



dsPIC[™] 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.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, MPLAB, PIC, PICmicro, PICSTART, PRO MATE and PowerSmart are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


AmpLab, FilterLab, microID, MXDEV, MXLAB, PICMASTER, SEEVAL, SmartShunt and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Application Maestro, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, PICKit, PICDEM, PICDEM.net, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, rLAB, rPIC, Select Mode, SmartSensor, SmartTel and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

Serialized Quick Turn Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

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

© 2003, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
== ISO/TS 16949:2002 ==

Microchip received ISO/TS-16949:2002 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona and Mountain View, California in October 2003. The Company's quality system processes and procedures are for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, non-volatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Table of Contents

Preface	xi
Chapter 1. Library Overview	
1.1 Introduction	1
1.2 Highlights	1
1.3 Startup Code	1
1.4 DSP Library	1
1.5 dsPIC Peripheral Libraries	2
1.6 Standard C Libraries (with Math Functions)	2
Chapter 2. DSP Library	
2.1 Introduction	3
2.2 Highlights	3
2.3 Using the DSP Library	4
2.4 Vector Functions	6
2.5 Window Functions	20
2.6 Matrix Functions	24
2.7 Filtering Functions	31
2.8 Transform Functions	50
Chapter 3. dsPIC Peripheral Libraries	
3.1 Introduction	63
3.2 Highlights	63
3.3 Using the dsPIC Peripheral Libraries	64
3.4 External LCD Functions	64
3.5 CAN Functions	69
3.6 ADC12 Functions	82
3.7 ADC10 Functions	89
3.8 Timer Functions	96
3.9 Reset/Control Functions	103
3.10 I/O Port Functions	107
3.11 Input Capture Functions	111
3.12 Output Compare Functions	117
3.13 UART Functions	126
3.14 DCI Functions	134
3.15 SPI Functions	141
3.16 QEI Functions	148
3.17 PWM Functions	153
3.18 I2C Functions	164
Chapter 4. Standard C Libraries with Math Functions	

4.1 Introduction	173
4.2 Highlights	173
4.3 Using the Standard C Libraries	174
4.4 <assert.h> diagnostics	174
4.5 <ctype.h> character handling	175
4.6 <errno.h> errors	185
4.7 <float.h> floating-point characteristics	185
4.8 <limits.h> implementation-defined limits	190
4.9 <locale.h> localization	193
4.9 <setjmp.h> non-local jumps	193
4.10 <signal.h> signal handling	194
4.11 <stdarg.h> variable argument lists	200
4.12 <stddef.h> common definitions	202
4.13 <stdio.h> input and output	203
4.14 <stdlib.h> utility functions	246
4.15 <string.h> string functions	270
4.16 <time.h> date and time functions	292
4.17 <math.h> mathematical functions	300
4.18 pic30-libs	339
Appendix A. ASCII Character Set	347
Index	349
Worldwide Sales and Service	365

Preface

INTRODUCTION

The purpose of this document is to define and describe the libraries that are available for use with Microchip Technology's dsPIC language tools, based on GNU technology. The related language tools are:

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

HIGHLIGHTS

Topics covered in this chapter are:

- About this Guide
- Recommended Reading
- Warranty Registration
- Troubleshooting
- The Microchip Internet Web Site

ABOUT THIS GUIDE

Document Layout

This document describes how to use GNU language tools to write code for dsPIC microcontroller 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: dsPIC Peripherals Libraries** – lists the library functions and macros for dsPIC device software and hardware peripheral operation.
- **Chapter 4: Standard C Library with Math Functions** – lists the library functions and macros for standard C operation.

Conventions Used in this Guide

This manual uses the following documentation conventions:

TABLE 1: DOCUMENTATION CONVENTIONS

Description	Represents	Examples
Main Document (Arial font):		
Italic characters	Referenced books	<i>MPLAB IDE User's Guide</i>
	Emphasized text	...is the <i>only</i> compiler...
Interface References (Arial font):		
Initial caps	A window, dialog or menu selection	Configuration Bits window, Settings dialog, Enable Programmer
Quotes	A field name in a window or dialog	"Save files before running the debugger"
Underlined, italic text with right arrow	A menu selection path	<u>File>Save</u>
Bold characters	A dialog button or tab	OK button, Power tab
Characters in angle brackets < >	A key on the keyboard	<Tab>, <Ctrl-C>
Code References (Courier font):		
Plain characters	File names and paths	c:\autoexec.bat
	Bit values	0, 1
	Sample code	#define START
Square brackets []	Optional arguments	mpasmwin [main.asm]
Curly brackets and pipe character: { }	Choice of mutually exclusive arguments An OR selection	errorlevel {0 1}
Italic characters	A variable argument; it can be either a type of data (in lower case characters) or a specific example (in uppercase characters).	pic30-gcc <i>filename</i>
Ellipses...	Replaces repeated instances of text	list ["list_option...", "list_option"]
0xnnnn	A hexadecimal number where <i>n</i> is a hexadecimal digit	0xFFFF, 0x007A, 0x1A
'bnnnn	A binary number where <i>n</i> is a digit	'b00100, 'b10

Documentation Updates

All documentation becomes dated, and this document is no exception. Since Microchip language and other tools are constantly evolving to meet customer needs, some actual tool descriptions may differ from those in this document. Please refer to our web site to obtain the latest documentation available.

Documentation Numbering Conventions

Documents are numbered with a "DS" number. The 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 = The document number.
A = The revision level of the document.

RECOMMENDED READING

This document describes dsPIC library functions and macros. For more information on dsPIC 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.

Getting Started with dsPIC Language Tools (DS51316)

A guide to installing and working with the Microchip language tools (MPLAB ASM30, MPLAB LINK30 and MPLAB C30) for dsPIC digital signal controllers (DSC's). Examples using the dsPIC simulator, MPLAB SIM30, are provided.

MPLAB ASM30, MPLAB LINK30 and Utilities User's Guide (DS51317)

A guide to using the dsPIC DSC assembler, MPLAB ASM30, dsPIC DSC linker, MPLAB LINK30 and various dsPIC DSC utilities, including MPLAB LIB30 archiver/librarian.

MPLAB C30 C Compiler User's Guide (DS51284)

A guide to using the dsPIC DSC C compiler. MPLAB LINK30 is used with this tool.

GNU HTML documentation

This documentation is provided on the language tool CD-ROM. It describes the standard GNU development tools, upon which these tools is based.

dsPIC High Performance Digital Signal Controllers (DS70032)

Data sheet for dsPIC30F digital sign controller (DSC). Gives an overview of the device and its architecture. Details memory organization, DSP operation and peripheral functionality. Includes electrical characteristics.

dsPIC30F Programmer's Reference Manual (DS70030)

Programmer's guide to dsPIC30F devices. Includes the programmer's model and instruction set.

Technical Library CD-ROM (DS00161)

This CD-ROM contains comprehensive application notes, data sheets, and technical briefs for all Microchip products. To obtain this CD-ROM, contact the nearest Microchip Sales and Service location (see back page).

Microchip Web Site

Our 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 on the Microchip World Wide Web (WWW) site. The website is used by Microchip as a means to make files and information easily available to customers. To view the site, you must have access to the Internet and a web browser such as Netscape Navigator or Microsoft Internet Explorer.

The Microchip web site is available by using your favorite Internet browser to attach to:

<http://www.microchip.com>

The web site provides a variety of services. Users may download files for the latest development tools, data sheets, application notes, user's guides, articles, and sample programs. A variety of information specific to the business of Microchip is also available, including listings of Microchip sales offices, distributors and factory representatives.

Technical Support

- Frequently Asked Questions (FAQ)
- Online Discussion Groups - Conferences for products, Development Systems, technical information and more
- Microchip Consultant Program Member Listing
- Links to other useful web sites related to Microchip products

Developer's Toolbox

- Design Tips
- Device Errata

Other available information

- Latest Microchip Press Releases
- Listing of seminars and events
- Job Postings

DEVELOPMENT SYSTEMS CUSTOMER NOTIFICATION SERVICE

Microchip started the customer notification service to help our customers keep current on Microchip products with the least amount of effort. Once you subscribe, you will receive email notification whenever we change, update, revise or have errata related to your specified product family or development tool of interest.

Go to the Microchip WWW web page (<http://www.microchip.com>) and click on Customer Change Notification under Items of Interest. Follow the instructions to register.

The Development Systems product group categories are:

- Compilers
- Emulators
- In-Circuit Debuggers
- MPLAB
- Programmers

Here is a description of these categories:

COMPILERS - The latest information on Microchip C compilers and other language tools. These include the MPLAB C17, 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 Microchip in-circuit debuggers. These include the MPLAB ICD and MPLAB ICD 2.

MPLAB - 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 device programmers. These include the PRO MATE II device programmer and 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)
- Corporate Applications Engineer (CAE)
- Hotline

Customers should call their distributor, representative or field application engineer (FAE) for support. Local sales offices are also available to help customers. See the back cover for a listing of sales offices and locations.

Corporate Applications Engineers (CAEs) may be contacted at (480) 792-7627.

In addition, there is a Systems Information and Upgrade Line. This line provides system users a listing of the latest versions of all of Microchip's development systems software products. Plus, this line provides information on how customers can receive any currently available upgrade kits.

The Hotline Numbers are:

1-800-755-2345 for U.S. and most of Canada.

1-480-792-7302 for the rest of the world.

NOTES:

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* for more information about making and using libraries.

1.1.1 Assembly Code Applications

Free versions of the dsPIC language tool libraries are available from the Microchip web site. DSP and dsPIC 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 dsPIC language tool libraries are included in the `c:\pic30_tools\lib` directory, where `c:\pic30_tools` is the MPLAB C30 C compiler install directory. These can be linked directly into an application with MPLAB LINK30.

1.2 HIGHLIGHTS

This chapter is organized as follows:

- Startup Code
- DSP Library
- dsPIC Peripheral Libraries
- Standard C Libraries (with Math Functions)

1.3 STARTUP CODE

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

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

For more on startup code, see the *MPLAB ASM30, MPLAB LINK30 and Utilities User's Guide* and, for C applications, the *MPLAB C30 C Compiler User's Guide*.

1.4 DSP LIBRARY

The DSP library (`libdsp.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 dsPIC PERIPHERAL LIBRARIES

The dsPIC (software and hardware) peripheral libraries provide functions and macros for setting up and controlling dsPIC30F DSC peripherals. Examples of use are also provided in each related chapter of this book.

These libraries are processor-specific and of the form `libpDeviceRev.a`, where *Device* = dsPIC device number (e.g., `libp30F6014.a` for the dsPIC30F6014 device) and *Rev* = revision of dsPIC device silicon (if appropriate.)

1.6 STANDARD C LIBRARIES (WITH MATH FUNCTIONS)

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

Additionally, some dsPIC standard C library helper functions, and standard functions that must be modified for use with dsPIC devices, are in `libpic30.a`.

A typical C application will require all three libraries: `libpic30.a`, `libc.a`, and `libm.a`.

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 digital signal controller (DSC). The library has been designed to provide you, the C software developer, with efficient implementation of the most common signal processing functions. In total, 49 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:\pic30_tools`) contains the following subdirectories with library-related files:

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

2.2 HIGHLIGHTS

This chapter is organized as follows:

- Using the DSP Library
- Vector Functions
- Window Functions
- Matrix Functions
- Filtering Functions
- Transform Functions

2.3 USING THE DSP LIBRARY

2.3.1 Building with the DSP Library

Building an application which utilizes the DSP Library requires only two files: `dsp.h` and `libdsp.a`. `dsp.h` is a header file which provides all the function prototypes, `#defines` and `typedefs` used by the library. `libdsp.a` is the archived library file which contains all the individual object files for each library function.

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.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.

2.3.2 Memory Models

The DSP Library is built with the “small code” and “small data” memory models to create the smallest library possible. Since several of the 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.

2.3.3 DSP Library Function Calling Convention

All the object modules within the DSP Library are compliant with the C compatibility guidelines for the dsPIC30F DSC and follow the function call conventions documented in the Microchip *MPLAB C30 C Compiler User’s Guide*. 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.

2.3.4 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 DSC. In this sense, most operations are computed using fractional arithmetic.

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

```
#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.

2.3.5 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.

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 computationally 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 dsPIC30F architecture.

2.3.6 CORCON Register Usage

Many functions of the DSP Library place the dsPIC30F 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 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*.

2.3.7 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 and OVBTE 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.

2.3.8 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 dsPIC30F DSC, 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.

2.3.9 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:\pic30_tools`. 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.

2.4 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.

2.4.1 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 n-th 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 dsPIC30F 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.

2.4.2 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 vectors 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:

a = Op1 (b, c), with b = Op2 (d), and c = Op3 (e, f), then

a = Op1 (Op2 (d), Op3 (e, f))

2.4.3 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+numElems-1`, with `numElems` elements from a source vector located between elements `M` and `M+numElems-1`. Then, the `VectorCopy` function could be used as follows:

```
fractional* dstV[DST_ELEMS] = {...};
fractional* srcV[SRC_ELEMS] = {...};
int n = NUM_ELEMS;
int N = N_PLACE;    /* NUM_ELEMS+N ≤ DST_ELEMS */
int M = M_PLACE;    /* NUM_ELEMS+M ≤ SRC_ELEMS */
fractional* dstVector = dstV+N;
fractional* srcVector = srcV+M;

