr>
<td>atof</td>
<td></td>
<td>282</td>
<td></td>
<td></td>
</tr>
<tr>
<td>atoi</td>
<td></td>
<td>283</td>
<td></td>
<td></td>
</tr>
<tr>
<td>atol</td>
<td></td>
<td>283</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>283</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BartlettInit</td>
<td></td>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base</td>
<td></td>
<td>299, 300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>215, 216, 217, 218, 358, 359</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>217</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td></td>
<td>357, 360</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLT_RADIX</td>
<td></td>
<td>214, 215, 216, 217, 218, 219</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary</td>
<td></td>
<td>217</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base</td>
<td></td>
<td>244, 272</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mode</td>
<td></td>
<td>244, 272</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Search</td>
<td></td>
<td>284</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Streams</td>
<td></td>
<td>233</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bitfields</td>
<td></td>
<td>232</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BitReverseComplex</td>
<td></td>
<td>59</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BlackmanInit</td>
<td></td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BORStatReset</td>
<td></td>
<td>127</td>
<td></td>
<td></td>
</tr>
<tr>
<td>brk</td>
<td></td>
<td>375, 380</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bsearch</td>
<td></td>
<td>284</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Buffer Size</td>
<td></td>
<td>234, 270</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BufferEmptyDCI</td>
<td></td>
<td>158</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Buffering Modes</td>
<td></td>
<td>270</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Buffering, See File Buffering</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BUFSIZ</td>
<td></td>
<td>234, 269</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Built-In Functions</td>
<td></td>
<td>204, 221</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>204, 221</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calendar Time</td>
<td></td>
<td>325, 327, 328, 330, 332</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calloc</td>
<td></td>
<td>286, 289</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN Interrupts</td>
<td></td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CANxAbortAll</td>
<td></td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CANxGetRXErrorCount</td>
<td></td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CANxGetTXErrorCount</td>
<td></td>
<td>90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CANxInitialize</td>
<td></td>
<td>98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CANxIsRXPassive</td>
<td></td>
<td>91</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CANxIsRXReady</td>
<td></td>
<td>90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CANxIsTXPassive</td>
<td></td>
<td>91</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CANxIsTXReady</td>
<td></td>
<td>92</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| CAN Interrupts         |         | 100 |
| CAN, Example of Use    |         | 101 |
| CANxAbortAll           |         | 89 |
| CANxGetRXErrorCount    |         | 89 |
| CANxGetTXErrorCount    |         | 90 |
| CANxInitialize         |         | 98 |
| CANxIsRXPassive        |         | 91 |
| CANxIsRXReady          |         | 90 |
| CANxIsTXPassive        |         | 91 |
| CANxIsTXReady          |         | 92 |
| C Locale               |         | 204, 221 |

© 2005 Microchip Technology Inc.
16-Bit Language Tools Libraries

Concatenation Functions
- strcat .......................................................... 308
- strncpy .......................................................... 314
- ConfigCNPullups .................................................. 130
- ConfigIntADC10 .................................................. 110
- ConfigIntADC12 .................................................. 103
- ConfigIntCANx .................................................. 99
- ConfigIntCapturex .................................................. 134
- ConfigIntCN .................................................. 130
- ConfigIntDCI .................................................. 159
- ConfigIntI2Cx .................................................. 191
- ConfigIntIOCx .................................................. 140
- ConfigIntMCPWM .................................................. 179
- ConfigIntQEI .................................................. 174
- ConfigIntSpIx .................................................. 166
- ConfigIntTimerxx .................................................. 118
- ConfigIntTimerxx .................................................. 119
- ConfigIntUARTx .................................................. 150
- ConfigINTx .................................................. 129

Control Character
- Defined .......................................................... 205
- Test for .......................................................... 205
- Control Transfers .................................................. 222
- Conversion .................................................. 261, 267, 271

Convert
- Character to Multibyte Character .................................. 301
- Multibyte Character to Wide Character .................................. 293
- Multibyte String to Wide Character String .................................. 293
- String to Double Floating Point .................................. 282, 297
- String to Integer .................................................. 283
- String to Long Integer .................................................. 283, 299
- String to Unsigned Long Integer .................................. 300
- To Lower Case Alphabetic Character .................................. 211
- To Upper Case Alphabetic Character .................................. 212
- Wide Character String to Multibyte String .................................. 301

ConvertADC10 .................................................. 111

ConvertADC12 .................................................. 104

Copying Functions
- memcpy .......................................................... 305
- memmove .......................................................... 306
- memset .......................................................... 307
- strcpy .......................................................... 311
- strncpy .......................................................... 317
- cos .......................................................... 341
- cosf .......................................................... 342
- CosFactorInit .................................................. 61
- cosh .......................................................... 343
- coshf .......................................................... 344
- cosine .......................................................... 341
- Double Floating Point .................................................. 341
- Single Floating Point .................................................. 342
- cr0, cr1 .......................................................... 8
- ctime .......................................................... 327
- ctype.h .......................................................... 204
- isalnum .......................................................... 204
- iscntrl .......................................................... 205
- isdigit .......................................................... 206
- isgraph .......................................................... 206
- isalpha .......................................................... 204
- islower .......................................................... 207
- ispring .......................................................... 208
- ispunct .......................................................... 208
- isspace .......................................................... 209
- isupper .......................................................... 210
- isxdigit .......................................................... 210
- tolower .......................................................... 211
- toupper .......................................................... 212

Current Argument .................................................. 229

Customer Notification Service ........................................ 5
Customer Support .................................................. 6
Customized Function .................................................. 289

D
- Dash (-) .......................................................... 268
- DataRdyDCI .................................................. 159
- DataRdyI2Cx .................................................. 192
- DataRdySPIx .................................................. 167
- DataRdyUARTx .................................................. 151
- Date and Time .................................................. 330
- Date and Time Functions, See time.h .................................. 325, 326, 330
- Day of the Month .................................................. 325, 326, 330
- Day of the Year .................................................. 325, 331
- Daylight Savings Time .................................................. 325, 328, 329
- DBL_DIG .......................................................... 214
- DBL_EPSILON .................................................. 214
- DBL_MANT_DIG .................................................. 214
- DBL_MAX .................................................. 214
- DBL_MAX_10_EXP .................................................. 215
- DBL_MAX_EXP .................................................. 215
- DBL_MIN .................................................. 215
- DBL_MIN_10_EXP .................................................. 215
- DBL_MIN_EXP .................................................. 216

DCI Functions
- Close DCI .................................................. 158
- Configure DCI .................................................. 159
- Configure DCI Interrupt .................................................. 159
- DCI RX Buffer Status .................................................. 159
- DCI TX Buffer Status .................................................. 158
- Example of Use .................................................. 164
- Read DCI RX Buffer .................................................. 162
- Write DCI TX Buffer .................................................. 162

DCI Macros
- Disable DCI Interrupt .................................................. 163
- Enable DCI Interrupt .................................................. 163
- Set DCI Interrupt Priority .................................................. 163