dstVector = VectorCopy (n, dstVector, srcVector);
```

Also in this context, the `VectorZeroPad` function can operate in place, where now `dstV = srcV`, `numElems` is the number of elements at the beginning of source vector to preserve, and `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.

2.4.4 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:

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

Arguments:

<i>numElems</i>	number of elements in source vectors
<i>dstV</i>	pointer to destination vector
<i>srcV1</i>	pointer to source one vector
<i>srcV2</i>	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 $1 \cdot 2^{-15}$, 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.asm`

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):
 $17 + 3(\text{numElems})$

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:

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

Arguments:

<i>numElems1</i>	number of elements in source one vector
<i>numElems2</i>	number of elements in source two vector
<i>dstV</i>	pointer to destination vector
<i>srcV1</i>	pointer to source one vector
<i>srcV2</i>	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.asm

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):
	58
	Cycles (including C-function call and return overheads):
	For $N = numElems1$, and $M = numElems2$,
	$28 + 13M + 6 \sum_{m=1}^M m + (N - M)(7 + 3M), \text{ for } M < N$
	$28 + 13M + 6 \sum_{m=1}^M m, \text{ for } M = N$

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:	extern fractional* VectorCopy (int numElems, fractional* dstV, 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 <i>must</i> already exist. Destination vectors <i>must</i> 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.asm

VectorCopy (Continued)

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* ≤ *N*.)

Include: *dsp.h*

Prototype:

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

Arguments:

<i>numElems1</i>	number of elements in source one vector
<i>numElems2</i>	number of elements in source two vector
<i>dstV</i>	pointer to destination vector
<i>srcV1</i>	pointer to source one vector
<i>srcV2</i>	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.
This function uses *VectorConvolve*.

Source File: *vcor.asm*

VectorCorrelate (Continued)

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:

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

Arguments:

<i>numElems</i>	number of elements in source vectors
<i>srcV1</i>	pointer to source one vector
<i>srcV2</i>	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.asm

VectorDotProduct (Continued)

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:

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

Arguments:

<i>numElems</i>	number of elements in source vector
<i>srcV</i>	pointer to source vector
<i>maxIndex</i>	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.asm

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 $srcV[n] \leq srcV[n+1], 0 \leq n < numElems-1$

$19 + 7(numElems-2)$

if $srcV[n] > srcV[n+1], 0 \leq n < numElems-1$

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.asm	
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:	<pre>extern fractional* VectorMultiply (int numElems, fractional* dstV, fractional* srcV1, fractional* srcV2);</pre>								
Arguments:	<table> <tr> <td><i>numElems</i></td><td>number of elements in source vector</td></tr> <tr> <td><i>dstV</i></td><td>pointer to destination vector</td></tr> <tr> <td><i>srcV1</i></td><td>pointer to source one vector</td></tr> <tr> <td><i>srcV2</i></td><td>pointer to source two vector</td></tr> </table>	<i>numElems</i>	number of elements in source vector	<i>dstV</i>	pointer to destination vector	<i>srcV1</i>	pointer to source one vector	<i>srcV2</i>	pointer to source two vector
<i>numElems</i>	number of elements in source vector								
<i>dstV</i>	pointer to destination vector								
<i>srcV1</i>	pointer to source one vector								
<i>srcV2</i>	pointer to source two vector								
Return Value:	Pointer to base address of destination vector.								

VectorMultiply (Continued)

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:	<code>vmul.asm</code>
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:	<code>dsp.h</code>						
Prototype:	<pre>extern fractional* VectorNeg (int numElems, fractional* dstV, fractional* srcV);</pre>						
Arguments:	<table><tr><td><i>numElems</i></td><td>number of elements in source vector</td></tr><tr><td><i>dstV</i></td><td>pointer to destination vector</td></tr><tr><td><i>srcV</i></td><td>pointer to source vector</td></tr></table>	<i>numElems</i>	number of elements in source vector	<i>dstV</i>	pointer to destination vector	<i>srcV</i>	pointer to source vector
<i>numElems</i>	number of elements in source vector						
<i>dstV</i>	pointer to destination vector						
<i>srcV</i>	pointer to source vector						
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:	<code>vneg.asm</code>						
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): $19 + 4(numElems)$						

VectorPower

Description:	VectorPower computes the power of a source vector as the sum of the squares of its elements.								
Include:	dsp.h								
Prototype:	<pre>extern fractional VectorPower (int numElems, fractional* srcV);</pre>								
Arguments:	<table> <tr> <td><i>numElems</i></td><td>number of elements in source vector</td></tr> <tr> <td><i>srcV</i></td><td>pointer to source vector</td></tr> </table>	<i>numElems</i>	number of elements in source vector	<i>srcV</i>	pointer to source vector				
<i>numElems</i>	number of elements in source vector								
<i>srcV</i>	pointer to source vector								
Return Value:	Value of the vector's power (sum of squares).								
Remarks:	<p>If the absolute value of the sum of squares is larger than $1-2^{-15}$, this operation results in saturation</p> <p>This function can be self applicable.</p>								
Source File:	vpow.asm								
Function Profile:	<p>System resources usage:</p> <table> <tr> <td>W0..W2</td><td>used, not restored</td></tr> <tr> <td>W4</td><td>used, not restored</td></tr> <tr> <td>ACCA</td><td>used, not restored</td></tr> <tr> <td>CORCON</td><td>saved, used, restored</td></tr> </table> <p>DO and REPEAT instruction usage:</p> <p>no DO instructions</p> <p>1 level REPEAT instructions</p> <p>Program words (24-bit instructions):</p> <p>12</p> <p>Cycles (including C-function call and return overheads):</p> <p>$16 + 2(numElems)$</p>	W0..W2	used, not restored	W4	used, not restored	ACCA	used, not restored	CORCON	saved, used, restored
W0..W2	used, not restored								
W4	used, not restored								
ACCA	used, not restored								
CORCON	saved, used, restored								

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:	<pre>extern fractional* VectorScale (int numElems, fractional* dstV, fractional* srcV, fractional sclVal);</pre>								
Arguments:	<table> <tr> <td><i>numElems</i></td><td>number of elements in source vector</td></tr> <tr> <td><i>dstV</i></td><td>pointer to destination vector</td></tr> <tr> <td><i>srcV</i></td><td>pointer to source vector</td></tr> <tr> <td><i>sclVal</i></td><td>value by which to scale vector elements</td></tr> </table>	<i>numElems</i>	number of elements in source vector	<i>dstV</i>	pointer to destination vector	<i>srcV</i>	pointer to source vector	<i>sclVal</i>	value by which to scale vector elements
<i>numElems</i>	number of elements in source vector								
<i>dstV</i>	pointer to destination vector								
<i>srcV</i>	pointer to source vector								
<i>sclVal</i>	value by which to scale vector elements								
Return Value:	Pointer to base address of destination vector.								
Remarks:	<p><i>sclVal</i> must be a fractional number in 1.15 format.</p> <p>This function can be computed in place.</p>								
Source File:	vscl.asm								

VectorScale (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):

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:

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

Arguments:

<i>numElems</i>	number of elements in source vectors
<i>dstV</i>	pointer to destination vector
<i>srcV1</i>	pointer to source one vector (minuend)
<i>srcV2</i>	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 $1 \cdot 2^{-15}$, 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: vsub.asm

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):

14

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

$17 + 4(numElems)$

VectorZeroPad

Description:	<p>VectorZeroPad copies the source vector into the beginning of the (already existing) destination vector, and then fills with zeros the remaining <code>numZeros</code> elements of destination vector:</p> $dstV[n] = srcV[n], 0 \leq n < numElems$ $dstV[n] = 0, numElems \leq n < numElems + numZeros$																		
Include:	<code>dsp.h</code>																		
Prototype:	<pre>extern fractional* VectorZeroPad (int numElems, int numZeros, fractional* dstV, fractional* srcV);</pre>																		
Arguments:	<table> <tr> <td><code>numElems</code></td><td>number of elements in source vector</td></tr> <tr> <td><code>numZeros</code></td><td>number of elements to fill with zeros at the tail of destination vector</td></tr> <tr> <td><code>dstV</code></td><td>pointer to destination vector</td></tr> <tr> <td><code>srcV</code></td><td>pointer to source vector</td></tr> </table>	<code>numElems</code>	number of elements in source vector	<code>numZeros</code>	number of elements to fill with zeros at the tail of destination vector	<code>dstV</code>	pointer to destination vector	<code>srcV</code>	pointer to source vector										
<code>numElems</code>	number of elements in source vector																		
<code>numZeros</code>	number of elements to fill with zeros at the tail of destination vector																		
<code>dstV</code>	pointer to destination vector																		
<code>srcV</code>	pointer to source vector																		
Return Value:	Pointer to base address of destination vector.																		
Remarks:	<p>The destination vector <i>must</i> already exist, with exactly <code>numElems+numZeros</code> number of elements.</p> <p>This function can be computed in place. See Additional Remarks at the beginning of the section for comments on this mode of operation.</p> <p>This function uses <code>VectorCopy</code>.</p>																		
Source File:	<code>vzpad.asm</code>																		
Function Profile:	<p>System resources usage:</p> <table> <tr> <td>W0..W6</td><td>used, not restored</td></tr> <tr> <td colspan="2">plus resources from <code>VectorCopy</code></td></tr> </table> <p>DO and REPEAT instruction usage:</p> <table> <tr> <td colspan="2">no DO instructions</td></tr> <tr> <td colspan="2">1 level REPEAT instructions</td></tr> <tr> <td colspan="2">plus DO/REPEAT from <code>VectorCopy</code></td></tr> </table> <p>Program words (24-bit instructions):</p> <table> <tr> <td>13,</td><td></td></tr> <tr> <td colspan="2">plus program words from <code>VectorCopy</code></td></tr> </table> <p>Cycles (including C-function call and return overheads):</p> <table> <tr> <td>18 + <code>numZeros</code></td><td></td></tr> <tr> <td colspan="2">plus cycles from <code>VectorCopy</code>.</td></tr> </table> <p>Note: In the description of <code>VectorCopy</code>, the number of cycles reported includes 3 cycles of C-function call overhead. Thus, the number of actual cycles from <code>VectorCopy</code> to add to <code>VectorCorrelate</code> is 3 less than whatever number is reported for a stand alone <code>VectorCopy</code>.</p>	W0..W6	used, not restored	plus resources from <code>VectorCopy</code>		no DO instructions		1 level REPEAT instructions		plus DO/REPEAT from <code>VectorCopy</code>		13,		plus program words from <code>VectorCopy</code>		18 + <code>numZeros</code>		plus cycles from <code>VectorCopy</code> .	
W0..W6	used, not restored																		
plus resources from <code>VectorCopy</code>																			
no DO instructions																			
1 level REPEAT instructions																			
plus DO/REPEAT from <code>VectorCopy</code>																			
13,																			
plus program words from <code>VectorCopy</code>																			
18 + <code>numZeros</code>																			
plus cycles from <code>VectorCopy</code> .																			

2.5 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.

2.5.1 User Considerations

- All the window initialization functions have been designed to generate window vectors allocated in default RAM memory space (X-Data or Y-Data).
- The windowing function is designed to operate on vectors allocated in default RAM memory space (X-Data or Y-Data).
- It is recommended that the Status register (SR) be examined after completion of each function call.
- Since the window initialization functions are implemented in C, consult the electronic documentation included in the release for up-to-date cycle count information.

2.5.2 Individual Functions

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

BartlettInit

Description:	BartlettInit initializes a Bartlett window of length <i>numElems</i> .				
Include:	dsp.h				
Prototype:	<pre>extern fractional* BartlettInit (int numElems, fractional* window);</pre>				
Arguments:	<table><tr><td><i>numElems</i></td><td>number of elements in window</td></tr><tr><td><i>window</i></td><td>pointer to window to be initialized</td></tr></table>	<i>numElems</i>	number of elements in window	<i>window</i>	pointer to window to be initialized
<i>numElems</i>	number of elements in window				
<i>window</i>	pointer to window to be initialized				
Return Value:	Pointer to base address of initialized window.				
Remarks:	The <i>window</i> vector <i>must</i> already exist, with exactly <i>numElems</i> number of elements.				
Source File:	initbart.c				

BartlettInit (Continued)

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.

BlackmanInit

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

Include: dsp.h

Prototype:

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

Arguments:

<i>numElems</i>	number of elements in window
<i>window</i>	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: initblk.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
);
```

HammingInit (Continued)

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 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.

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 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.

KaiserInit

Description:	KaiserInit initializes a Kaiser window with shape determined by argument <i>betaVal</i> and of length <i>numElems</i> .						
Include:	dsp.h						
Prototype:	<pre>extern fractional* KaiserInit (int numElems, fractional* window, float betaVal);</pre>						
Arguments:	<table> <tr> <td><i>numElems</i></td><td>number of elements in window</td></tr> <tr> <td><i>window</i></td><td>pointer to window to be initialized</td></tr> <tr> <td><i>betaVal</i></td><td>window shaping parameter</td></tr> </table>	<i>numElems</i>	number of elements in window	<i>window</i>	pointer to window to be initialized	<i>betaVal</i>	window shaping parameter
<i>numElems</i>	number of elements in window						
<i>window</i>	pointer to window to be initialized						
<i>betaVal</i>	window shaping parameter						
Return Value:	Pointer to base address of initialized window.						
Remarks:	The window vector <i>must</i> already exist, with exactly <i>numElems</i> number of elements.						
Source File:	initkais.c						
Function Profile:	<p>System resources usage:</p> <table> <tr> <td>W0..W7</td><td>used, not restored</td></tr> <tr> <td>W8..W14</td><td>saved, used, not restored</td></tr> </table> <p>DO and REPEAT instruction usage:</p> <p>None</p> <p>Program words (24-bit instructions):</p> <p>See the file "Readme for dsPIC Language Tools Libraries.txt" for this information.</p> <p>Cycles (including C-function call and return overheads):</p> <p>See the file "Readme for dsPIC Language Tools Libraries.txt" for this information.</p>	W0..W7	used, not restored	W8..W14	saved, used, not restored		
W0..W7	used, not restored						
W8..W14	saved, used, not restored						

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:	<pre>extern fractional* VectorWindow (int numElems, fractional* dstV, fractional* srcV, fractional* window);</pre>								
Arguments:	<table> <tr> <td><i>numElems</i></td><td>number of elements in source vector</td></tr> <tr> <td><i>dstV</i></td><td>pointer to destination vector</td></tr> <tr> <td><i>srcV</i></td><td>pointer to source vector</td></tr> <tr> <td><i>window</i></td><td>pointer to initialized window</td></tr> </table>	<i>numElems</i>	number of elements in source vector	<i>dstV</i>	pointer to destination vector	<i>srcV</i>	pointer to source vector	<i>window</i>	pointer to initialized window
<i>numElems</i>	number of elements in source vector								
<i>dstV</i>	pointer to destination vector								
<i>srcV</i>	pointer to source vector								
<i>window</i>	pointer to initialized window								
Return Value:	Pointer to base address of destination vector.								
Remarks:	<p>The window vector <i>must</i> have already been initialized, with exactly <i>numElems</i> number of elements.</p> <p>This function can be computed in place.</p> <p>This function can be self applicable.</p> <p>This function uses VectorMultiply.</p>								

VectorWindow (Continued)

Source File: dowindow.asm

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.

2.6 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.

2.6.1 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:

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

Note that the factor of 2 is used because of the byte addressing capabilities of the dsPIC30F.

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(es). 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.

2.6.2 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:

a = Op1 (b, c), with b = Op2 (d), and c = Op3 (e, f), then

a = Op1 (Op2 (d), Op3 (e, f))

2.6.3 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 not 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.

2.6.4 Individual Functions

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

MatrixAdd

Description: `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.

Include: `dsp.h`

Prototype:

```
extern fractional* MatrixAdd (
    int numRows,
    int numCols,
    fractional* dstM,
    fractional* srcM1,
    fractional* srcM2
);
```

Arguments:

<code>numRows</code>	number of rows in source matrices
<code>numCols</code>	number of columns in source matrices
<code>dstM</code>	pointer to destination matrix
<code>srcM1</code>	pointer to source one matrix
<code>srcM2</code>	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 $1 \cdot 2^{-15}$, 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.asm`

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):

$20 + 3(numRows * numCols)$

MatrixMultiply

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

$$dstM[i][j] = \sum_k (srcM1[i][k])(srcM2[k][j])$$

where:

$$0 \leq i < numRows1$$

$$0 \leq j < numCols2$$

$$0 \leq k < numCols1Rows2$$

Include: dsp.h

Prototype:

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

Arguments:

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

Return Value: Pointer to base address of destination matrix.