DCT .................................................. 61
- DCTIP .................................................. 63
- Deallocate Memory .................................................. 289, 295
- Debugging Logic Errors .................................................. 203
- Decimal .................................................. 262, 268, 299, 300
Decimal Digit
   Defined ........................................ 206
   Number Of ................................. 214, 216, 217
   Test for ................................ 206
Decimal Point ........................................ 261
Default Handler ..................................... 223
Diagnostics, See assert.h
difftime ........................................ 328
Digit, Decimal, See Decimal Digit
Digit, Hexadecimal, See Hexadecimal Digit
Direct Input/Output Functions
  fread ........................................ 248
  fwrite .................................... 256
DisableCNx ..................................... 131
DisableIntADC ................................ 108, 115
DisableIntCANx .................................. 100
DisableIntDCI .................................. 163
DisableInterrupts ................................ 127
DisableIntFLTA ................................ 188
DisableIntFLTB ................................ 189
DisableIntICI .................................. 137
DisableIntIUXRX ............................. 155
DisableIntMCPWM ................................ 188
DisableIntM2C .................................. 198
DisableIntQEI .................................. 177
DisableIntOCx .................................. 147
DisableIntSPIx .................................. 199
DisableIntSIC .................................. 199
DisableIntTlx .................................. 171
DisableIntTx .................................. 123
DisableIntUtx .................................. 156
DisableIntTx .................................. 132
div ........................................ 278, 286
div_type ........................................ 278
Divide
   Integer ...................................... 286
   Long Integer .............................. 291
Divide by Zero .................................. 224, 227, 286
Document
   Layout ....................................... 2
Documentation
   Conventions .................................. 3
Domain Error... 213, 333, 334, 335, 337, 339, 341, 342,
   349, 351, 357, 358, 359, 360, 365, 366, 369, 370, 371
dot ........................................ 261
Double Precision Floating Point
   Machine Epsilon ............................ 214
   Maximum Exponent (base 10) .......... 215
   Maximum Exponent (base 2) .......... 215
   Maximum Value ............................ 214
   Minimum Exponent (base 10) .......... 215
   Minimum Exponent (base 2) .......... 216
   Minimum Value ............................ 215
   Number of Binary Digits .............. 214
   Number of Decimal Digits ............ 214
double Type .................................. 233
Dream Function .................................. 261
DSP Libraries .................................. 9
E
   EDOM ......................................... 213
   edom ....................................... 333
   Ellipses (…) ................................ 229, 268
   Empty Binary File ......................... 244
   Empty Text File ........................... 244
   EnableCNx .................................. 131
   EnableIntADC ................................ 108, 115
   EnableIntCANx ................................ 100
   EnableIntDCI ................................ 163
   EnableIntFLTA ................................ 188
   EnableIntFLT ................................ 189
   EnableIntCx .................................. 137
   EnableIntUtx ................................ 155
   EnableIntMCPWM ................................ 188
   EnableIntM2C .................................. 198
   EnableIntOCx .................................. 147
   EnableIntQEI .................................. 177
   EnableIntSIC .................................. 199
   EnableIntSPIx .................................. 171
   EnableIntTx .................................. 123
   EnableIntUtx ................................ 155
   EnableIntx .................................. 131
   End Of File .................................. 235
   Indicator ................................... 233
   Seek ........................................ 252
   Test For .................................... 239
Environment Function
   getenv ...................................... 289
   Environment Variable .................... 377
   EOF .......................................... 235
   ERANGE ...................................... 213
   erange ...................................... 333
   errno ........................................ 213, 333
   errno.h ........................................ 213, 333, 378
   EDOM ......................................... 213
   ERANGE ...................................... 213
   errno ........................................ 213
Error Codes ..................................... 213, 313
   Error Conditions .......................... 333
   Error Handler ................................ 286
Error Handling Functions
   clearerr ................................... 237
   feof ....................................... 239
   ferror ...................................... 240
   perror ...................................... 260
Error Indicator .................................. 233
Error Indicators
   Clearing .................................... 237, 266
   End Of File .................................. 237, 243
   Error ........................................ 237, 243
   Test For .................................... 240
Error Signal ..................................... 223
Errors, See errno.h
Errors, Testing For ......................... 213
16-Bit Language Tools Libraries

Example of Use
- ADC, 10-Bit ........................................ 116
- ADC, 12-Bit .................................... 109
- CAN ........................................ 101
- DCI .......................................... 164
- I2C .......................................... 200
- Input Capture ................................ 138
- Output Compare ............................ 148
- PWM ....................................... 190
- QEI Functions ............................... 178
- SPI ......................................... 172
- Timers ...................................... 124
- UART ..................................... 157

Exception Error ................................ 286
exit ........................................... 272, 278, 280, 288, 375
EXIT_FAILURE ............................. 278
EXIT_SUCCESS ................................ 278

File Access Functions
- FILE ....................................... 233, 234
- fopen ................................... 244
- fclose .................................... 238, 376
- freopen .................................. 250
- fprintf .................................. 242
- fwrite .................................... 241
- fgets ..................................... 243
- fread ..................................... 251
- feof ....................................... 240
- ferror .................................... 237
- flush ...................................... 241
- fputc ..................................... 244
- fseek ...................................... 252
- ftell ....................................... 255
- rewind .................................... 266
- fputc ...................................... 241
- fsetpos ................................... 253
- freopen .................................. 250
- fsetpos ................................... 253
- ftell ....................................... 255
- rewind .................................... 266

Exponential and Logarithmic Functions
- exp .......................................... 345
- expf ........................................ 346
- frexp ........................................ 352
- frexpf .................................... 353
- ldexp ....................................... 354
- ldexpf .................................... 355
- log .......................................... 357
- log10 ....................................... 358
- log10f ..................................... 359
- logf ......................................... 360
- modf ........................................ 361
- modff ....................................... 362

Exponential Function
- Double Floating Point ................. 345
- Single Floating Point ................. 346

F
- fabs ........................................ 347
- fabf ......................................... 347
- fclose .................................... 238, 376
- feof ....................................... 237, 239
- ferror .................................... 237, 240
- flush ...................................... 241, 381
- FFTComplex ............................. 64
- FFTComplexIP .......................... 66
- fgetc ...................................... 241, 379
- fgets ..................................... 242, 377
- fgetpos ................................ 243, 379
- flags ....................................... 261
- Field Access Functions
- fclose .................................... 238
- flush ...................................... 241
- fopen .................................... 244
- freopen .................................. 250
- setbuf ................................... 269
- setvbuf .................................. 270
- File Access Modes ..................... 233, 244

File Buffering
- Fully Buffered .......................... 233, 234
- Line Buffered ........................... 233, 234
- Unbuffered ............................... 233, 234

File Operations
- Remove ................................ 265
- Rename .................................. 265

File Positioning Functions
- fgetpos ................................ 242
- fseek ...................................... 252
- ftell ....................................... 255
- rewind .................................... 266
- FILENAME_MAX ......................... 235

File-Position Indicator ........................... 235

Filtering Functions .......................... 38
- FIR ......................................... 41
- FIRDecimate ............................. 42
- FIRDelayInit ............................ 43
- FIRInterpDelayInit ..................... 45
- FIRInterpolate ......................... 44
- FIRLattice ............................. 45
- FIRLMS ................................ 46
- FIRLMSNorm ......................... 47
- FIRStruct ............................. 40
- FIRStructInit ......................... 49
- IIRCanonic ............................ 50
- IIRCanonicInit ......................... 51
- IIRLattice ............................ 52
- IIRLatticeOCTAVE model ........... 56
- IIRLatticeInit ......................... 53
- IIRTransposed ......................... 54
- IIRTransposedInit .................... 55

FIR ......................................... 41
- FIRDecimate ............................. 42
- FIRDelayInit ............................ 43
- FIRInterpDelayInit ..................... 45
- FIRInterpolate ......................... 44
- FIRLattice ............................ 45
- FIRLMS ................................ 46
- FIRLMSNorm ......................... 47
- FIRStruct ............................. 40
- FIRStructInit ......................... 49
- flags ....................................... 261
Index

float.h ................................................................. 214
DBL_DIG ............................................................ 214
DBL_EPSILON ................................................... 214
DBL_MANT_DIG ................................................ 214
DBL_MAX .......................................................... 214
DBL_MAX_10_EXP ........................................... 215
DBL_MAX_EXP ................................................ 215
DBL_MIN .......................................................... 215
DBL_MIN_10_EXP ........................................... 215
DBL_MIN_EXP ................................................ 216
FLT_DIG ............................................................ 216
FLT_EPSILON ................................................... 216
FLT_MANT_DIG ................................................ 216
FLT_MAX .......................................................... 216
FLT_MAX_10_EXP ........................................... 216
FLT_MAX_EXP ................................................ 217
FLT_MIN .......................................................... 217
FLT_MIN_10_EXP ........................................... 217
FLT_MIN_EXP ................................................ 217
FLT_RADIX ....................................................... 217
FLT_ROUNDS ..................................................... 217
LDBL_DIG .......................................................... 217
LDBL_EPSILON ................................................ 218
LDBL_MANT_DIG ............................................. 218
LDBL_MAX ....................................................... 218
LDBL_MAX_10_EXP ........................................ 218
LDBL_MAX_EXP ................................................ 218
LDBL_MIN ....................................................... 218
LDBL_MIN_10_EXP ........................................ 218
LDBL_MIN_EXP ................................................ 219
Floating Point
Limits ............................................................. 214
No Conversion ................................................. 233
Types, Properties Of ........................................ 214
Floating Point, See float.h
Floating-Point Error Signal ......................... 224
floor ................................................................. 348

Double Floating Point ................................ 348
Single Floating Point ................................... 348
floor ................................................................. 348

FLT_DIG ............................................................ 216
FLT_EPSILON ................................................... 216
FLT_MANT_DIG ................................................ 216
FLT_MAX .......................................................... 216
FLT_MAX_10_EXP ........................................... 216
FLT_MAX_EXP ................................................ 217
FLT_MIN .......................................................... 217
FLT_MIN_10_EXP ........................................... 217
FLT_MIN_EXP ................................................ 217
FLT_RADIX ....................................................... 217

FLT_RADIX Digit
Number Of ...................................................... 214, 216, 218

FLT_ROUNDS ..................................................... 217
Flush .............................................................. 241, 288
fmod ............................................................... 349
fmodf .............................................................. 351
-fno-short-double ........................................ 214, 215, 216, 233
fopen .............................................................. 233, 244, 270, 378
FOPEN_MAX ................................................... 235
Form Feed ......................................................... 209

Format Specifiers ........................................... 261, 267
Formatted I/O Routines ................................ 233
Formatted Input/Output Functions
fprintf ............................................................ 246
fscanf ............................................................. 250
printf ............................................................ 261
scanf ............................................................. 267
sprintf ........................................................... 271
sscanf ............................................................. 271
vfprintf ........................................................... 275
vprintf ........................................................... 276
vsprintf ........................................................... 277

Formatted Text
Printing ........................................................ 271
Scanning ......................................................... 271

fraction and exponent function
Double Floating Point ................................ 352
Single Floating Point ...................................... 353

Fraction Digits ................................................. 261
fread .............................................................. 248, 379
free ................................................................. 289
Free Memory .................................................... 289
freopen ............................................................ 233, 250, 378
frexp ............................................................... 352
frexpf ............................................................... 353
fscanf ............................................................. 233, 250
fseek ............................................................... 252, 273, 377
fsetpos ............................................................ 253, 273, 377
ftell ............................................................... 255, 377

Full Buffering ................................................. 269, 270
Fully Buffered ................................................. 233, 234
fwrite .............................................................. 256

G
getc ................................................................. 258
getchar ............................................................ 259
getcSPIx ........................................................... 171
getcUARTx ....................................................... 155
getenv ............................................................ 289, 377
gets ............................................................... 259, 379
getsSPIx ........................................................... 170
getsUARTx ....................................................... 154
GMT .............................................................. 328
gmtime ............................................................ 328, 329

Graphical Character
Defined ......................................................... 206
Test for ......................................................... 206

Greenwich Mean Time .................................. 328

© 2005 Microchip Technology Inc.
16-Bit Language Tools Libraries

H

H modifier .................................................. 262, 267
HammingInit .................................................. 28
Handler
  Default .................................................. 223
  Error .................................................. 286
  Interrupt .............................................. 227
  Nested .................................................. 222
  Signal .................................................. 223, 228
  Signal Type ........................................... 223
Handling
  Interrupt Signal ...................................... 228
HanningInit .................................................. 28
Header Files
  assert.h ............................................... 203
  ctype.h ............................................... 204
  errno.h ................................................ 213, 333, 378
  float.h ............................................... 214
  limits.h ............................................. 219
  locale.h ............................................. 221
  math.h ............................................... 333
  setjmp.h ............................................. 222
  signal.h ............................................. 223
  stdarg.h ............................................. 229
  stddef.h ............................................. 231
  stdio.h .............................................. 233, 379
  stdlib.h ............................................. 278, 377, 380
  string.h ............................................. 302
  time.h .............................................. 325, 381
Heap ..................................................... 376
Helper Functions ....................................... 374
Hexadecimal ........................................... 262, 268, 299, 300
Hexadecimal Conversion ................................ 261
Hexadecimal Digit
  Defined ............................................. 210
  Test for ............................................ 210
Horizontal Tab ........................................ 209
Hour ..................................................... 325, 326, 330
HUGE_VAL .............................................. 333
Hyperbolic Cosine
  Double Floating Point ................................ 343
  Single Floating Point ................................ 344
Hyperbolic Functions
  cosh .................................................. 343
  coshf ............................................... 344
  sinh ................................................. 367
  sinhfl ............................................... 368
  tanh .................................................. 372
  tanhf ................................................. 373
Hyperbolic Sine
  Double Floating Point ................................ 367
  Single Floating Point ................................ 368
Hyperbolic Tangent
  Double Floating Point ................................ 372
  hyperbolic tangent ................................ 372
  Single Floating Point ................................ 373