Remarks: If the absolute value of

$$\sum_k (srcM1[i][k])(srcM2[k][j])$$

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.asm

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):

$$36 + numRows1 * (8 + numCols2 * (7 + 4 * numCols1Rows2))$$

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:	<pre>extern fractional* MatrixScale (int numRows, int numCols, fractional* dstM, fractional* srcM, fractional sclVal);</pre>										
Arguments:	<table><tr><td><i>numRows</i></td><td>number of rows in source matrix</td></tr><tr><td><i>numCols</i></td><td>number of columns in source matrix</td></tr><tr><td><i>dstM</i></td><td>pointer to destination matrix</td></tr><tr><td><i>srcM</i></td><td>pointer to source matrix</td></tr><tr><td><i>sclVal</i></td><td>value by which to scale matrix elements</td></tr></table>	<i>numRows</i>	number of rows in source matrix	<i>numCols</i>	number of columns in source matrix	<i>dstM</i>	pointer to destination matrix	<i>srcM</i>	pointer to source matrix	<i>sclVal</i>	value by which to scale matrix elements
<i>numRows</i>	number of rows in source matrix										
<i>numCols</i>	number of columns in source matrix										
<i>dstM</i>	pointer to destination matrix										
<i>srcM</i>	pointer to source matrix										
<i>sclVal</i>	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.asm										
Function Profile:	<p>System resources usage:</p> <table><tr><td>W0..W5</td><td>used, not restored</td></tr><tr><td>ACCA</td><td>used, not restored</td></tr><tr><td>CORCON</td><td>saved, used, restored</td></tr></table> <p>DO and REPEAT instruction usage:</p> <table><tr><td>1 level DO instructions</td></tr><tr><td>no REPEAT instructions</td></tr></table> <p>Program words (24-bit instructions):</p> <table><tr><td>14</td></tr></table> <p>Cycles (including C-function call and return overheads):</p> <table><tr><td>$20 + 3(numRows * numCols)$</td></tr></table>	W0..W5	used, not restored	ACCA	used, not restored	CORCON	saved, used, restored	1 level DO instructions	no REPEAT instructions	14	$20 + 3(numRows * numCols)$
W0..W5	used, not restored										
ACCA	used, not restored										
CORCON	saved, used, restored										
1 level DO instructions											
no REPEAT instructions											
14											
$20 + 3(numRows * numCols)$											

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:	<pre>extern fractional* MatrixSubtract (int numRows, int numCols, fractional* dstM, fractional* srcM1, fractional* srcM2);</pre>

MatrixSubtract (Continued)

Arguments:	<i>numRows</i> number of rows in source matrix(ces) <i>numCols</i> number of columns in source matrix(ces) <i>dstM</i> pointer to destination matrix <i>srcM1</i> pointer to source one matrix (minuend) <i>srcM2</i> pointer to source two matrix (subtrahend)								
Return Value:	Pointer to base address of destination matrix.								
Remarks:	<p>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.</p> <p>This function can be computed in place.</p> <p>This function can be self applicable.</p>								
Source File:	msub.asm								
Function Profile:	<p>System resources usage:</p> <table> <tr><td>W0..W4</td><td>used, not restored</td></tr> <tr><td>ACCA</td><td>used, not restored</td></tr> <tr><td>ACCB</td><td>used, not restored</td></tr> <tr><td>CORCON</td><td>saved, used, restored</td></tr> </table> <p>DO and REPEAT instruction usage:</p> <p>1 level DO instructions</p> <p>no REPEAT instructions</p> <p>Program words (24-bit instructions):</p> <p>15</p> <p>Cycles (including C-function call and return overheads):</p> <p>$20 + 4(numRows * numCols)$</p>	W0..W4	used, not restored	ACCA	used, not restored	ACCB	used, not restored	CORCON	saved, used, restored
W0..W4	used, not restored								
ACCA	used, not restored								
ACCB	used, not restored								
CORCON	saved, used, restored								

MatrixTranspose

Description:	<p>MatrixTranspose transposes the rows by the columns in the source matrix, and places the result in destination matrix. In effect:</p> $dstM[i][j] = srcM[j][i],$ $0 \leq i < numRows, 0 \leq j < numCols.$
Include:	dsp.h
Prototype:	<pre>extern fractional* MatrixTranspose (int numRows, int numCols, fractional* dstM, fractional* srcM);</pre>
Arguments:	<i>numRows</i> number of rows in source matrix <i>numCols</i> number of columns in source matrix <i>dstM</i> pointer to destination matrix <i>srcM</i> pointer to source matrix
Return Value:	Pointer to base address of destination matrix.
Remarks:	<p>If the source matrix is square, this function can be computed in place.</p> <p>See Additional Remarks at the beginning of the section for comments on this mode of operation.</p>
Source File:	mtrp.asm

MatrixTranspose (Continued)

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):
	$16 + numCols * (6 + (numRows - 1) * 3)$

2.6.5 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 $\{-1, \dots, 1-2^{-15}\}$. 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:	<pre>extern float* MatrixInvert (int numRowsCols, float* dstM, float* srcM, float* pivotFlag, int* swappedRows, int* swappedCols);</pre>	
Arguments:	<i>numRowCols</i>	number of rows and columns in (square) source matrix
	<i>dstM</i>	pointer to destination matrix
	<i>srcM</i>	pointer to source matrix
	Required for internal use:	
	<i>pivotFlag</i>	pointer to a length numRowsCols vector
	<i>swappedRows</i>	pointer to a length numRowsCols vector
	<i>swappedCols</i>	pointer to a length numRowsCols vector
Return Value:	Pointer to base address of destination matrix, or NULL if source matrix is singular.	
Remarks:	<p>Even though the vectors <i>pivotFlag</i>, <i>swappedRows</i>, and <i>swappedCols</i>, are for internal use only, they must be allocated prior to calling this function.</p> <p>If source matrix is singular (determinant equal to zero) the matrix does not have an inverse. In this case the function returns NULL.</p> <p>This function can be computed in place.</p>	
Source File:	minv.asm (assembled from C-code)	

MatrixInvert (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 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.	

2.7 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.

2.7.1 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.

2.7.2 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 $\{1, \dots, M\}$. 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 constraints imposed by fractional arithmetic.

All these considerations lead to a proliferation of filtering operations, of which a subset is provided by the DSP Library.

2.7.3 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.

2.7.4 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 if:

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

$a = \text{Op1}(\text{Op2}(d), \text{Op3}(e, f))$

2.7.5 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 Schaffer's *Discrete-Time Signal Processing*, Prentice Hall, 1989. For implementation details of Least Mean Square FIR filters, please refer to T. Hsia's *Convergence 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:

<i>numCoeffs</i>	number of coefficients in filter (also M)
<i>coeffsBase</i>	base address for filter coefficients (also h)
<i>coeffsEnd</i>	end address for filter coefficients
<i>coeffsPage</i>	coefficients buffer page number
<i>delayBase</i>	base address for delay buffer
<i>delayEnd</i>	end address for delay buffer
<i>delay</i>	current value of delay pointer (also d)

Remarks: Number of coefficients in filter is M.

Coefficients, $h[m]$, defined in $0 \leq m < M$, either within X-Data or program memory.

Delay buffer $d[m]$, defined in $0 \leq m < M$, *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 coefficients. This latter value can be calculated using the inline assembly operator `psvpage()`.

delayBase points to the actual address where the delay buffer is allocated.

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.

FIRStruct (Continued)

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:

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

Arguments:

<i>numSamps</i>	number of input samples to filter (also N)
<i>dstSamps</i>	pointer to destination samples (also y)
<i>srcSamps</i>	pointer to source samples (also x)
<i>filter</i>	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.asm`

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
XMODSTR	saved, used, restored
XMODEND	saved, used, restored
YMODSTR	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.

FIRDecimate

Description:	<p>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,</p> $y[n] = x[Rn].$ <p>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.</p>										
Include:	dsp.h										
Prototype:	<pre>extern fractional* FIRDecimate (int numSamps, fractional* dstSamps, fractional* srcSamps, FIRStruct* filter, int rate);</pre>										
Arguments:	<table> <tr> <td><i>numSamps</i></td><td>number of <i>output</i> samples (also N, $N = Rp$, p integer)</td></tr> <tr> <td><i>dstSamp</i></td><td>pointer to destination samples (also y)</td></tr> <tr> <td><i>srcSamps</i></td><td>pointer to source samples (also x)</td></tr> <tr> <td><i>filter</i></td><td>pointer to FIRStruct filter structure</td></tr> <tr> <td><i>rate</i></td><td>rate of decimation (downsampling factor, also R)</td></tr> </table>	<i>numSamps</i>	number of <i>output</i> samples (also N, $N = Rp$, p integer)	<i>dstSamp</i>	pointer to destination samples (also y)	<i>srcSamps</i>	pointer to source samples (also x)	<i>filter</i>	pointer to FIRStruct filter structure	<i>rate</i>	rate of decimation (downsampling factor, also R)
<i>numSamps</i>	number of <i>output</i> samples (also N, $N = Rp$, p integer)										
<i>dstSamp</i>	pointer to destination samples (also y)										
<i>srcSamps</i>	pointer to source samples (also x)										
<i>filter</i>	pointer to FIRStruct filter structure										
<i>rate</i>	rate of decimation (downsampling factor, also R)										
Return Value:	Pointer to base address of destination samples.										
Remarks:	<p>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.</p> <p>Delay, $d[m]$, defined in $0 \leq m < M$, not implemented as a circular modulo buffer.</p> <p>Source samples, $x[n]$, defined in $0 \leq n < NR$.</p> <p>Destination samples, $y[n]$, defined in $0 \leq n < N$.</p> <p>(See also FIRStruct, FIRStructInit, and FIRDelayInit.)</p>										
Source File:	firdecim.asm										
Function Profile:	<p>System resources usage:</p> <table> <tr> <td>W0..W7</td><td>used, not restored</td></tr> <tr> <td>W8..W12</td><td>saved, used, restored</td></tr> <tr> <td>ACCA</td><td>used, not restored</td></tr> <tr> <td>CORCON</td><td>saved, used, restored</td></tr> <tr> <td>PSVPAG</td><td>saved, used, restored (only if coefficients in P memory)</td></tr> </table> <p>DO and REPEAT instruction usage:</p> <ul style="list-style-type: none"> 1 level DO instructions 1 level REPEAT instructions <p>Program words (24-bit instructions):</p> <p>48</p> <p>Cycles (including C-function call and return overheads):</p> <p>45 + $N(10 + 2M)$, or</p> <p>48 + $N(13 + 2M)$ if coefficients in P memory.</p>	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)
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)										

FIRDelayInit

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

FIRDelayInit (Continued)

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.asm`

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

Arguments: *numSamps* number of input samples (also N, $N = R \cdot p$, 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.

FIRInterpolate (Continued)

Remarks:	<p>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 <code>FIRStruct</code>, <code>FIRStructInit</code>, and <code>FIRInterpDelayInit</code>.)</p>										
Source File:	<code>firinter.asm</code>										
Function Profile:	<p>System resources usage:</p> <table> <tr> <td>W0..W7</td><td>used, not restored</td></tr> <tr> <td>W8..W13</td><td>saved, used, restored</td></tr> <tr> <td>ACCA</td><td>used, not restored</td></tr> <tr> <td>CORCON</td><td>saved, used, restored</td></tr> <tr> <td>PSVPAG</td><td>saved, used, restored (only if coefficients in P memory)</td></tr> </table> <p>DO and REPEAT instruction usage:</p> <p>2 level DO instructions 1 level REPEAT instructions</p> <p>Program words (24-bit instructions): 63</p> <p>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.</p>	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)
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)										

FIRInterpDelayInit

Description:	<code>FIRInterpDelayInit</code> initializes to zero the delay values in an <code>FIRStruct</code> filter structure, optimized for use with an FIR interpolating filter.				
Include:	<code>dsp.h</code>				
Prototype:	<pre>extern void FIRDelayInit (FIRStruct* filter, int rate);</pre>				
Arguments:	<table> <tr> <td><i>filter</i></td><td>pointer to <code>FIRStruct</code> filter structure</td></tr> <tr> <td><i>rate</i></td><td>rate of interpolation (upsampling factor, also R)</td></tr> </table>	<i>filter</i>	pointer to <code>FIRStruct</code> filter structure	<i>rate</i>	rate of interpolation (upsampling factor, also R)
<i>filter</i>	pointer to <code>FIRStruct</code> filter structure				
<i>rate</i>	rate of interpolation (upsampling factor, also R)				
Remarks:	<p>Delay, $d[m]$, defined in $0 \leq m < M/R$, with M the number of filter coefficients in the interpolator. See description of <code>FIRStruct</code> structure above.</p>				
Source File:	<code>firintdl.asm</code>				

FIRInterpDelayInit (Continued)

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.asm*

FIRLattice (Continued)

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
	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):
	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:	<pre>extern fractional* FIRLMS (int numSamps, fractional* dstSamps, fractional* srcSamps, FIRStruct* filter, fractional* refSamps, fractional muVal);</pre>	
Arguments:	<i>numSamps</i>	number of input samples (also N)
	<i>dstSamps</i>	pointer to destination samples (also y)
	<i>srcSamps</i>	pointer to source samples (also x)
	<i>filter</i>	pointer to FIRStruct filter structure
	<i>refSamps</i>	pointer to reference samples (also r)
	<i>muVal</i>	adapting factor (also mu)
Return Value:	Pointer to base address of destination samples.	

FIRLMS (Continued)

Remarks:	<p>Number of coefficients in filter is M.</p> <p>Coefficients, $h[m]$, defined in $0 \leq m < M$, implemented as a circular increasing modulo buffer.</p> <p>delay, $d[m]$, defined in $0 \leq m < M-1$, implemented as a circular increasing modulo buffer.</p> <p>Source samples, $x[n]$, defined in $0 \leq n < N$.</p> <p>Reference samples, $r[n]$, defined in $0 \leq n < N$.</p> <p>Destination samples, $y[n]$, defined in $0 \leq n < N$.</p> <p>Adaptation:</p> $h_m[n] = h_m[n-1] + \mu * (r[n] - y[n]) * x[n-m],$ <p>for $0 \leq n < N, 0 \leq m < M$.</p> <p>The operation could result in saturation if the absolute value of $(r[n] - y[n])$ is greater than or equal to one.</p> <p>Filter coefficients <i>must not</i> 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.)</p>																		
Source File:	firlms.asm																		
Function Profile:	<p>System resources usage:</p> <table><tr><td>W0..W7</td><td>used, not restored</td></tr><tr><td>W8..W12</td><td>saved, used, restored</td></tr><tr><td>ACCA</td><td>used, not restored</td></tr><tr><td>ACCB</td><td>used, not restored</td></tr><tr><td>CORCON</td><td>saved, used, restored</td></tr><tr><td>MODCON</td><td>saved, used, restored</td></tr><tr><td>XMODSTR</td><td>saved, used, restored</td></tr><tr><td>XMODEND</td><td>saved, used, restored</td></tr><tr><td>YMODSTR</td><td>saved, used, restored</td></tr></table> <p>DO and REPEAT instruction usage:</p> <p>2 level DO instructions</p> <p>1 level REPEAT instructions</p> <p>Program words (24-bit instructions):</p> <p>76</p> <p>Cycles (including C-function call and return overheads):</p> <p>$61 + N(13 + 5M)$</p>	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	XMODSTR	saved, used, restored	XMODEND	saved, used, restored	YMODSTR	saved, used, restored
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																		
XMODSTR	saved, used, restored																		
XMODEND	saved, used, restored																		
YMODSTR	saved, used, restored																		

FIRLMSNorm

Description:	<p>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.</p> <p>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.</p>
Include:	dsp.h

FIRLMSNorm (Continued)