I

I/O Port Functions
  Configure CN Interrupts ............................. 130
  Configure CN Pull-Ups ............................. 130
  Configure INT ....................................... 129
  Disable INT .......................................... 128
I/O Port Macros
  Disable CN Interrupts ......................... 131
  Disable Interrupts ............................... 132
  Enable CN Interrupts ............................. 131
  Enable Interrupts ................................. 131
  Set Interrupts Priority .......................... 132
I2C Functions
  Acknowledge I2C .................................. 192
  Close I2C ........................................... 191
  Configure I2C ...................................... 195
  Configure I2C Interrupt .......................... 191
  Data Ready I2C .................................... 192
  Example of Use .................................... 200
  Idle I2C ............................................. 192
  Master I2C Get String ......................... 193
  Master I2C Put String ............................ 193
  Not Acknowledge I2C .............................. 194
  Read Master I2C .................................. 194
  Read Slave I2C ..................................... 197
  Restart I2C ......................................... 196
  Slave I2C Get String ............................. 196
  Slave I2C Put String .............................. 197
  Start I2C ............................................ 198
  Stop I2C .............................................. 198
  Write Master I2C .................................. 194
  Write Slave I2C .................................... 197
I2C Macros
  Disable Master I2C Interrupt ................... 198
  Disable Slave I2C .................................. 199
  Enable Master I2C Interrupt .................... 198
  Enable Slave I2C .................................. 199
  Set Master I2C Interrupt Priority ............. 199
  Set Slave I2C Interrupt Priority .............. 199
  IdelI2C .............................................. 192
  IFFTComplex ....................................... 67
  IFFTComplexIP ................................... 69
  Ignore Signal ....................................... 223
  IIRCanonic ......................................... 50
  IIRCanonicInit ..................................... 51
  IIRLattice .......................................... 52
  IIRLattice OCTAVE model ...................... 56
  IIRlatticeInit ..................................... 53
  IIRTransposed ...................................... 54
  IIRTransposedInit ................................. 55
  Illegal Instruction Signal ..................... 225
  Implementation-Defined Limits, See limits.h
  Indicator
    End Of File ....................................... 233, 235
    Error ............................................... 233, 240
    File Position .................................... 233, 241, 242, 247, 248, 253, 256
    Infinity ............................................ 333
    Input and Output, See stdio.h

DS51456C-page 410 © 2005 Microchip Technology Inc.
16-Bit Language Tools Libraries

Line Buffered ................................................................. 233, 234
Line Buffering ................................................................. 270
Il modifier .................................................................. 262, 267
LLONG_MAX .............................................................. 219
LLONG_MIN .............................................................. 220
Load Exponent Function
  Double Floating Point ..................................................... 354
  Single Floating Point ..................................................... 355
Local Time .................................................................. 327, 329, 330
Locale, C ....................................................................... 204, 221
Locale, Other .............................................................. 221
locale.h ....................................................................... 221
localeconv ................................................................... 221
Localization, See locale.h
localtime ...................................................................... 327, 328, 329
Locate Character ............................................................ 309
log ............................................................................. 357
log10 .......................................................................... 358
log10f .......................................................................... 359
Logarithm Function
  Double Floating Point ..................................................... 358
  Single Floating Point ..................................................... 359
Logarithm Function, Natural
  Double Floating Point ..................................................... 357
  Single Floating Point ..................................................... 360
logf ............................................................................. 360
Logic Errors, Debugging .................................................. 203
Long Double Precision Floating Point
  Machine Epsilon ......................................................... 218
  Maximum Exponent (base 10) ....................................... 218
  Maximum Exponent (base 2) ....................................... 218
  Maximum Value ......................................................... 218
  Minimum Exponent (base 10) ....................................... 218
  Minimum Exponent (base 2) ....................................... 219
  Minimum Value ......................................................... 218
  Number of Binary Digits ............................................. 218
  Number of Decimal Digits ......................................... 217
long double Type ......................................................... 233
long int
  Maximum Value ......................................................... 220
  Minimum Value ......................................................... 220
long long int
  Maximum Value ......................................................... 219
  Minimum Value ......................................................... 220
long long unsigned int
  Maximum Value ......................................................... 221
long unsigned int
  Maximum Value ......................................................... 221
LONG_MAX .............................................................. 220
LONG_MIN .............................................................. 220
longjmp................................................................. 222
Lower Case Alphabetic Character
  Convert To ................................................................. 211
  Defined ................................................................... 207
  Test for ................................................................... 207
lseek ............................................................................ 377

M
  Machine Epsilon ......................................................... 218
  Double Floating Point ..................................................... 214
  Long Double Floating Point ........................................ 218
  Single Floating Point .................................................. 216
Magnitude ................................................................. 333, 345, 346, 349, 351, 356, 367, 368
malloc ........................................................................ 289, 292, 375, 380
Mapping Characters ..................................................... 204
MastergetsI2C ............................................................. 193
MasterputsI2C ............................................................. 193
MasterReadI2C ............................................................ 194
MasterWriteI2C ........................................................... 194
Math Exception Error ..................................................... 286
math.h ........................................................................ 333
acos ........................................................................... 333
acosf ........................................................................... 334
asin ............................................................................ 335
asinf ............................................................................ 335
atan ............................................................................ 336
atan2 ........................................................................... 337
atan2f ........................................................................... 339
atanf ............................................................................ 337
ceil ............................................................................. 340
cel1f ............................................................................. 341
cos .............................................................................. 341
cosh ............................................................................. 342
coshf ............................................................................. 343
coshf ............................................................................. 344
cosf ............................................................................. 345
exf .............................................................................. 346
exp .............................................................................. 346
fabs ................................................................. 347
fabsf ............................................................................. 347
floor ............................................................................. 348
floorf ............................................................................. 348
fmod ............................................................................. 349
fmodf ............................................................................. 351
frem ............................................................................. 352
fremf ............................................................................. 353
HUGE_VAL ................................................................. 333
ldexp ............................................................................ 334
ldexpf ............................................................................ 335
log .............................................................................. 335
log10 .......................................................................... 336
log10f .......................................................................... 338
logf .............................................................................. 339
mod .............................................................................. 340
modf ............................................................................. 341
modff ............................................................................. 342
pow .............................................................................. 343
powf ............................................................................. 344
powf ............................................................................. 345
sin .............................................................................. 345
sinf .............................................................................. 346
sinh ............................................................................. 346
sinhf ............................................................................. 347
sqrt .............................................................................. 348
sqrtf ............................................................................. 349
tan .............................................................................. 350
tanf .............................................................................. 351
tanh .............................................................................. 352
tanhf ............................................................................. 353

Mathematical Functions, See math.h
<table>
<thead>
<tr>
<th><strong>Index</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Functions .......................................................... 31</td>
</tr>
<tr>
<td>MatrixAdd ........................................................................ 33</td>
</tr>
<tr>
<td>MatrixInvert ..................................................................... 37</td>
</tr>
<tr>
<td>MatrixMultiply .............................................................. 34</td>
</tr>
<tr>
<td>MatrixScale ...................................................................... 35</td>
</tr>
<tr>
<td>Matrix Subtract .................................................................. 35</td>
</tr>
<tr>
<td>Matrix Transpose ............................................................. 36</td>
</tr>
<tr>
<td>Maximum Value .................................................................</td>
</tr>
<tr>
<td>Multibyte Character ......................................................... 279</td>
</tr>
<tr>
<td>Maximum Value ..................................................................</td>
</tr>
<tr>
<td>Double Floating-Point Exponent (base 10) ......................... 215</td>
</tr>
<tr>
<td>Double Floating-Point Exponent (base 2) ............................. 215</td>
</tr>
<tr>
<td>Long Double Floating-Point Exponent (base 10) ................. 218</td>
</tr>
<tr>
<td>Long Double Floating-Point Exponent (base 2) ..................... 218</td>
</tr>
<tr>
<td>Multibyte Character ......................................................... 220</td>
</tr>
<tr>
<td>rand .................................................................................. 279</td>
</tr>
<tr>
<td>Single Floating-Point Exponent (base 10) ............................ 216</td>
</tr>
<tr>
<td>Single Floating-Point Exponent (base 2) ............................... 217</td>
</tr>
<tr>
<td>Type char ........................................................................... 219</td>
</tr>
<tr>
<td>Type Double ....................................................................... 214</td>
</tr>
<tr>
<td>Type int .............................................................................. 219</td>
</tr>
<tr>
<td>Type Long Double ............................................................ 218</td>
</tr>
<tr>
<td>Type long int ..................................................................... 220</td>
</tr>
<tr>
<td>Type long long int .......................................................... 220</td>
</tr>
<tr>
<td>Type long unsigned int .................................................... 221</td>
</tr>
<tr>
<td>Type long unsigned int .................................................... 221</td>
</tr>
<tr>
<td>Type short int .................................................................... 220</td>
</tr>
<tr>
<td>Type signed char .............................................................. 220</td>
</tr>
<tr>
<td>Type Single ....................................................................... 216</td>
</tr>
<tr>
<td>Type unsigned char .......................................................... 221</td>
</tr>
<tr>
<td>Type unsigned int ............................................................ 221</td>
</tr>
<tr>
<td>Type unsigned short int ..................................................... 221</td>
</tr>
<tr>
<td>MB_CUR_MAX ....................................................................... 279</td>
</tr>
<tr>
<td>MB_LEN_MAX ..................................................................... 220</td>
</tr>
<tr>
<td>mblen ................................................................................. 293</td>
</tr>
<tr>
<td>mbstowcs .......................................................................... 293</td>
</tr>
<tr>
<td>mbtowc .............................................................................. 293</td>
</tr>
<tr>
<td>memchr ............................................................................. 302</td>
</tr>
<tr>
<td>memcmp ............................................................................ 303</td>
</tr>
<tr>
<td>memcmp .............................................................................. 305</td>
</tr>
<tr>
<td>memmove ........................................................................... 306</td>
</tr>
<tr>
<td>Memory .............................................................................</td>
</tr>
<tr>
<td>Allocate .............................................................................. 286, 292</td>
</tr>
<tr>
<td>Deallocate ........................................................................ 289</td>
</tr>
<tr>
<td>Free .................................................................................... 289</td>
</tr>
<tr>
<td>Reallocate ......................................................................... 295</td>
</tr>
<tr>
<td>memset ............................................................................... 307</td>
</tr>
<tr>
<td>Message .............................................................................</td>
</tr>
<tr>
<td>Arithmetic Error ............................................................... 224</td>
</tr>
<tr>
<td>Interrupt ........................................................................... 225</td>
</tr>
<tr>
<td>Invalid Executable Code ..................................................... 225</td>
</tr>
<tr>
<td>Invalid Storage Request .................................................... 226</td>
</tr>
<tr>
<td>Termination Request .......................................................... 226</td>
</tr>
<tr>
<td>Microchip Internet Web Site ................................................ 5</td>
</tr>
<tr>
<td>Minimum Value ....................................................................</td>
</tr>
<tr>
<td>Double Floating-Point Exponent (base 10) .......................... 215</td>
</tr>
<tr>
<td>Double Floating-Point Exponent (base 2) ............................ 215</td>
</tr>
<tr>
<td>Long Double Floating-Point Exponent (base 10) ................. 218</td>
</tr>
<tr>
<td>Long Double Floating-Point Exponent (base 2) ..................... 219</td>
</tr>
<tr>
<td>Single Floating-Point Exponent (base 10) ......................... 217</td>
</tr>
<tr>
<td>Single Floating-Point Exponent (base 2) ............................. 217</td>
</tr>
<tr>
<td>Type char ........................................................................... 219</td>
</tr>
<tr>
<td>Type Double ....................................................................... 215</td>
</tr>
<tr>
<td>Type int .............................................................................. 219</td>
</tr>
<tr>
<td>Type Long Double ............................................................ 218</td>
</tr>
<tr>
<td>Type long int ..................................................................... 220</td>
</tr>
<tr>
<td>Type long long int .......................................................... 220</td>
</tr>
<tr>
<td>Type short int .................................................................... 220</td>
</tr>
<tr>
<td>Type signed char .............................................................. 220</td>
</tr>
<tr>
<td>Type Single ....................................................................... 217</td>
</tr>
<tr>
<td>Minute ................................................................................ 325, 326, 331</td>
</tr>
<tr>
<td>mktime ............................................................................... 330</td>
</tr>
<tr>
<td>mod ..................................................................................... 361</td>
</tr>
<tr>
<td>modf .................................................................................... 362</td>
</tr>
<tr>
<td>modulus function ...............................................................</td>
</tr>
<tr>
<td>Double Floating Point ....................................................... 361</td>
</tr>
<tr>
<td>Single Floating Point ....................................................... 362</td>
</tr>
<tr>
<td>Month ............................................................................... 325, 326, 330, 331</td>
</tr>
<tr>
<td>-msmart-io ......................................................................... 233</td>
</tr>
<tr>
<td>Multibyte Character ......................................................... 279, 293, 301</td>
</tr>
<tr>
<td>Maximum Number of Bytes ................................................ 220</td>
</tr>
<tr>
<td>Multibyte String ............................................................... 293, 301</td>
</tr>
<tr>
<td>N .......................................................................................</td>
</tr>
<tr>
<td>NaN .................................................................................... 333</td>
</tr>
<tr>
<td>Natural Logarithm .............................................................</td>
</tr>
<tr>
<td>Double Floating Point ....................................................... 357</td>
</tr>
<tr>
<td>Single Floating Point ....................................................... 360</td>
</tr>
<tr>
<td>NDEBUG ............................................................................. 203</td>
</tr>
<tr>
<td>Nearest Integer Functions ..................................................</td>
</tr>
<tr>
<td>ceil .................................................................................... 340</td>
</tr>
<tr>
<td>ceilf ................................................................................... 341</td>
</tr>
<tr>
<td>floor ................................................................................... 346</td>
</tr>
<tr>
<td>floorf .................................................................................. 346</td>
</tr>
<tr>
<td>Nested Signal Handler ....................................................... 222</td>
</tr>
<tr>
<td>Newline .......................................................... 209, 233, 243, 248, 259, 260, 264</td>
</tr>
<tr>
<td>No Buffering ................................................................. 233, 234, 269, 270</td>
</tr>
<tr>
<td>Non-Local Jumps, See setjmp.h ........................................ 194</td>
</tr>
<tr>
<td>NULL ............................................................................... 221, 231, 235, 279, 302, 326</td>
</tr>
<tr>
<td>O .......................................................................................</td>
</tr>
<tr>
<td>Object Module Format ..................................................... 7</td>
</tr>
<tr>
<td>Octal ............................................................................... 282, 286, 289, 300</td>
</tr>
<tr>
<td>Octal Conversion ............................................................ 281</td>
</tr>
<tr>
<td>offset ................................................................. 381, 382, 383</td>
</tr>
<tr>
<td>OMF ..................................................................................... 7</td>
</tr>
<tr>
<td>open ............................................................................... 378</td>
</tr>
<tr>
<td>OpenADC10 ....................................................................... 111</td>
</tr>
<tr>
<td>OpenADC12 ....................................................................... 104</td>
</tr>
<tr>
<td>OpenCapturex ................................................................. 135</td>
</tr>
<tr>
<td>OpenDCI .......................................................................... 159</td>
</tr>
<tr>
<td>OpenI2C .......................................................................... 195</td>
</tr>
</tbody>
</table>