Prototype:

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

Arguments:

<i>numSamps</i>	number of input samples (also N)
<i>dstSamps</i>	pointer to destination samples (also y)
<i>srcSamps</i>	pointer to source samples (also x)
<i>filter</i>	pointer to FIRStruct filter structure
<i>refSamps</i>	pointer to reference samples (also r)
<i>muVal</i>	adapting factor (also mu)
<i>energyEstimate</i>	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 $0 \leq m < M$, 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 $0 \leq n < N$.
Reference samples, $r[n]$, defined in $0 \leq n < N$.
Destination samples, $y[n]$, defined in $0 \leq n < N$.
Adaptation:
$$h_m[n] = h_m[n-1] + \mu[n] * (r[n] - y[n]) * x[n-m],$$

for $0 \leq n < N$, $0 \leq m < M$,
where $\mu[n] = \mu / (\mu + E[n])$
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:
$$E[n] = (x[n])^2 + (x[n-1])^2 + \dots + (x[n-M+2])^2.$$

Thus, to avoid saturation while computing the estimate, the input sample values should be bound so that

$$\sum_{m=0}^{-M+2} (x[n+m])^2 < 1, \text{ for } 0 \leq n < N.$$

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.asm

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
	XMODSTR	saved, used, restored
	XMODEND	saved, used, restored
	YMODSTR	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 FIR-Struct FIR Filter structure.	
Include:	dsp.h	
Prototype:	extern void FIRStructInit (FIRStruct* <i>filter</i> , int <i>numCoeffs</i> , fractional* <i>coeffsBase</i> , int <i>coeffsPage</i> , fractional* <i>delayBase</i>);	
Arguments:	<i>filter</i>	pointer to FIRStruct filter structure
	<i>numCoeffs</i>	number of coefficients in filter (also M)
	<i>coeffsBase</i>	base address for filter coefficients (also h)
	<i>coeffsPage</i>	coefficient buffer page number
	<i>delayBase</i>	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 delay-Base.	
Source File:	firinit.asm	
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:

```
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:

Filter structure:	
<i>numSectionsLess1</i>	1 less than number of cascaded second order (biquadratic) sections (also S-1)
<i>coeffsBase</i>	pointer to filter coefficients (also {a, b}), either within X-Data or program memory
<i>coeffsPage</i>	coefficients buffer page number, or 0xFF00 (defined value COEFFS_IN_DATA) if coefficients in data space
<i>delayBase</i>	pointer to filter delay (also d), <i>only</i> in Y-Data
<i>initialGain</i>	initial gain value
<i>finalShift</i>	output scaling (shift left)

Filter Description:

<i>numSamps</i>	number of input samples to filter (also N)
<i>dstSamps</i>	pointer to destination samples (also y)
<i>srcSamps</i>	pointer to source samples (also x)
<i>filter</i>	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]}, $0 \leq s < S$. 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]}, $0 \leq s < S$. Source samples, $x[n]$, defined in $0 \leq n < N$. Destination samples, $y[n]$, defined in $0 \leq n < N$. 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.asm

IIRCanonic (Continued)

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: IIRCanonicInit initializes to zero the delay values in an IIR-CanonicStruct 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]},
 $0 \leq s < S$.

Source File: iircan.asm

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

IIRLattice (Continued)

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, $M \leq N$; 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 COEFFS_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, $N \geq M$; 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 $0 \leq m \leq M$.

Ladder coefficients, $g[m]$, defined in $0 \leq m \leq M$ (unless if implementing an all-pole filter).

Delay, $d[m]$, defined in $0 \leq m \leq M$.

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

Destination samples, $y[n]$, defined in $0 \leq n < N$.

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.asm

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 IIRLattice function).

Initialization Description:
filter pointer to IIRLatticeStruct filter structure.

Source File: iirlattd.asm

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:

Filter structure:	
<i>numSectionsLess1</i>	1 less than number of cascaded second order (biquadratic) sections (also S-1)
<i>coeffsBase</i>	pointer to filter coefficients (also {a, b}), either in X-Data or program memory
<i>coeffsPage</i>	coefficient buffer page number, or 0xFF00 (defined value COEFFS_IN_DATA) if coefficients in data space
<i>delayBase1</i>	pointer to filter state 1, with one word of delay per second order section (also d1), <i>only</i> in Y-Data
<i>delayBase2</i>	pointer to filter state 2, with one word of delay per second order section (also d2), <i>only</i> in Y-Data
<i>finalShift</i>	output scaling (shift left)
Filter Description:	
<i>numSamps</i>	number of input samples to filter (also N)
<i>dstSamps</i>	pointer to destination samples (also y)
<i>srcSamps</i>	pointer to source samples (also x)
<i>filter</i>	pointer to IIRTransposedStruct filter structure

Return Value: Pointer to base address of destination samples.

IIRTransposed (Continued)

Remarks:	<p>There are 5 coefficients per second order (biquadratic) section arranged in the ordered set {b0[s], b1[s], a1[s], b2[s], a2[s]}, $0 \leq s < S$. Coefficient values should be generated with dsPICFD filter design package from Momentum Data Systems, Inc, or similar tool.</p> <p>The delay is made up of two independent buffers, each buffer containing one word of filter state per section {d2[s], d1[s]}, $0 \leq s < S$.</p> <p>Source samples, x[n], defined in $0 \leq n < N$.</p> <p>Destination samples, y[n], defined in $0 \leq n < N$.</p> <p>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.</p>												
Source File:	iirtrans.asm												
Function Profile:	<p>System resources usage:</p> <table><tr><td>W0..W7</td><td>used, not restored</td></tr><tr><td>W8..W11</td><td>saved, used, restored</td></tr><tr><td>ACCA</td><td>used, not restored</td></tr><tr><td>ACCB</td><td>used, not restored</td></tr><tr><td>CORCON</td><td>saved, used, restored</td></tr><tr><td>PSVPAG</td><td>saved, used, restored</td></tr></table> <p>DO and REPEAT instruction usage:</p> <p>2 level DO instructions</p> <p>1 level REPEAT instructions</p> <p>Program words (24-bit instructions):</p> <p>48</p> <p>Cycles (including C-function call and return overheads):</p> <p>$35 + N(11 + 11S)$, or</p> <p>$38 + N(9 + 17S)$ if coefficients in P memory.</p> <p>S is number of second order sections.</p>	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
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												

IIRTransposedInit

Description:	IIRTransposedInit initializes to zero the delay values in an IIR-TransposedStruct filter structure.
Include:	dsp.h
Prototype:	<pre>extern void IIRTransposedInit (IIRTransposedStruct* filter);</pre>
Arguments:	<p>Filter structure: (See description of IIRTransposed function).</p> <p>Initialization Description: <i>filter</i> pointer to IIRTransposedStruct filter structure.</p>
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.asm

IIRTransposedInit (Continued)

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.

2.7.6 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

```
function [out, del, forward, backward] = iirlatt (in, kappas, gammas, delay)
## FUNCTION.-
## IIRLATT: IIR Fileter 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.
    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).
    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.
return;

#.....

endfunction
```

2.8 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.

2.8.1 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.

2.8.2 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:

```
#ifdef fractional
#ifndef fractcomplex
typedef struct {
    fractional real;
    fractional imag;
} fractcomplex;
#endif
#endif
```

2.8.3 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:
 $a = \text{Op1}(b, c)$, with $b = \text{Op2}(d)$, and $c = \text{Op3}(e, f)$, then
 $a = \text{Op1}(\text{Op2}(d), \text{Op3}(e, f))$.

2.8.4 Individual Functions

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:

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

Arguments:

<i>log2N</i>	based 2 logarithm of N (number of complex elements in source vector)
<i>srcCV</i>	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.asm

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):

Transform Size	# Complex Elements	# Cycles
32 point	32	245
64 point	64	485
128 point	128	945
256 point	256	1905

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:

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

Include: dsp.h

CosFactorInit (Continued)

Prototype:	extern fractcomplex* CosFactorInit (int log2N, fractcomplex* cosFactors);	
Arguments:	log2N	based 2 logarithm of N (number of complex factors needed by a DCT)
	cosFactors	pointer to complex cosine factors
Return Value:	Pointer to base address of cosine factors.	
Remarks:	<p>N <i>must</i> be an integer power of 2. Only the first N/2 cosine factors are generated. A complex vector of size N/2 <i>must</i> have already been allocated and assigned to cosFactors prior to invoking the function. The complex vector <i>should</i> reside in X-Data memory. Factors are computed in floating point arithmetic and converted to 1.15 complex fractionals.</p>	
Source File:	initcosf.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 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.	

DCT

Description:	DCT computes the Discrete Cosine Transform of a source vector, and stores the results in the destination vector.	
Include:	dsp.h	
Prototype:	<pre>extern fractional* DCT (int log2N, fractional* dstV, fractional* srcV, fractcomplex* cosFactors, fractcomplex* twiddleFactors, int factPage);</pre>	
Arguments:	<i>log2N</i>	based 2 logarithm of N (number of complex elements in source vector)
	<i>dstCV</i>	pointer to destination vector
	<i>srcCV</i>	pointer to source vector
	<i>cosFactors</i>	pointer to cosine factors
	<i>twiddleFactors</i>	pointer to twiddle factors
	<i>factPage</i>	memory page for transform factors
Return Value:	Pointer to base address of destination vector.	

DCT (Continued)

Remarks:	<p><i>N</i> must be an integer power of 2.</p> <p>This function operates out of place. A vector of size 2N elements, <i>must</i> already have been allocated and assigned to <code>dstV</code>.</p> <p>The <code>dstV</code> vector must be allocated at a modulo alignment of N.</p> <p>The results of computation are stored in the first N elements of the destination vector.</p> <p>To avoid saturation (overflow) during computation, the values of the source vector <i>should</i> be in the range [-0.5, 0.5].</p> <p>Only the first N/2 cosine factors are needed.</p> <p>Only the first N/2 twiddle factors are needed.</p> <p>If the transform factors are stored in X-Data space, <code>cosFactors</code> and <code>twiddleFactors</code> point to the actual address where the factors are allocated. If the transform factors are stored in program memory, <code>cosFactors</code> and <code>twiddleFactors</code> are the offset from the program page boundary where the factors are allocated. This latter value can be calculated using the inline assembly operator <code>psvoffset()</code>.</p> <p>If the transform factors are stored in X-Data space, <code>factPage</code> must be set to 0xFF00 (defined value <code>COEFFS_IN_DATA</code>). If they are stored in program memory, <code>factPage</code> is the program page number containing the factors. This latter value can be calculated using the inline assembly operator <code>psvpage()</code>.</p> <p>The twiddle factors <i>must</i> be initialized with <code>conjFlag</code> set to a value different than zero.</p> <p>Only the first N/2 cosine factors are needed.</p> <p>Output is scaled by the factor $1/(\sqrt{2N})$</p>
Source File:	<code>dctoop.asm</code>
Function Profile:	<p>System resources usage:</p> <p>W0..W5 used, not restored</p> <p>plus system resources from <code>VectorZeroPad</code>, and <code>DCTIP</code>.</p> <p>DO and REPEAT instruction usage:</p> <p>no DO instructions</p> <p>no REPEAT instructions</p> <p>plus DO/REPEAT instructions from <code>VectorZeroPad</code>, and <code>DCTIP</code>.</p> <p>Program words (24-bit instructions):</p> <p>16</p> <p>plus program words from <code>VectorZeroPad</code>, and <code>DCTIP</code>.</p> <p>Cycles (including C-function call and return overheads):</p> <p>22</p> <p>plus cycles from <code>VectorZeroPad</code>, and <code>DCTIP</code>.</p> <p>Note: In the description of <code>VectorZeroPad</code> the number of cycles reported includes 4 cycles of C-function call overhead. Thus, the number of actual cycles from <code>VectorZeroPad</code> to add to DCT is 4 less than whatever number is reported for a stand alone <code>VectorZeroPad</code>. In the same way, the number of actual cycles from <code>DCTIP</code> to add to DCT is 3 less than whatever number is reported for a stand alone <code>DCTIP</code>.</p>

DCTIP

Description:	DCTIP computes the Discrete Cosine Transform of a source vector in place.
Include:	<code>dsp.h</code>

DCTIP (Continued)

Prototype:	<pre>extern fractional* DCTIP (int log2N, fractional* srcV, fractcomplex* cosFactors, fractcomplex* twidFactors, int factPage);</pre>											
Arguments:	<table><tr><td><i>log2N</i></td><td>based 2 logarithm of N (number of complex elements in source vector)</td></tr><tr><td><i>srcCV</i></td><td>pointer to source vector</td></tr><tr><td><i>cosFactors</i></td><td>pointer to cosine factors</td></tr><tr><td><i>twidFactors</i></td><td>pointer to twiddle factors</td></tr><tr><td><i>factPage</i></td><td>memory page for transform factors</td></tr></table>	<i>log2N</i>	based 2 logarithm of N (number of complex elements in source vector)	<i>srcCV</i>	pointer to source vector	<i>cosFactors</i>	pointer to cosine factors	<i>twidFactors</i>	pointer to twiddle factors	<i>factPage</i>	memory page for transform factors	
<i>log2N</i>	based 2 logarithm of N (number of complex elements in source vector)											
<i>srcCV</i>	pointer to source vector											
<i>cosFactors</i>	pointer to cosine factors											
<i>twidFactors</i>	pointer to twiddle factors											
<i>factPage</i>	memory page for transform factors											
Return Value:	Pointer to base address of destination vector.											
Remarks:	<p>N <i>must</i> be an integer power of 2.</p> <p>This function expects that the source vector has been zero padded to length 2N.</p> <p>The <i>srcV</i> vector must be allocated at a modulo alignment of N.</p> <p>The results of computation are stored in the first N elements of source vector.</p> <p>To avoid saturation (overflow) during computation, the values of the source vector <i>should</i> be in the range [-0.5, 0.5].</p> <p>Only the first N/2 cosine factors are needed.</p> <p>Only the first N/2 twiddle factors are needed.</p> <p>If the transform factors are stored in X-Data space, <i>cosFactors</i> and <i>twidFactors</i> point to the actual address where the factors are allocated. If the transform factors are stored in program memory, <i>cosFactors</i> and <i>twidFactors</i> are the offset from the program page boundary where the factors are allocated. This latter value can be calculated using the inline assembly operator <i>psvoffset()</i>.</p> <p>If the transform factors are stored in X-Data space, <i>factPage</i> must be set to 0xFF00 (defined value <i>COEFFS_IN_DATA</i>). If they are stored in program memory, <i>factPage</i> is the program page number containing the factors. This latter value can be calculated using the inline assembly operator <i>psvpage()</i>.</p> <p>The twiddle factors <i>must</i> be initialized with <i>conjFlag</i> set to a value different than zero.</p> <p>Output is scaled by the factor $1/(\sqrt{2N})$.</p>											
Source File:	dctoop.asm											

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* twiddleFactors,  
    int factPage  
);
```

Arguments:

<i>log2N</i>	based 2 logarithm of N (number of complex elements in source vector)
<i>dstCV</i>	pointer to destination complex vector
<i>srcCV</i>	pointer to source complex vector
<i>twiddleFactors</i>	base address of twiddle factors
<i>factPage</i>	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:

`fft0op.asm`

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:	extern fractcomplex* FFTComplexIP (int log2N, fractcomplex* srcCV, fractcomplex* twiddleFactors, int factPage);	
Arguments:	log2N	based 2 logarithm of N (number of complex elements in source vector)
	srcCV	pointer to source complex vector
	twiddleFactors	base address of twiddle factors
	factPage	memory page for transform factors
Return Value:	Pointer to base address of source complex vector.	
Remarks:	<p>N <i>must</i> be an integer power of 2.</p> <p>The elements in source complex vector are expected in natural order. The resulting transform is stored in bit reverse order.</p> <p>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.</p> <p>If the twiddle factors are stored in X-Data space, twiddleFactors points to the actual address where the factors are allocated. If the twiddle factors are stored in program memory, twiddleFactors 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().</p> <p>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().</p> <p>The twiddle factors <i>must</i> be initialized with conjFlag set to zero.</p> <p>Output is scaled by the factor 1/N.</p>	
Source File:	fft.asm	
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):	

Transform Size	# Cycles if Twiddle Factors in X-mem	# Cycles if Twiddle Factors in P-mem
32 point	1,633	1,795
64 point	3,739	4,125
128 point	8,485	9,383
256 point	19,055	21,105

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:

<i>log2N</i>	based 2 logarithm of N (number of complex elements in source vector)
<i>dstCV</i>	pointer to destination complex vector
<i>srcCV</i>	pointer to source complex vector
<i>twidFactors</i>	base address of twiddle factors
<i>factPage</i>	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.asm

IFFTComplex (Continued)

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* twiddleFactors,  
    int factPage  
);
```

Arguments:

<i>log2N</i>	based 2 logarithm of N (number of complex elements in source vector)
<i>srcCV</i>	pointer to source complex vector
<i>twiddleFactors</i>	base address of twiddle factors
<i>factPage</i>	memory page for transform factors

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

IFFTComplexIP (Continued)

Remarks:	<p>N <i>must</i> be an integer power of 2.</p> <p>The elements in source complex vector are expected in bit reverse order. The resulting transform is stored in natural order.</p> <p>The <code>srcCV</code> vector must be allocated at a modulo alignment of N.</p> <p>To avoid saturation (overflow) during computation, the magnitude of the values of the source complex vector <i>should</i> be in the range [-0.5, 0.5].</p> <p>If the twiddle factors are stored in X-Data space, <code>twidFactors</code> points to the actual address where the factors are allocated. If the twiddle factors are stored in program memory, <code>twidFactors</code> is the offset from the program page boundary where the factors are allocated. This latter value can be calculated using the inline assembly operator <code>psvoffset()</code>.</p> <p>If the twiddle factors are stored in X-Data space, <code>factPage</code> must be set to <code>0xFF00</code> (defined value <code>COEFFS_IN_DATA</code>). If they are stored in program memory, <code>factPage</code> is the program page number containing the factors. This latter value can be calculated using the inline assembly operator <code>psvpage()</code>.</p> <p>The twiddle factors <i>must</i> be initialized with <code>conjFlag</code> set to a value other than zero.</p> <p>Only the first N/2 twiddle factors are needed.</p>
Source File:	<code>ifft.asm</code>
Function Profile:	<p>System resources usage:</p> <p>W0..W3 used, not restored plus system resources from <code>FFTComplexIP</code>, and <code>BitReverseComplex</code>.</p> <p>DO and REPEAT instruction usage:</p> <p>no DO instructions no REPEAT instructions plus DO/REPEAT instructions from <code>FFTComplexIP</code>, and <code>BitReverseComplex</code>.</p> <p>Program words (24-bit instructions):</p> <p>11 plus program words from <code>FFTComplexIP</code>, and <code>BitReverseComplex</code>.</p> <p>Cycles (including C-function call and return overheads):</p> <p>15 plus cycles from <code>FFTComplexIP</code>, and <code>BitReverseComplex</code>.</p> <p>Note: In the description of <code>FFTComplexIP</code> the number of cycles reported includes 3 cycles of C-function call overhead. Thus, the number of actual cycles from <code>FFTComplexIP</code> to add to <code>IFFTComplexIP</code> is 3 less than whatever number is reported for a stand alone <code>FFTComplexIP</code>. In the same way, the number of actual cycles from <code>BitReverseComplex</code> to add to <code>IFFTComplexIP</code> is 2 less than whatever number is reported for a stand alone <code>BitReverseComplex</code>.</p>

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:

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

$$WN(k) = e^{j\frac{2\pi k}{N}}, \text{ where } 0 \leq k \leq N/2, \text{ for } conjFlag \neq 0$$

Include: dsp.h

Prototype:

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

Arguments:

<i>log2N</i>	based 2 logarithm of N (number of complex factors needed by a DFT)
<i>twidFactors</i>	pointer to complex twiddle factors
<i>conjFlag</i>	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

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 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.

Chapter 3. dsPIC Peripheral Libraries

3.1 INTRODUCTION

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

Code size for each library function or macro may be found in the file “Readme for dsPIC Language Tools Libraries.txt”.

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:\pic30_tools`) contains the following subdirectories with library-related files:

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

3.2 HIGHLIGHTS

This chapter is organized as follows:

- Using the dsPIC 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
- QEI Functions
- PWM Functions
- I2C Functions

3.3 USING THE DSPIC PERIPHERAL LIBRARIES

Building an application which utilizes the dsPIC Peripheral Libraries requires only two files: the processor-specific header file and the processor-specific library file.

The 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 `pDeviceRev.h`, where *Device* = dsPIC device number (e.g., `p30F6014.h` for the dsPIC30F6014 device) and *Rev* = revision of dsPIC device silicon (if appropriate.)

The library files are of the form `libpDeviceRev.a`, where *Device* = dsPIC device number (e.g., `libp30F6014.a` for the dsPIC30F6014 device) and *Rev* = revision of dsPIC device silicon (if appropriate.)

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.

Use the batch file `makelib.bat` to rebuild the libraries for all the supported devices.

3.4 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

3.4.1 Individual Functions

BusyXLCD

Description:	This function checks for the busy flag of the P-tec PCOG1602B LCD controller.
Include:	<code>xlcd.h</code>
Prototype:	<code>char BusyXLCD(void);</code>
Arguments:	None
Return Value:	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.
Remarks:	This function returns the status of the busy flag of the P-tec PCOG1602B LCD controller.
Source File:	<code>BusyXLCD.c</code>
Code Example:	<code>while (BusyXLCD());</code>

OpenXLCD

Description:	This function configures the I/O pins and initializes the P-tec PCOG1602B LCD controller.																								
Include:	xlcd.h																								
Prototype:	void OpenXLCD (unsigned char <i>lcdtype</i>);																								
Arguments:	<div><i>lcdtype</i> This contains the LCD controller parameters to be configured as defined below: <u>Type of interface</u> FOUR_BIT EIGHT_BIT <u>Number of lines</u> SINGLE_LINE TWO_LINE <u>Segment data transfer direction</u> SEG1_50_SEG51_100 SEG1_50_SEG100_51 SEG100_51_SEG50_1 SEG100_51_SEG1_50 <u>COM data transfer direction</u> COM1_COM16 COM16_COM1</div>																								
Return Value:	None																								
Remarks:	<p>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:</p> <p><u>Control I/O pin definitions</u></p> <table><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></table> <p>where x is the PORT, ? is the pin number</p> <p><u>Data Pin definitions</u></p> <table><tr><td>DATA_PIN_?</td><td>PORTxbits.RD?</td></tr><tr><td>TRIS_DATA_PIN_?</td><td>TRISxbits.TRISD?</td></tr></table> <p>where x is the PORT, ? is the pin number</p> <p>The Data pins can be from either one port or from multiple ports.</p> <p>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.</p> <p>After these definitions have been made, the user must compile the application code into an object to be linked.</p> <p>This function also requires three external routines for specific delays:</p> <table><tr><td>DelayFor18TCY()</td><td>18 Tcy delay</td></tr><tr><td>DelayPORXLCD()</td><td>15ms delay</td></tr><tr><td>DelayXLCD()</td><td>5ms delay</td></tr><tr><td>Delay100XLCD()</td><td>100Tcy delay</td></tr></table>	RW_PIN	PORTxbits.Rx?	TRIS_RW	TRISxbits.Rx?	RS_PIN	PORTxbits.Rx?	TRIS_RS	TRISxbits.Rx?	E_PIN	PORTxbits.Rx?	TRIS_E	TRISxbits.Rx?	DATA_PIN_?	PORTxbits.RD?	TRIS_DATA_PIN_?	TRISxbits.TRISD?	DelayFor18TCY()	18 Tcy delay	DelayPORXLCD()	15ms delay	DelayXLCD()	5ms delay	Delay100XLCD()	100Tcy delay
RW_PIN	PORTxbits.Rx?																								
TRIS_RW	TRISxbits.Rx?																								
RS_PIN	PORTxbits.Rx?																								
TRIS_RS	TRISxbits.Rx?																								
E_PIN	PORTxbits.Rx?																								
TRIS_E	TRISxbits.Rx?																								
DATA_PIN_?	PORTxbits.RD?																								
TRIS_DATA_PIN_?	TRISxbits.TRISD?																								
DelayFor18TCY()	18 Tcy delay																								
DelayPORXLCD()	15ms delay																								
DelayXLCD()	5ms delay																								
Delay100XLCD()	100Tcy delay																								
Source File:	openXLCD.c																								

OpenXLCD (Continued)

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: `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 till 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
PutrsXLCD.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: `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 `SetRamAddr()` function that was called where ?? can be CG or DD.

Source File: `ReadAddrXLCD.c`

Code Example: `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: `char ReadDataXLCD (void);`

ReadDataXLCD (Continued)

Arguments:	None
Remarks:	<p>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 <code>BusyXLCD()</code> function.</p> <p>The data read from the controller is for the character generator RAM or the display data RAM depending on the previous <code>Set??RamAddr()</code> function that was called where ?? is either CG or DD.</p>
Return Value:	This function returns the 8-bit data value pointed by the address.
Source File:	<code>ReadDataXLCD.c</code>
Code Example:	<pre>char data; while (BusyXLCD()); data = ReadDataXLCD();</pre>

SetCGRamAddr

Description:	This function sets the character generator address.
Include:	<code>xlcd.h</code>
Prototype:	<code>void SetCGRamAddr (unsigned char CGaddr);</code>
Arguments:	<i>CGaddr</i> Character generator address.
Return Value:	None
Remarks:	<p>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 <code>BusyXLCD()</code> function.</p>
Source File:	<code>SetCGRamAddr.c</code>
Code Example:	<pre>char cgaddr = 0x1F; while (BusyXLCD()); SetCGRamAddr(cgaddr);</pre>

SetDDRamAddr

Description:	This function sets the display data address.
Include:	<code>xlcd.h</code>
Prototype:	<code>void SetDDRamAddr (unsigned char DDaddr);</code>
Arguments:	<i>DDaddr</i> Display data address.
Return Value:	None
Remarks:	<p>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 <code>BusyXLCD()</code> function.</p>
Source File:	<code>SetDDRamAddr.c</code>
Code Example:	<pre>char ddaddr = 0x10; while (BusyXLCD()); SetDDRamAddr(ddaddr);</pre>

WriteDataXLCD

Description:	This function writes a data byte (one character) from the P-tec PCOG1602B LCD controller.
Include:	<code>xlcd.h</code>

WriteDataXLCD (Continued)

Prototype:	<code>void WriteDataXLCD (char data);</code>
Arguments:	<i>data</i> The value of <i>data</i> 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:	<p>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 <code>BusyXLCD()</code> function.</p> <p>The data read from the controller is for the character generator RAM or the display data RAM depending on the previous <code>Set??RamAddr()</code> function that was called where ?? refers to either CG or DD.</p>
Source File:	<code>WriteDataXLCD.c</code>
Code Example:	<code>WriteDataXLCD(0x30);</code>

WriteCmdXLCD

Description:	This function writes a command to the P-tec PCOG1602B LCD controller.
Include:	<code>xlcd.h</code>
Prototype:	<code>void WriteCmdXLCD (unsigned char cmd);</code>
Arguments:	<p><i>cmd</i> This contains the LCD controller parameters to be configured as defined below:</p> <p><u>Type of interface</u> <code>FOUR_BIT</code> <code>EIGHT_BIT</code></p> <p><u>Number of lines</u> <code>SINLE_LINE</code> <code>TWO_LINE</code></p> <p><u>Segment data transfer direction</u> <code>SEG1_50_SEG51_100</code> <code>SEG1_50_SEG100_51</code> <code>SEG100_51_SEG50_1</code> <code>SEG100_51_SEG1_50</code></p> <p><u>COM data transfer direction</u> <code>COM1_COM16</code> <code>COM16_COM1</code></p> <p><u>Display On/Off control</u> <code>DON</code> <code>DOFF</code> <code>CURSOR_ON</code> <code>CURSOR_OFF</code> <code>BLINK_ON</code> <code>BLINK_OFF</code></p> <p><u>Cursor or Display Shift defines</u> <code>SHIFT_CUR_LEFT</code> <code>SHIFT_CUR_RIGHT</code> <code>SHIFT_DISP_LEFT</code> <code>SHIFT_DISP_RIGHT</code></p>
Return Value:	None

WriteCmdXLCD (Continued)

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);
```

3.4.2 Example of Use

```
#define __dsPIC30F6014__
#include <p30fxxxx.h>
#include<xlcd.h>
/* holds the address of message */
char * buffer;
char data ;
char mesg1[] = {'H','A','R','D','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_SEG51_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 display data in the second line */
  SetDDRamAddr(0x40);
  while(BusyXLCD());
  buffer = mesg2;
  PutsXLCD(buffer);
  while(BusyXLCD());
  return 0;
}
```

3.5 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.

3.5.1 Individual Functions

CAN1AbortAll CAN2AbortAll

Description: This function initiates abort of all the pending transmissions.

Include: `can.h`

Prototype:

```
void CAN1AbortAll(void);
void CAN2AbortAll(void);
```

CAN1AbortAll (Continued) CAN2AbortAll

Arguments:	None
Return Value:	None
Remarks:	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.
Source File:	CAN1AbortAll.c CAN2AbortAll.c
Code Example:	CAN1AbortAll();

CAN1GetRXErrorCount CAN2GetRXErrorCount

Description:	This function returns the receive error count value.
Include:	can.h
Prototype:	unsigned char CAN1GetRXErrorCount(void); unsigned char CAN2GetRXErrorCount(void);
Arguments:	None
Return Value:	contents of CiRERRCNT, which is 8 bits.
Remarks:	This function returns the contents of CiRERRCNT (lower byte of CiEC register) which indicates the receive error count.
Source File:	CAN1GetRXErrorCount.c CAN2GetRXErrorCount.c
Code Example:	unsigned char rx_error_count; rx_error_count = CAN1GetRXErrorCount();

CAN1GetTXErrorCount CAN2GetTXErrorCount

Description:	This function returns the transmit error count value.
Include:	can.h
Prototype:	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:	unsigned char tx_error_count; tx_error_count = CAN1GetTXErrorCount();

CAN1IsBusOff CAN2IsBusOff

Description:	This function determines whether the CAN node is in BusOff mode.
Include:	can.h

CAN1IsBusOff (Continued) CAN2IsBusOff

Prototype: `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: `while (CAN1IsBusOff());`

CAN1IsRXReady CAN2IsRXReady

Description: This function returns the receive buffer full status.

Include: `can.h`

Prototype: `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 Recieve control register.

Source File: `CAN1IsRXReady.c`
`CAN2IsRXReady.c`

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

CAN1IsRXPassive CAN2IsRXPassive

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

Include: `can.h`

Prototype: `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`

CAN1IsRXPassive (Continued) CAN2IsRXPassive

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

CAN1IsTXPassive CAN2IsTXPassive

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

Include: `can.h`

Prototype: `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: `char tx_bus_status;
 tx_bus_status = CAN1IsTXPassive();`

CAN1IsTXReady CAN2IsTXReady

Description: This function returns the transmitter status indicating if the CAN node is ready for next transmission.

Include: `can.h`

Prototype: `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: `char tx_2_status;
 tx_2_status = CAN1IsTXReady(2);`

CAN1ReceiveMessage CAN2ReceiveMessage

Description: This function read the data from the receive buffer.

Include: `can.h`

CAN1ReceiveMessage (Continued) CAN2ReceiveMessage

Prototype:	<pre>void CAN1ReceiveMessage(unsigned char * data, unsigned char datalen, char MsgFlag); void CAN2ReceiveMessage(unsigned char * data, unsigned char datalen, char MsgFlag);</pre>	
Arguments:	<i>data</i>	The pointer to the location where received data is to be stored from.
	<i>datalen</i>	The number of bytes of data expected.
	<i>MsgFlag</i>	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 <i>data</i> .	
Return Value:	None.	
Source File:	CAN1ReceiveMessage.c CAN2ReceiveMessage.c	
Code Example:	<pre>unsigned char*rx_data; CAN1ReceiveMessage(rx_data, 5, 0);</pre>	

CAN1SendMessage CAN2SendMessage

Description:	This function writes data to be transmitted to TX registers, sets the data length and initiates the transmission.	
Include:	can.h	
Prototype:	<pre>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);</pre>	
Arguments:	<i>sid</i>	The 16-bit value to be written into CiTXnSID registers. CAN_TX_SID(x) x is the required SID value. <u>Substitute Remote Request</u> CAN_SUB_REM_TX_REQ CAN_SUB_NOR_TX_REQ <u>Message ID Type</u> CAN_TX_EID_EN CAN_TX_EID_DIS
	<i>eid</i>	The 32-bit value to be written into CiTXnEID and CiTXnDLC registers. CAN_TX_EID(x) x is the required EID value. <u>Substitute Remote Request</u> CAN_REM_TX_REQ CAN_NOR_TX_REQ
	<i>data</i>	The pointer to the location where data to be transmitted is stored.
	<i>datalen</i>	The number of bytes of data to be transmitted.

CAN1SendMessage (Continued) CAN2SendMessage

<i>MsgFlag</i>	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:	<pre>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);</pre>

CAN1SetFilter CAN2SetFilter

Description:	This function sets the acceptance filter values (SID and EID) for the specified filter.						
Include:	can.h						
Prototype:	<pre>void CAN1SetFilter(char filter_no, unsigned int sid, unsigned long eid); void CAN2SetFilter(char filter_no, unsigned int sid, unsigned long eid);</pre>						
Arguments:	<table border="0"> <tr> <td><i>filter_no</i></td><td>The filter (0, 1, 2, 3, 4 or 5) for which new acceptance values have to be configured.</td></tr> <tr> <td><i>sid</i></td><td>The 16-bit value to be written into CiRXFnSID registers. CAN_FILTER_SID(x) x is the required SID value. <u>Type of message to be received</u> CAN_RX_EID_EN CAN_RX_EID_DIS</td></tr> <tr> <td><i>eid</i></td><td>The 32-bit value to be written into CiRXFnEIDH and CiRXFnEIDL registers. CAN_FILTER_EID(x) x is the required EID value.</td></tr> </table>	<i>filter_no</i>	The filter (0, 1, 2, 3, 4 or 5) for which new acceptance values have to be configured.	<i>sid</i>	The 16-bit value to be written into CiRXFnSID registers. CAN_FILTER_SID(x) x is the required SID value. <u>Type of message to be received</u> CAN_RX_EID_EN CAN_RX_EID_DIS	<i>eid</i>	The 32-bit value to be written into CiRXFnEIDH and CiRXFnEIDL registers. CAN_FILTER_EID(x) x is the required EID value.
<i>filter_no</i>	The filter (0, 1, 2, 3, 4 or 5) for which new acceptance values have to be configured.						
<i>sid</i>	The 16-bit value to be written into CiRXFnSID registers. CAN_FILTER_SID(x) x is the required SID value. <u>Type of message to be received</u> CAN_RX_EID_EN CAN_RX_EID_DIS						
<i>eid</i>	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 <i>sid</i> into the CiRXFnSID register and or the 32-bit value of <i>eid</i> into the CiRXFnEIDH and CiRXFnEIDL registers corresponding to the filter specified by <i>filter_no</i> . Filter 0 is taken as default.						
Source File:	CAN1SetFilter.c CAN2SetFilter.c						
Code Example:	<pre>CAN1SetFilter(1, CAN_FILTER_SID(7) & CAN_RX_EID_EN, CAN_FILTER_EID(3));</pre>						

CAN1SetMask CAN2SetMask

Description:	This function sets the acceptance mask values (SID and EID) for the specified mask.
Include:	can.h
Prototype:	<pre>void CAN1SetMask(char mask_no, unsigned int sid, unsigned long eid); void CAN2SetMask(char mask_no, unsigned int sid, unsigned long eid);</pre>
Arguments:	<p><i>mask_no</i> The mask ('0' or '1') for which mask values have to be configured.</p> <p><i>sid</i> The 16-bit value to be written into CiRXMnSID registers. CAN_MASK_SID(x) x is the required SID value. <u>Match/ignore message type specified in filter</u> CAN_MATCH_FILTER_TYPE CAN_IGNORE_FILTER_TYPE</p> <p><i>eid</i> The 32-bit value to be written into CiRXMnEIDH and CiRXMnEIDL registers. CAN_MASK_EID(x) x is the required EID value.</p>
Return Value:	None
Remarks:	<p>This function writes the 16-bit value of <i>sid</i> into the CiRXFnSID register and or the 32-bit value of <i>eid</i> into the CiRXFnEIDH and CiRXFnEIDL registers corresponding to the mask specified by <i>mask_no</i>. Filter 0 is taken as default.</p>
Source File:	CAN1SetMask.c CAN2SetMask.c
Code Example:	<pre>CAN1SetMask(1, CAN_MASK_SID(7) & CAN_MATCH_FILTER_TYPE, CAN_MASK_EID(3));</pre>

CAN1SetOperationMode CAN2SetOperationMode

Description:	This function configures the CAN module
Include:	can.h
Prototype:	<pre>void CAN1SetOperationMode(unsigned int config); void CAN2SetOperationMode(unsigned int config);</pre>
Arguments:	<p><i>config</i> The 16-bit value to be loaded into CiCTRL register, the combination of the following defines.</p> <p>CAN_IDLE_CON CAN On in idle mode CAN_IDLE_STOP CAN Stop in idle mode</p> <p>CAN_MASTERCLOCK_1 F_{CAN} is F_{cy} CAN_MASTERCLOCK_0 F_{CAN} is $4F_{cy}$</p> <p><u>CAN modes of operation</u> CAN_REQ_OPERMODE_NOR CAN_REQ_OPERMODE_DIS CAN_REQ_OPERMODE_LOOPBK CAN_REQ_OPERMODE_LISTENONLY CAN_REQ_OPERMODE_CONFIG CAN_REQ_OPERMODE_LISTENALL</p>

CAN1SetOperationMode (Continued) CAN2SetOperationMode

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);

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_CON_NO_WAIT CAN On in idle mode

CAN_IDLE_STOP_NO_WAIT CAN Stop in idle mode

CAN_MASTERCLOCK_1_NO_WAIT F_{CAN} is F_{cy}

CAN_MASTERCLOCK_0_NO_WAIT F_{CAN} is 4F_{cy}

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_CON &
CAN_MASTERCLOCK_1 & CAN_REQ_OPERMODE_LISTEN);

CAN1SetRXMode CAN2SetRXMode

Description:	This function configures the CAN receiver.
Include:	can.h
Prototype:	<pre>void CAN1SetRXMode(char buffno, unsigned int config); void CAN2SetRXMode(char buffno, unsigned int config);</pre>
Arguments:	<p><i>buffno</i> buffno indicates the control reg to be configured.</p> <p><i>config</i> The value to be written into CiRXnCON reg, the combination of the following defines.</p> <p><u>Clear RXFUL bit</u> CAN_RXFUL_CLEAR</p> <p><u>Double buffer enable/disable</u> CAN_BUF0_DBLBUFFER_EN CAN_BUF0_DBLBUFFER_DIS</p>
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 CAN2SetRXMode.c
Code Example:	<pre>CAN1SetRXMode(0, CAN_RXFUL_CLEAR & CAN_BUF0_DBLBUFFER_EN);</pre>

CAN1SetTXMode (function) CAN2SetTXMode

Description:	This function configures the CAN transmitter module
Include:	can.h
Prototype:	<pre>void CAN1SetTXMode(char buffno, unsigned int config); void CAN2SetTXMode(char buffno, unsigned int config);</pre>
Arguments:	<p><i>buffno</i> buffno indicates the control reg to be configured.</p> <p><i>config</i> The value to be written into CiTXnCON reg, the combination of the following defines.</p> <p><u>Message send request</u> CAN_TX_REQ CAN_TX_STOP_REQ</p> <p><u>Message transmission priority</u> CAN_TX_PRIORITY_HIGH CAN_TX_PRIORITY_HIGH_INTER CAN_TX_PRIORITY_LOW_INTER CAN_TX_PRIORITY_LOW</p>
Return Value:	None
Remarks:	This function configures the following bits of CiTXnCON register: TXRTR, TXREQ, DLC, TXPRI<1:0>
Source File:	CAN1SetTXMode.c CAN2SetTXMode.c

CAN1SetTXMode (Continued)(function) CAN2SetTXMode

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_PROPAGATIONTIME_SEG_TQ(x)
 (((x-1) & 0x7) | 0xC7F8)

CAN_PHASE_SEG1_TQ(x)
 ((((x-1) & 0x7) * 0x8) | 0xC7C7)

CAN_PHASE_SEG2_TQ(x)
 ((((x-1) & 0x7) * 0x100) | 0xC0FF)

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

CAN1Initialize (Continued) CAN2Initialize

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 ConfigIntCAN2

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:
User must enter either enable or disable option for all the individual interrupts.

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
```

ConfigIntCAN1 (Continued)

ConfigIntCAN2

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

Code Example:

```
ConfigIntCAN1(CAN_INDI_INVMESS_EN &
               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 ,
               CAN_INT_PRI_3 &
               CAN_INT_ENABLE);
```

3.5.2 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);

3.5.3 Example of Use

```
#define __dsPIC30F6014__
#include<p30fxxx.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_MASTERCLOCK_1 &
        CAN_REQ_OPERMODE_CONFIG );
    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_SAMPLE1TIME);
    /* 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;
}
```

3.6 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.

3.6.1 Individual Functions

BusyADC12

Description:	This function returns the ADC conversion status.
Include:	adc12.h
Prototype:	char BusyADC12(void);
Arguments:	None
Return Value:	If the value of CONV is '1', then '1' is returned, indicating that the ADC is busy in conversion. If the value of CONV is '0', then '0' is returned, indicating that the ADC has completed conversion.
Remarks:	This function returns the status of the ADCON1 <CONV> bit status which indicates whether the ADC is busy in conversion.
Source File:	BusyADC12.c
Code Example:	while (BusyADC12());

CloseADC12

Description:	This function turns off the ADC module and disables the ADC interrupts.
Include:	adc12.h
Prototype:	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

CloseADC12 (Continued)

Code Example: `CloseADC12();`

ConfigIntADC12

Description: This function configures the ADC interrupt.

Include: `adc12.h`

Prototype: `void ConfigIntADC12(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`

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`

OpenADC12 (Continued)

Prototype:	<code>void OpenADC12(unsigned int <i>config1</i>, unsigned int <i>config2</i>, unsigned int <i>config3</i>, unsigned int <i>configport</i>, unsigned int <i>configscan</i>)</code>	
Arguments:	<i>config1</i>	<p>This contains the parameters to be configured in the ADCON1 register as defined below:</p> <p><u>Module On/Off</u> ADC_MODULE_ON ADC_MODULE_OFF</p> <p><u>Idle mode operation</u> ADC_IDLE_CONTINUE ADC_IDLE_STOP</p> <p><u>Result output format</u> ADC_FORMAT_SIGN_FRACT ADC_FORMAT_FRACT ADC_FORMAT_SIGN_INT ADC_FORMAT_INTG</p> <p><u>Conversion trigger source</u> ADC_CLK_AUTO ADC_CLK_TMR ADC_CLK_INT0 ADC_CLK_MANUAL</p> <p><u>Auto sampling select</u> ADC_AUTO_SAMPLING_ON ADC_AUTO_SAMPLING_OFF</p> <p><u>Sample enable</u> ADC_SAMP_ON ADC_SAMP_OFF</p>
	<i>config2</i>	<p>This contains the parameters to be configured in the ADCON2 register as defined below:</p> <p><u>Voltage Reference</u> ADC_VREF_AVDD_AVSS ADC_VREF_EXT_AVSS ADC_VREF_AVDD_EXT ADC_VREF_EXT_EXT</p> <p><u>Scan selection</u> ADC_SCAN_ON ADC_SCAN_OFF</p> <p><u>Number of samples between interrupts</u> ADC_SAMPLES_PER_INT_1 ADC_SAMPLES_PER_INT_2 ADC_SAMPLES_PER_INT_15 ADC_SAMPLES_PER_INT_16</p> <p><u>Buffer mode select</u> ADC_ALT_BUF_ON ADC_ALT_BUF_OFF</p> <p>Alternate input sample mode select ADC_ALT_INPUT_ON ADC_ALT_INPUT_OFF</p>
	<i>config3</i>	<p>This contains the parameters to be configured in the ADCON3 register as defined below:</p>

OpenADC12 (Continued)

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_CLOCK_Tcy2

ADC_CLOCK_Tcy

ADC_CLOCK_3Tcy2

.....

ADC_CLOCK_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

OpenADC12 (Continued)

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_CLOCK_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:

SetChanADC12 (Continued)

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) ;

3.6.2 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) ;

3.6.3 Example of Use