© 2005 Microchip Technology Inc. DS1456C-page 413
16-Bit Language Tools Libraries

OpenMCPWM ................................................................. 180
OpenOCx ................................................................. 141
OpenQEI ................................................................. 175
OpenSPIx ................................................................. 168
OpenTimerx .............................................................. 119
OpenTimerxx ............................................................. 120
OpenUART ............................................................... 151
OpenXLCD ............................................................... 83

Output Compare Functions
  Close Compare ....................................................... 139
  Configure Compare ............................................... 140
  Configure Compare Interrupt .................................... 140
  Example of Use ................................................... 148
  Read Compare Duty Cycle ....................................... 144
  Read Compare Duty Cycle - PWM mode ....................... 143
  Set Compare Duty Cycle ......................................... 146
  Set Compare Duty Cycle - PWM mode ......................... 145

Output Compare Macros
  Disable Compare Interrupt ...................................... 147
  Enable Compare Interrupt ....................................... 147
  Set Compare Interrupt Priority .................................. 147

Overflow Errors 213, 333, 345, 346, 354, 356, 363, 364
Overlap ........................................................................ 305, 306, 308, 311, 311, 314, 317

P

Pad Characters .......................................................... 261
Percent ......................................................................... 262, 267, 268, 331
Peripheral Libraries, 16-Bit ........................................... 81
 perror ......................................................................... 260
pic30-libs ...................................................................... 374
  _exit ......................................................................... 375
  brk .......................................................................... 375
  close ........................................................................ 376
  getenv ................................................................. 377
  lseek ......................................................................... 377
  open ......................................................................... 378
  read ........................................................................ 378
  remove ................................................................. 379
  rename ................................................................. 379
  sbrk ......................................................................... 380
  system ...................................................................... 380
  time ......................................................................... 381
  write ....................................................................... 381
 Plus Sign ........................................................................ 261
 Pointer, Temporary .................................................... 295
 PORStatReset ........................................................... 127
 pow ................................................................. 363
 Power Function
  Double Floating Point ............................................... 363
  Single Floating Point ............................................... 364
 Power Functions
  pow .......................................................................... 363
  powf ................................................................. 364
  powlf ............................................................... 364
 precision .................................................................. 261
 Prefix ........................................................................ 210, 261
 Print Formats ........................................................... 233
 Printable Character
  Defined ................................................................. 208
  Test for ............................................................... 208

printf ................................................................. 233, 261
Processor Clocks per Second ........................................... 325
Processor Time .......................................................... 325, 326
Pseudo-Random Number ............................................... 295, 297
ptrdiff_t ................................................................. 231

Punctuation Character
  Defined ................................................................. 208
  Test for ............................................................... 208

Pushed Back .................................................................. 273
putc ........................................................................... 263
putchar ................................................................. 264
putcSPIx ................................................................. 171
putcUARTx .............................................................. 155
putsXLCD .................................................................. 384
puts ................................................................. 264
putcSPIx ................................................................. 170
putcUARTx .............................................................. 154
putsXLCD .................................................................. 384
PverrideMCPWM ...................................................... 182

PWM Functions
  Close PWM .................................................................. 179
  Configure Override .................................................. 182
  Configure PWM ....................................................... 180
  Configure PWM Interrupt ......................................... 179
  Example of Use ...................................................... 190
  Set PWM Dead Time Assignment ............................. 184
  Set PWM Duty Cycle ............................................... 185
  Set PWM Duty Cycle ............................................... 183
  Set PWM FaultA ...................................................... 186
  Set PWM FaultB ...................................................... 187

PWM Macros
  Disable FLTA Interrupt ............................................ 188
  Disable FLTB Interrupt ............................................ 189
  Disable PWM Interrupt ............................................ 188
  Enable FLTA Interrupt ............................................ 188
  Enable FLTB Interrupt ............................................ 189
  Enable PWM Interrupt ............................................ 188
  Set FLTA Interrupt Priority ..................................... 189
  Set FLTB Interrupt Priority ..................................... 189
  Set PWM Interrupt Priority ..................................... 188

Q

QEI Functions
  Close QEI ............................................................... 174
  Configure QEI ........................................................ 175
  Configure QEI Interrupt .......................................... 174
  Example of Use ..................................................... 178
  Read QEI Position Count ....................................... 176
  Write QEI Position Count ....................................... 177

QEI Macros
  Disable QEI Interrupt ............................................. 177
  Enable QEI Interrupt .............................................. 177
  Set QEI Interrupt Priority ....................................... 177
qsort ........................................................................ 284, 294
Quick Sort ................................................................... 294
Index

R
Radix ................................................................. 217
raise ............................................................... 223, 224, 225, 226, 227, 228
rand ................................................................. 295, 297
RAND_MAX ......................................................... 279, 295
Range ................................................................. 268
Range Error ... 213, 299, 300, 343, 344, 345, 346, 354,
356, 363, 364, 367, 368
read ................................................................. 379
ReadADC10 .......................................................... 114
ReadADC12 .......................................................... 107
ReadAddrXLCD ..................................................... 85
ReadCapture ....................................................... 136
ReadDataXLCD ..................................................... 85
ReadDCI ............................................................. 162
ReadDCOcxPWM .................................................. 143
Reading, Recommended ........................................... 4
ReadQEI ............................................................. 176
ReadRegOcx ......................................................... 144
ReadSPIx ........................................................... 167
ReadTimerx ......................................................... 121
ReadTimex ........................................................ 122
ReadUARTx ........................................................ 153
realloc ............................................................. 289, 295
Reallocate Memory ................................................ 295
Rebuilding the libpic30 library .................................. 374
Registered Functions ........................................... 280, 288
Remainder
Double Floating Point .......................................... 349
Single Floating Point .......................................... 351
Remainder Functions
fmod ................................................................. 349
fmodf ............................................................... 351
remove ............................................................. 265, 379
rename ............................................................. 265, 379
Reset ............................................................... 279, 301
Reset File Pointer ................................................ 266
Reset/Control Functions
Low Voltage Detect ............................................. 125
Master Clear Reset ............................................. 126
Reset from Brown-out ........................................ 125
Reset from Power-on ........................................ 125
Wake-up from Sleep ........................................... 127
Watchdog Timer Time-out .................................... 126
Watchdog Timer Wake-up ..................................... 126
Reset/Control Macros
Disable All Interrupts ........................................ 127
Disable Watchdog Timer ..................................... 128
Enable Watchdog Timer ...................................... 128
Reset BOR bit ................................................... 127
Reset POR bit ................................................... 127
Restart12C ........................................................ 196
rewind ............................................................. 266, 273, 377
Rounding Mode .................................................. 217
S
sbrk ................................................................. 376, 380
Scan Formats .................................................... 233
scanf .............................................................. 233, 267
SCHAR_MAX ....................................................... 220
SCHAR_MIN ........................................................ 220
Search Functions
memchr ........................................................... 302
strchr ............................................................. 309
strcspn ........................................................... 312
strchr ............................................................. 319
strchr ............................................................. 320
strspn ............................................................. 321
strstr .............................................................. 322
strtok .............................................................. 323
Second ........................................................... 325, 326, 328, 331
Seed ............................................................... 295, 297
Seek
From Beginning of File ....................................... 252
From Current Position ....................................... 252
From End Of File .............................................. 252
SEEK_CUR ........................................................ 235, 252
SEEK_END ......................................................... 236, 252
SEEK_SET ........................................................ 236, 252
setbuf ............................................................. 233, 234, 269
SetCGRamAddr .................................................. 86
SetChanADC10 .................................................. 114
SetChanADC12 .................................................. 107
SetDCMCPWM ................................................... 183
SetDCOCxPWM .................................................. 145
SetDDRRamAddr ................................................ 86
setjmp ............................................................. 222
setjmp.h ........................................................... 222
jmp_buf ........................................................... 222
longjmp ........................................................... 222
setjump ........................................................... 222
setlocale .......................................................... 221
SetMCPWMDeadTimeAssignment ......................... 184
SetMCPWMDeadTimeGeneration ............................ 185
SetMCPWMFaultIA ............................................ 186
SetMCPWMFaultIB ............................................ 187
SetPointIntUxRX ............................................... 156
SetPriorityIntADC .......................................... 108, 115
SetPriorityIntCANx .......................................... 100
SetPriorityIntDCI .............................................. 163
SetPriorityIntFLTA .......................................... 189
SetPriorityIntFLTB .......................................... 189
SetPriorityIntCx .............................................. 137, 147
SetPriorityIntMCPWM ....................................... 188
SetPriorityIntMPC2 .......................................... 199
SetPriorityIntQEI .............................................. 177
SetPriorityIntSI2C ............................................ 199
SetPriorityIntSPIx .......................................... 171
SetPriorityIntTx .............................................. 124
SetPriorityIntUxTX .......................................... 156
SetPriorityIntx ................................................. 132
SetPulseOCx .................................................... 146
setvbuf ........................................................... 233, 234, 270
short int
Maximum Value .................................................. 220
Minimum Value .................................................. 220
SHRT_MAX ....................................................... 220
SHRT_MIN ....................................................... 220
sig_atomic_t ..................................................... 223
SIG_DFL .......................................................... 223
16-Bit Language Tools Libraries

SIG_ERR ................................................................. 223
SIG_IGN ................................................................. 223
SIGABRT ............................................................... 224
SIGFPE ............................................................... 224
SIGILL ................................................................. 225
SIGINT ............................................................... 225
Signal
Abnormal Termination........................................... 224
Error .................................................................. 223
Floating-Point Error ........................................... 224
Ignore .................................................................. 223
Illegal Instruction ................................................. 225
Interrupt............................................................. 225
Reporting............................................................ 227
Termination Request............................................. 226
signal............................................................... 224, 225, 226, 228
Signal Handler .................................................. 223, 228
Signal Handler Type .......................................... 223
Signal Handling, See signal.h
signal.h............................................................... 223
raise .................................................................. 227
SIG_DFL ............................................................... 223
SIG_ERR ............................................................... 223
SIG_IGN ............................................................... 223
SIGABRT ............................................................. 224
SIGFPE ............................................................... 224
SIGILL ................................................................. 225
SIGINT ............................................................... 225
signal ............................................................... 228
SIGSEGV ........................................................ 226
SIGTERM .......................................................... 226
signed char
Maximum Value ................................................. 220
Minimum Value ................................................. 220
SIGSEGV .......................................................... 226
SIGTERM .......................................................... 226
sim30 simulator .................................................. 374
sin ................................................................. 365
dsine
Double Floating Point ........................................... 365
Single Floating Point .......................................... 366
sinf ................................................................. 366
Single Precision Floating Point
Machine Epsilon.................................................. 216
Maximum Exponent (base 10) ....................... 216
Maximum Exponent (base 2) ......................... 217
Maximum Value ................................................. 216
Minimum Exponent (base 10) ....................... 217
Minimum Exponent (base 2) ......................... 217
Minimum Value ................................................. 217
Number of Binary Digits ................................ 216
Number of Decimal Digits ................................ 216
sinh ................................................................. 367
sinhf ............................................................... 368
size ................................................................. 368
size_t .............................................................. 231, 234, 278, 302, 325
sizeof ............................................................ 231, 234, 278, 302, 325
SlaveRead2C ..................................................... 196
SlaveWrite2C ..................................................... 197
SlaveputsI2C.......................................................... 197
SlaveRead2C ..................................................... 197
Sort, Quick ....................................................... 294
Source File Name .............................................. 203
Source Line Number .......................................... 203
Space ............................................................. 261
Space Character
Defined ................................................................ 209
Test for ................................................................ 209
Specifiers .......................................................... 261, 267
SPI Functions
Close SPI .......................................................... 166
Configure SPI .................................................. 168
Configure SPI Interrupt .................................. 166
Example of Use .................................................. 172
Read SPI RX Buffer ......................................... 167
SPI Buffer Status .............................................. 167
SPI Get Character ............................................. 171
SPI Get String .................................................. 170
SPI Put Character ............................................. 171
SPI Put String ................................................... 170
Write SPI TX Buffer ........................................... 168
SPI Macros
Disable SPI Interrupt ....................................... 171
Enable SPI Interrupt ......................................... 171
Set SPI Interrupt Priority .................................. 171
sprintf ............................................................. 233, 271
sqrt ................................................................. 369
sqrtf ............................................................... 370
Square Root Function
Double Floating Point ...................................... 369
Single Floating Point ........................................ 370
Square Root Functions
sqrt ................................................................. 369
sqrtf ............................................................... 370
srand ............................................................... 297
sscanf .............................................................. 233, 271
Stack ............................................................... 376
Standard C Library ........................................... 201
Standard C Locale ............................................. 204
Standard Error .................................................. 233, 236
Standard Input .................................................. 233, 236
Standard Output ................................................ 233, 236
Start2C ............................................................ 198
Start-up ............................................................ 233
stdarg.h .................................................................. 229
va_arg ............................................................. 229
va_end ............................................................ 231
va_list ............................................................ 229
va_start .......................................................... 231
stddef.h .................................................................. 231
NULL ............................................................... 231
offsetof ............................................................. 232
ptrdiff_t .......................................................... 231
size_t .............................................................. 231
wchar_t ............................................................ 231
stderr ............................................................. 203, 233, 235, 236, 260
stdin .............................................................. 233, 235, 236, 259, 267
### 16-Bit Language Tools Libraries