```
#define __dsPIC30F6014__
#include <p30fxxx.h>
#include<adc12.h>
unsigned int Channel, PinConfig, Scanselct, 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;
    Scanselct = 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, Scanselct);
    i = 0;
    while( i <16 )
    {
        ADCON1bits.SAMP = 1;
        while(!ADCON1bits.SAMP);
        ConvertADC12();
        while(ADCON1bits.SAMP);
        while(!BusyADC12());
        while(BusyADC12());
        result[i] = ReadADC12(i);
        i++;
    }
}
```

3.7 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.

3.7.1 Individual Functions

BusyADC10

Description:	This function returns the ADC conversion status.
Include:	<code>adc10.h</code>
Prototype:	<code>char BusyADC10(void);</code>
Arguments:	None
Return Value:	If the value of CONV is '1', then '1' is returned, indicating that the ADC is busy in conversion. If the value of CONV is '0', then '0' is returned, indicating that the ADC has completed conversion.
Remarks:	This function returns the status of the ADCON1 <CONV> bit status which indicates whether the ADC is busy in conversion.
Source File:	<code>BusyADC10.c</code>
Code Example:	<code>while(BusyADC10());</code>

CloseADC10

Description:	This function turns off the ADC module and disables the ADC interrupts.
Include:	<code>adc10.h</code>
Prototype:	<code>void CloseADC10(void);</code>
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:	<code>CloseADC10.c</code>
Code Example:	<code>CloseADC10();</code>

ConfigIntADC10

Description:	This function configures the ADC interrupt.
Include:	<code>adc10.h</code>
Prototype:	<code>void ConfigIntADC10(unsigned int config);</code>
Arguments:	<i>config</i> ADC interrupt priority and enable/disable information as defined below: <u>ADC Interrupt enable/disable</u> ADC_INT_ENABLE ADC_INT_DISABLE

ConfigIntADC10 (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: ConfigIntADC10.c

Code Example: ConfigIntADC10(ADC_INT_PRI_3 &
ADC_INT_DISABLE);

ConvertADC10

Description: This function starts the A/D conversion.

Include: adc10.h

Prototype: void ConvertADC10(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: ConvertADC10.c

Code Example: ConvertADC10();

OpenADC10

Description: This function configures the ADC.

Include: adc10.h

Prototype: void OpenADC10(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

OpenADC10 (Continued)

Result output format

ADC_FORMAT_SIGN_FRACT
ADC_FORMAT_FRACT
ADC_FORMAT_SIGN_INT
ADC_FORMAT_INTG

Conversion trigger source

ADC_CLK_AUTO
ADC_CLK_MPWM
ADC_CLK_TMR
ADC_CLK_INT0
ADC_CLK_MANUAL

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

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

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

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

OpenADC10 (Continued)

	<u>Conversion Clock Source select</u> ADC_CONV_CLK_INTERNAL_RC ADC_CONV_CLK_SYSTEM <u>Conversion clock select</u> ADC_CLOCK_Tcy2 ADC_CLOCK_Tcy ADC_CLOCK_3Tcy2 ADC_CLOCK_32Tcy
<i>configport</i>	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
<i>configscan</i>	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 behaviour, 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:	<pre>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_CLOCK_Tcy, ENABLE_AN1_ANA, SKIP_SCAN_AN0 & SKIP_SCAN_AN3 & SKIP_SCAN_AN4 & SKIP_SCAN_AN5);</pre>

ReadADC10

Description:	This function reads the ADC buffer register which contains the conversion value.
Include:	adc10.h
Prototype:	unsigned int ReadADC10(unsigned char <i>bufIndex</i>);
Arguments:	<i>bufIndex</i> 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 <i>bufIndex</i> value between '0' to '15' to ensure correct read of the ADCBUF0 to ADCBUFF.
Source File:	ReadADC10.c
Code Example:	<pre>unsigned int result; result = ReadADC10(3);</pre>

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 <i>channel</i>);
Arguments:	<i>channel</i> This contains the input select parameter to be configured into the ADCHS register as defined below: <u>A/D Channel 1, 2, 3 Negative input for Sample A</u> ADC_CHX_NEG_SAMPLEA_AN9AN10AN11 ADC_CHX_NEG_SAMPLEA_AN6AN7AN8 ADC_CHX_NEG_SAMPLEA_NVREF <u>A/D Channel 1, 2, 3 Negative input for Sample B</u> ADC_CHX_NEG_SAMPLEB_AN9AN10AN11 ADC_CHX_NEG_SAMPLEB_AN6AN7AN8 ADC_CHX_NEG_SAMPLEB_NVREF <u>A/D Channel 1, 2, 3 Positive input for Sample A</u> ADC_CHX_POS_SAMPLEA_AN3AN4AN5 ADC_CHX_POS_SAMPLEA_AN0AN1AN2 <u>A/D Channel 1, 2, 3 Positive input for Sample B</u> ADC_CHX_POS_SAMPLEA_AN3AN4AN5 ADC_CHX_POS_SAMPLEB_AN0AN1AN2 <u>A/D Channel 0 positive i/p select for Sample A</u> ADC_CH0_POS_SAMPLEA_AN0 ADC_CH0_POS_SAMPLEA_AN1 ADC_CH0_POS_SAMPLEA_AN15 <u>A/D Channel 0 negative i/p select for Sample A</u> ADC_CH0_NEG_SAMPLEA_AN1 ADC_CH0_NEG_SAMPLEA_NVREF

SetChanADC10 (Continued)

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 sample multiplexers A and B by writing to ADCHS register.

Source File: SetChanADC10.c

Code Example: SetChanADC10(ADC_CH0_POS_SAMPLEA_AN0 &
ADC_CH0_NEG_SAMPLEA_NVREF);

3.7.2 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);

3.7.3 Example of Use

```
#define __dsPIC30F6010__
#include <p30fxxx.h>
#include<adc10.h>
unsigned int Channel, PinConfig, Scanselct, 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;
    Scanselct = 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, Scanselct);
    i = 0;
    while(i <16 )
    {
        ADCON1bits.SAMP = 1;
        while(!ADCON1bits.SAMP);
        ConvertADC10();
        while(ADCON1bits.SAMP);
        while(!BusyADC10());
        while(BusyADC10());
        result[i] = ReadADC10(i);
        i++;
    }
}
```

3.8 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.

3.8.1 Individual Functions

CloseTimer1 CloseTimer2 CloseTimer3 CloseTimer4 CloseTimer5

Description:	This function turns off the 16-bit timer module.
Include:	timer.h
Prototype:	<pre>void CloseTimer1(void); void CloseTimer2(void); void CloseTimer3(void); void CloseTimer4(void); void CloseTimer5(void);</pre>
Arguments:	None
Return Value:	None
Remarks:	This function first disables the 16-bit timer interrupt and then turns off the timer module. The Interrupt flag bit (TxIF) is also cleared.
Source File:	<pre>CloseTimer1.c CloseTimer2.c CloseTimer3.c CloseTimer4.c CloseTimer5.c</pre>
Code Example:	<pre>CloseTimer1();</pre>

CloseTimer23 CloseTimer45

Description:	This function turns off the 32-bit timer module.
Include:	timer.h
Prototype:	<pre>void CloseTimer23 (void) void CloseTimer45 (void)</pre>
Arguments:	None
Return Value:	None
Remarks:	This function disables the 32-bit timer interrupt and then turns off the timer module. The Interrupt flag bit (TxIF) is also cleared. CloseTimer23 turns off Timer2 and disables Timer3 Interrupt. CloseTimer45 turns off Timer4 and disables Timer5 Interrupt.
Source File:	<pre>CloseTimer23.c CloseTimer45.c</pre>
Code Example:	<pre>CloseTimer23();</pre>

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:

ConfigIntTimer23 (Continued)

ConfigIntTimer45

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

32-bit timer mode disable (only if x = 2 or 4)
Tx_32BIT_MODE_OFF

Timer Gate time accumulation enable
Tx_GATE_ON
Tx_GATE_OFF

OpenTimer1 (Continued)

OpenTimer2

OpenTimer3

OpenTimer4

OpenTimer5

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

Return Value: None

Remarks: This function configures the 16-bit timer control register and sets the period match value into the PR register

Source File: OpenTimer1.c
OpenTimer2.c
OpenTimer3.c
OpenTimer4.c
OpenTimer5.c

Code Example:

```
OpenTimer1(T1_ON & T1_GATE_OFF &
            T1_PS_1_8 & T1_SYNC_EXT_OFF &
            T1_SOURCE_INT, 0xFF);
```

OpenTimer23

OpenTimer45

Description: This function configures the 32-bit timer module.

Include: timer.h

Prototype:

```
void OpenTimer23(unsigned int config,
                 unsigned long period);
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

32-bit timer mode enable

Tx_32BIT_MODE_ON

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

ReadTimer45

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

WriteTimer2

WriteTimer3

WriteTimer4

WriteTimer5

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

WriteTimer45

Description: This function writes the 32-bit value into the timer register.

Include: `timer.h`

Prototype: `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

WriteTimer23 (Continued)

WriteTimer45

Source File: WriteTimer23.c
WriteTimer45.c

Code Example: unsigned long timer23_init = 0xABCD;
WriteTimer23(timer23_init);

3.8.2 Individual Macros

EnableIntT1 EnableIntT2 EnableIntT3 EnableIntT4 EnableIntT5

Description: This macro enables the timer interrupt.

Include: timer.h

Arguments: None

Remarks: This macro sets timer interrupt enable bit of interrupt enable control register.

Code Example: EnableIntT1;

DisableIntT1 DisableIntT2 DisableIntT3 DisableIntT4 DisableIntT5

Description: This macro disables the timer interrupt.

Include: timer.h

Arguments: None

Remarks: This macro clears timer interrupt enable bit of interrupt enable control register.

Code Example: DisableIntT2;

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);

3.8.3 Example of Use

```
#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 = 0xFFFF;
    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();
}
```

3.9 RESET/CONTROL FUNCTIONS

This section contains a list of individual functions for Reset/Control. Functions may be implemented as macros.

3.9.1 Individual Functions

isBOR

Description:	This function checks if Reset is due to brown-out detect.
Include:	reset.h
Prototype:	char isBOR(void);
Arguments:	None
Return Value:	This function returns the RCON<BOR> bit status. If return value is 1, then reset is due to brown-out. If return value is 0, then no brown-out occurred.
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.
If return value is 1, then reset is due to Power on..
If return value is 0, then no Power on reset occurred.

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.
If return value is 1, then low voltage detect interrupt occurred.
If return value is 0, then low voltage detect interrupt did not occur.

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: `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:

```
char reset_state;
reset_state = isMCLR();
```

isWDTTO

Description: This function checks if reset condition is due to WDT time-out.

Include: `reset.h`

Prototype: `char isWDTTO(void);`

Arguments: None

isWDTTO (Continued)

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:

```
char reset_state;  
reset_state = isWDTTO();
```

isWDTWU

Description: This function checks if wakeup from sleep is due to WDT time-out.

Include: reset.h

Prototype: 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:

```
char reset_state;  
reset_state = isWDTWU();
```

isWU

Description: This function checks if wakeup from sleep is due to MCLR, POR, BOR or any interrupt.

Include: reset.h

Prototype: char isWU(void);

Arguments: None

Return Value: This function checks if wakeup from sleep has occurred.
If yes, it checks for the cause for wakeup.
if '1', wakeup is due to the occurrence of interrupt.
if '2', wakeup is due to MCLR.
if '3', wakeup is due to POR.
if '4', wakeup is due to BOR.
If wakeup from sleep has not occurred, then a value of '0' is returned.

Remarks: None

Source File: isWU.c

Code Example:

```
char reset_state;  
reset_state = isWU();
```

3.9.2 Individual Macros

DisableInterrupts

Description:	This macro disables all the peripheral interrupts for specified number of instruction cycles.
Include:	<code>reset.h</code>
Arguments:	<code>cycles</code>
Remarks:	This macro executes DISI instruction to disable all the peripheral interrupts for specified number of instruction cycles.
Code Example:	<code>DisableInterrupts(15);</code>

PORStatReset

Description:	This macro sets POR bit of RCON register to reset state.
Include:	<code>reset.h</code>
Arguments:	None
Remarks:	None
Code Example:	<code>PORStatReset;</code>

BORStatReset

Description:	This macro sets BOR bit of RCON register to reset state.
Include:	<code>reset.h</code>
Arguments:	None
Remarks:	None
Code Example:	<code>BORStatReset;</code>

WDTSWEnable

Description:	This macro turns on the watchdog timer
Include:	<code>reset.h</code>
Arguments:	None
Remarks:	This macro sets software wdt enable(SWDTEN) bit of RCON register
Code Example:	<code>WDTSWEnable;</code>

WDTSWDisable

Description:	This macro clears software wdt enable(SWDTEN) bit of RCON register
Include:	<code>reset.h</code>
Arguments:	None
Remarks:	This macro disables WDT if FWDTEN fuse bit is 0.
Code Example:	<code>WDTSWDisable;</code>

3.10 I/O PORT FUNCTIONS

This section contains a list of individual functions for I/O ports. Functions may be implemented as macros.

3.10.1 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
```

ConfigINT0 (Continued)

ConfigINT1

ConfigINT2

ConfigINT3

ConfigINT4

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 pullups. 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(0xFFFF);

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.

ConfigIntCN (Continued)

The lower 24 bits contain the individual enable/disable information for the CN interrupts. Setting bit x (x = 0, 1, ..., 23) would enable the CNx 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(CHANGE_INT_OFF & CHANGE_INT_PRI_4 &
0xFF000087);
```

3.10.2 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);`

3.11 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.

3.11.1 Individual Functions

CloseCapture1
CloseCapture2
CloseCapture3
CloseCapture4
CloseCapture5
CloseCapture6
CloseCapture7
CloseCapture8

Description:	This function turns off the Input Capture module.
Include:	InCap.h
Prototype:	<pre>void CloseCapture1(void); void CloseCapture2(void); void CloseCapture3(void); void CloseCapture4(void); void CloseCapture5(void); void CloseCapture6(void); void CloseCapture7(void); void CloseCapture8(void);</pre>
Arguments:	None
Return Value:	None
Remarks:	This function disables the Input Capture interrupt and then turns off the module. The Interrupt flag bit is also cleared.
Source File:	<pre>CloseCapture1.c CloseCapture2.c CloseCapture3.c CloseCapture4.c CloseCapture5.c CloseCapture6.c CloseCapture7.c CloseCapture8.c</pre>
Code Example:	<pre>CloseCapture1();</pre>

ConfigIntCapture1
ConfigIntCapture2
ConfigIntCapture3
ConfigIntCapture4
ConfigIntCapture5
ConfigIntCapture6
ConfigIntCapture7
ConfigIntCapture8

Description:	This function configures the Input Capture interrupt.
Include:	InCap.h

ConfigIntCapture1 (Continued)

ConfigIntCapture2

ConfigIntCapture3

ConfigIntCapture4

ConfigIntCapture5

ConfigIntCapture6

ConfigIntCapture7

ConfigIntCapture8

Prototype:

```
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: `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
```

OpenCapture1 (Continued)

OpenCapture2

OpenCapture3

OpenCapture4

OpenCapture5

OpenCapture6

OpenCapture7

OpenCapture8

Code Example: `OpenCapture1(IC_IDLE_CON & IC_TIMER2_SRC &
IC_INT_1CAPTURE & IC_EVERY_RISE_EDGE);`

ReadCapture1

ReadCapture2

ReadCapture3

ReadCapture4

ReadCapture5

ReadCapture6

ReadCapture7

ReadCapture8

Description:	This function reads all the pending Input Capture buffers.
Include:	InCap.h
Prototype:	<code>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);</code>
Arguments:	<i>buffer</i> 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 till the buffers are empty indicated by the ICxCON<ICBNE> bit getting cleared.
Source File:	<code>ReadCapture1.c ReadCapture2.c ReadCapture3.c ReadCapture4.c ReadCapture5.c ReadCapture6.c ReadCapture7.c ReadCapture8.c</code>
Code Example:	<code>unsigned int *buffer = 0x1900; ReadCapture1(buffer);</code>

3.11.2 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);

3.11.3 Example of Use

```
#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_1CAPTURE & 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();
}
```

3.12 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.