<table>
<thead>
<tr>
<th>String Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>313</td>
</tr>
<tr>
<td>Search</td>
<td>322</td>
</tr>
<tr>
<td>Transform</td>
<td>324</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String Functions, See string.h</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>string.h</td>
<td>302</td>
</tr>
<tr>
<td>memchr</td>
<td>302</td>
</tr>
<tr>
<td>memcmp</td>
<td>303</td>
</tr>
<tr>
<td>memmove</td>
<td>306</td>
</tr>
<tr>
<td>memset</td>
<td>307</td>
</tr>
<tr>
<td>NULL</td>
<td>302</td>
</tr>
<tr>
<td>size_t</td>
<td>302</td>
</tr>
<tr>
<td>strcat</td>
<td>308</td>
</tr>
<tr>
<td>strchr</td>
<td>309</td>
</tr>
<tr>
<td>strcmp</td>
<td>310</td>
</tr>
<tr>
<td>strcpy</td>
<td>311</td>
</tr>
<tr>
<td>strcspn</td>
<td>312</td>
</tr>
<tr>
<td>strerror</td>
<td>313</td>
</tr>
<tr>
<td>strlen</td>
<td>313</td>
</tr>
<tr>
<td>strncmp</td>
<td>314</td>
</tr>
<tr>
<td>strncpy</td>
<td>316</td>
</tr>
<tr>
<td>strncmp</td>
<td>316</td>
</tr>
<tr>
<td>strpbrk</td>
<td>317</td>
</tr>
<tr>
<td>strrchr</td>
<td>318</td>
</tr>
<tr>
<td>strspn</td>
<td>320</td>
</tr>
<tr>
<td>strstr</td>
<td>321</td>
</tr>
<tr>
<td>strtok</td>
<td>322</td>
</tr>
<tr>
<td>strxfrm</td>
<td>323</td>
</tr>
<tr>
<td>strtof</td>
<td>282, 297</td>
</tr>
<tr>
<td>strtol</td>
<td>283, 299</td>
</tr>
<tr>
<td>strtof</td>
<td>300</td>
</tr>
<tr>
<td>struct lconv</td>
<td>324</td>
</tr>
<tr>
<td>Substrings</td>
<td>325</td>
</tr>
<tr>
<td>Subtracting Pointers</td>
<td>326</td>
</tr>
<tr>
<td>Successful Termination</td>
<td>327</td>
</tr>
<tr>
<td>system</td>
<td>301, 380</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tab</td>
<td>209</td>
</tr>
<tr>
<td>tan</td>
<td>371</td>
</tr>
<tr>
<td>tanf</td>
<td>371</td>
</tr>
<tr>
<td>tangent</td>
<td>371</td>
</tr>
<tr>
<td>Double Floating Point</td>
<td>371</td>
</tr>
<tr>
<td>Single Floating Point</td>
<td>372</td>
</tr>
<tr>
<td>tanh</td>
<td>373</td>
</tr>
<tr>
<td>tanhf</td>
<td>373</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Temporary Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>272, 288</td>
</tr>
<tr>
<td>Filename</td>
<td>235, 273</td>
</tr>
<tr>
<td>Pointer</td>
<td>295</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Termination</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Message</td>
<td>226</td>
</tr>
<tr>
<td>Request Signal</td>
<td>226</td>
</tr>
<tr>
<td>Successful</td>
<td>278</td>
</tr>
<tr>
<td>Unsuccessful</td>
<td>278</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Text Mode</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text Streams</td>
<td>233</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time Functions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close</td>
<td>117</td>
</tr>
<tr>
<td>Close 32-bit</td>
<td>117</td>
</tr>
<tr>
<td>Configure Interrupt</td>
<td>118</td>
</tr>
<tr>
<td>Configure Interrupt 32-bit</td>
<td>119</td>
</tr>
<tr>
<td>Example of Use</td>
<td>124</td>
</tr>
<tr>
<td>Open</td>
<td>119</td>
</tr>
<tr>
<td>Open 32-bit</td>
<td>120</td>
</tr>
<tr>
<td>Read</td>
<td>121</td>
</tr>
<tr>
<td>Read 32-bit</td>
<td>122</td>
</tr>
<tr>
<td>Write</td>
<td>122</td>
</tr>
<tr>
<td>Write 32-bit</td>
<td>123</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Timer Macros</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable Interrupt</td>
<td>123</td>
</tr>
<tr>
<td>Enable Interrupt</td>
<td>123</td>
</tr>
<tr>
<td>Set Interrupt Priority</td>
<td>124</td>
</tr>
</tbody>
</table>

| TMP_MAX                         | 236  |

| tmfile                          | 272  |
| tmnam                           | 273  |
| Tokens                          | 323  |
| toler                           | 211  |
| toupper                         | 212  |
| Transferring Control            | 222  |
Window Functions

- BartlettInit......................................................... 26
- BlackmanInit ................................................... 27
- HammingInit...................................................... 28
- HanningInit...................................................... 28
- KaiserInit ......................................................... 29
- VectorWindow.................................................... 29

write ........................................................................ 381
WriteCmdXLCD....................................................... 87
WriteDataXLCD....................................................... 86
WriteDCI ............................................................... 162
WriteQEIX ............................................................ 177
WriteSPIx ............................................................. 168
WriteTimerx ......................................................... 122
WriteTimerxx ......................................................... 123
WriteUARTx ......................................................... 153
WWW Address...................................................... 5

Y
Year................................................................. 325, 326, 331

Z
Zero ........................................................................ 333
Zero, divide by ..................................................... 224, 227, 286