3.12.1 Individual Functions

CloseOC1
CloseOC2
CloseOC3
CloseOC4
CloseOC5
CloseOC6
CloseOC7
CloseOC8

Description:	This function turns off the Output Compare module.
Include:	outcompare.h
Prototype:	<pre>void CloseOC1(void); void CloseOC2(void); void CloseOC3(void); void CloseOC4(void); void CloseOC5(void); void CloseOC6(void); void CloseOC7(void); void CloseOC8(void);</pre>
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:	<pre>CloseOC1.c CloseOC2.c CloseOC3.c CloseOC4.c CloseOC5.c CloseOC6.c CloseOC7.c CloseOC8.c</pre>
Code Example:	CloseOC1();

ConfigIntOC1
ConfigIntOC2
ConfigIntOC3
ConfigIntOC4
ConfigIntOC5
ConfigIntOC6
ConfigIntOC7
ConfigIntOC8

Description:	This function configures the Output Compare interrupt.
Include:	outcompare.h

ConfigIntOC1 (Continued)

ConfigIntOC2

ConfigIntOC3

ConfigIntOC4

ConfigIntOC5

ConfigIntOC6

ConfigIntOC7

ConfigIntOC8

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);`

OpenOC1
OpenOC2
OpenOC3
OpenOC4
OpenOC5
OpenOC6
OpenOC7
OpenOC8

Description:	This function configures the Output Compare module.
Include:	outcompare.h
Prototype:	<pre>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);</pre>
Arguments:	<p><i>config</i> This contains the parameters to be configured in the OCxCON register as defined below:</p> <p><u>Idle mode operation</u> OC_IDLE_STOP OC_IDLE_CON</p> <p><u>Clock select</u> OC_TIMER2_SRC OC_TIMER3_SRC</p> <p><u>Output Compare modes of operation</u> 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</p> <p><i>value1</i> This contains the value to be stored into OCxRS Secondary Register.</p> <p><i>value2</i> This contains the value to be stored into OCxR Main Register.</p>
Return Value:	None
Remarks:	<p>This function configures the Output Compare module control register (OCxCON) with the following parameters:</p> <p>Clock select, mode of operation, operation in idle mode.</p> <p>It also configures the OCxRS and OCxR registers.</p>

OpenOC1 (Continued)

OpenOC2

OpenOC3

OpenOC4

OpenOC5

OpenOC6

OpenOC7

OpenOC8

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'.

ReadDCOC1PWM (Continued)

ReadDCOC2PWM

ReadDCOC3PWM

ReadDCOC4PWM

ReadDCOC5PWM

ReadDCOC6PWM

ReadDCOC7PWM

ReadDCOC8PWM

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.

ReadRegOC1 (Continued)

ReadRegOC2

ReadRegOC3

ReadRegOC4

ReadRegOC5

ReadRegOC6

ReadRegOC7

ReadRegOC8

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

Description: This function configures the Output Compare secondary duty cycle register (OCxRS) when the module is in PWM mode.

Include: outcompare.h

Prototype: 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*);

Arguments: *dutycycle* This is the duty cycle value to be stored into Output Compare secondary duty cycle register (OCxRS.)

Return Value: None

Remarks: The Output Compare secondary duty cycle register (OCxRS) will be configured with new value only if the module is in PWM mode.

SetDCOC1PWM (Continued)

SetDCOC2PWM

SetDCOC3PWM

SetDCOC4PWM

SetDCOC5PWM

SetDCOC6PWM

SetDCOC7PWM

SetDCOC8PWM

Source File: SetDCOC1PWM.c
SetDCOC2PWM.c
SetDCOC3PWM.c
SetDCOC4PWM.c
SetDCOC5PWM.c
SetDCOC6PWM.c
SetDCOC7PWM.c
SetDCOC8PWM.c

Code Example: SetDCOC1PWM(dutycycle);

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:

<i>pulse_start</i>	This is the value to be stored into Output Compare main register (OCxR).
<i>pulse_stop</i>	This is the value to be stored into Output Compare secondary register (OCxRS).

Return Value: None

SetPulseOC1 (Continued)

SetPulseOC2

SetPulseOC3

SetPulseOC4

SetPulseOC5

SetPulseOC6

SetPulseOC7

SetPulseOC8

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);

3.12.2 Individual Macros

EnableIntOC1

EnableIntOC2

EnableIntOC3

EnableIntOC4

EnableIntOC5

EnableIntOC6

EnableIntOC7

EnableIntOC8

Description: This macro enables the interrupt on output compare match.

Include: outcompare.h

Arguments: None

Remarks: This macro sets output compare(OC) interrupt enable bit of interrupt enable control register.

Code Example: EnableIntOC8;

DisableIntOC1

DisableIntOC2

DisableIntOC3

DisableIntOC4

DisableIntOC5

DisableIntOC6

DisableIntOC7

DisableIntOC8

Description: This macro disables the interrupt on compare match.

Include: outcompare.h

DisableIntOC1 (Continued)

DisableIntOC2

DisableIntOC3

DisableIntOC4

DisableIntOC5

DisableIntOC6

DisableIntOC7

DisableIntOC8

Arguments:	None
Remarks:	This macro clears OC interrupt enable bit of interrupt enable control register.
Code Example:	<code>DisableIntOC7;</code>

SetPriorityIntC1

SetPriorityIntC2

SetPriorityIntC3

SetPriorityIntC4

SetPriorityIntC5

SetPriorityIntC6

SetPriorityIntC7

SetPriorityIntC8

Description:	This macro sets priority for output compare interrupt.
Include:	<code>outcompare.h</code>
Arguments:	<i>priority</i>
Remarks:	This macro sets OC interrupt priority bits of interrupt priority control register.
Code Example:	<code>SetPriorityIntOC4(0);</code>

3.12.3 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;
}
```

3.13 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.

3.13.1 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.

CloseUART1 (Continued)

CloseUART2

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

DataRdyUART2

Description: This function returns the UART receive buffer status.

Include: uart.h

DataRdyUART1 (Continued) DataRdyUART2

Prototype: `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: `while(DataRdyUART1());`

OpenUART1 OpenUART2

Description: This function configures the UART module

Include: `uart.h`

Prototype: `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_ALTTX
UART_RX_TX
UART communication with ALT pins is available only for certain devices and the suitable data sheet should be referred to.
- UART wakeup 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
- Parity and data bits select
UART_NO_PAR_9BIT
UART_ODD_PAR_8BIT
UART_EVEN_PAR_8BIT
UART_NO_PAR_8BIT

OpenUART1 (Continued) OpenUART2

	<u>Number of STOP bits</u> UART_2STOPBITS UART_1STOPBIT
<i>config2</i>	This contains the parameters to be configured in the UxSTA register as defined below: <u>UART transmission mode interrupt select</u> UART_INT_TX_BUF_EMPTY UART_INT_TX <u>UART transmit break bit</u> UART_TX_PIN_NORMAL UART_TX_PIN_LOW <u>UART transmit enable/disable</u> UART_TX_ENABLE UART_TX_DISABLE <u>UART receive interrupt mode select</u> UART_INT_RX_BUF_FUL UART_INT_RX_3_4_FUL UART_INT_RX_CHAR <u>UART address detect enable/disable</u> UART_ADR_DETECT_EN UART_ADR_DETECT_DIS <u>UART OVERRUN bit clear</u> UART_RX_OVERRUN_CLEAR
<i>ubrg</i>	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:	<pre> baud = 5; UMODEvalue = UART_EN & UART_IDLE_CON & UART_DIS_WAKE & UART_EN_LOOPBACK & UART_EN_ABAUD & UART_NO_PAR_8BIT & UART_1STOPBIT; U1STAv alue = 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, U1STAv alue, baud); </pre>

ReadUART1 ReadUART2

Description:	This function returns the content of UART receive buffer (UxRXREG) register.
Include:	uart.h
Prototype:	unsigned int ReadUART1(void); unsigned int ReadUART2(void);

ReadUART1 (Continued) ReadUART2

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:	ReadUART1.c ReadUART2.c
Code Example:	<pre>unsigned int RX_data; RX_data = ReadUART1();</pre>

WriteUART1 WriteUART2

Description:	This function writes data to be transmitted into the transmit buffer (UxTXREG) register.
Include:	uart.h
Prototype:	<pre>void WriteUART1(unsigned int data); void WriteUART2(unsigned int data);</pre>
Arguments:	<i>data</i> 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:	WriteUART1.c WriteUART2.c
Code Example:	<pre>WriteUART1(0xFF);</pre>

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:	<pre>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);</pre>
Arguments:	<i>length</i> This is the length of the string to be received. <i>buffer</i> This is the pointer to the location where the data received have to be stored.

getsUART1 (Continued) getsUART2

uart_data_wait This is the timeout count for which the module has to wait before return.
If the timeout count is 'N', the actual timeout 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 '0', it indicates that the complete string has been received.
If the return value is nonzero, it indicates that the complete string has not been received.

Remarks: None

Source File: getsUART1.c
getsUART2.c

Code Example: 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: 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 till data gets transmitted and then writes the next data into the transmit register.

Source File: putsUART1.c
putsUART2.c

Code Example: putsUART1(Txdata_loc);

getcUART1 getcUART2

Description: This function is identical to ReadUART1 and ReadUART2.

Source File: #define to ReadUART1 and ReadUART2 in uart.h

putcUART1 putcUART2

Description: This function is identical to WriteUART1 and WriteUART2.

Source File: #define to WriteUART1 and WriteUART2 in uart.h

3.13.2 Individual Macros

EnableIntU1RX EnableIntU2RX

Description: This macro enables the UART receive interrupt.

Include: `uart.h`

Arguments: None

Remarks: This macro sets uart receive interrupt enable bit of interrupt enable control register.

Code Example: `EnableIntU2RX;`

EnableIntU1TX EnableIntU2TX

Description: This macro enables the UART transmit interrupt.

Include: `uart.h`

Arguments: None

Remarks: This macro sets uart transmit interrupt enable bit of interrupt enable control register.

Code Example: `EnableIntU2TX;`

DisableIntU1RX DisableIntU2RX

Description: This macro disables the UART receive interrupt.

Include: `uart.h`

Arguments: None

Remarks: This macro clears uart receive interrupt enable bit of interrupt enable control register.

Code Example: `DisableIntU1RX;`

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*

SetPriorityIntU1RX (Continued)

SetPriorityIntU2RX

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);

3.13.3 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','D','2','\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 UART1module */
    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_EN_LOOPBACK &
        UART_EN_ABAUD & UART_NO_PAR_8BIT &
        UART_1STOPBIT;
    U1STAvale = 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, U1STAvale, 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;
}
```

3.14 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.

3.14.1 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

BufferEmptyDCI (Continued)

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:	<pre>while(!BufferEmptyDCI());</pre>

ConfigIntDCI

Description:	This function configures the DCI interrupt.
Include:	dci.h
Prototype:	<pre>void ConfigIntDCI(unsigned int config);</pre>
Arguments:	<i>config</i> DCI interrupt priority and enable/disable information as defined below: <u>DCI Interrupt enable/disable</u> DCI_INT_ON DCI_INT_OFF <u>DCI Interrupt priority</u> 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:	<pre>ConfigIntDCI(DCI_INT_PRI_6 & DCI_INT_ENABLE);</pre>

DataRdyDCI

Description:	This function returns the status of DCI receive buffers.
Include:	dci.h
Prototype:	<pre>char DataRdyDCI(void);</pre>
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:	<pre>while(!DataRdyDCI());</pre>

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:

Module On/Off

DCI_EN

DCI_DIS

Idle mode operation

DCI_IDLE_CON

DCI_IDLE_STOP

DCI Loopback mode enable

DCI_DIGI_LPBACK_EN

DCI_DIGI_LPBACK_DIS

CKSK pin direction select

DCI_SCKD_INP

DCI_SCKD_OUP

DCI sampling edge selection

DCI_SAMP_CLK_RIS

DCI_SAMP_CLK_FAL

FS pin direction select

DCI_FSD_INP

DCI_FSD_OUP

data to be transmitted during underflow

DCI_TX_LASTVAL_UNF

DCI_TX_ZERO_UNF

SDO pin status during transmit disable

DCI_SDO_TRISTAT

DCI_SDO_ZERO

Data justification control

DCI_DJST_ON

DCI_DJST_OFF

Frame sync mode select

DCI_FSM_ACLINK_20BIT

DCI_FSM_ACLINK_16BIT

DCI_FSM_I2S

DCI_FSM_MULTI

`config2` This contains the parameters to be configured in the DCICON2 register as defined below:

Buffer length

DCI_BUFF_LEN_4

DCI_BUFF_LEN_3

DCI_BUFF_LEN_2

DCI_BUFF_LEN_1

OpenDCI (Continued)

	<u>DCI Frame sync generator control</u> DCI_FRAME_LEN_16 DCI_FRAME_LEN_15 DCI_FRAME_LEN_14 DCI_FRAME_LEN_1 <u>DCI data word size</u> DCI_DATA_WORD_16 DCI_DATA_WORD_15 DCI_DATA_WORD_14 DCI_DATA_WORD_5 DCI_DATA_WORD_4
<i>config3</i>	This contains the bit clock generator value to be configured in the DCICON3 register.
<i>trans_mask / recv_mask</i>	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

OpenDCI (Continued)

Code Example:

```
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);
```

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*.

WriteDCI (Continued)

Source File: WriteDCI.c
Code Example: unsigned int DCI_tx0 = 0x60;
WriteDCI(DCI_tx0, 0);

3.14.2 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);

3.14.3 Example of Use

```
#define __dsPIC30F6014__
#include<p30fxxx.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 framelength, wordsize and buffer length */
```

```
    unsigned int DCICON2value;
/* Holds the information reagarding bit clock
generator */
    unsigned int DCICON3value ;
/* Holds the information reagarding data to be received
or ignored during this time slot */
    unsigned int RSCONvalue ;
/* Holds the information reagarding 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_FAL &
                  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;
}
```

3.15 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.

3.15.1 Individual Functions

ConfigIntSPI1 ConfigIntSPI2

Description:	This function configures the SPI Interrupt.
Include:	<code>spi.h</code>
Prototype:	<pre>void ConfigIntSPI1(unsigned int config); void ConfigIntSPI2(unsigned int config);</pre>
Arguments:	<p><i>config</i> SPI interrupt priority and enable/disable information as defined below:</p> <p><u>Interrupt enable/disable</u></p> <pre>SPI_INT_EN SPI_INT_DIS</pre> <p><u>Interrupt Priority</u></p> <pre>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</pre>
Return Value:	None
Remarks:	This function clears the Interrupt flag bit, sets the interrupt priority and enables/disables the interrupt.
Source File:	<pre>ConfigIntSPI1.c ConfigIntSPI2.c</pre>
Code Example:	<code>ConfigIntSPI1(SPI_INT_PRI_3 & SPI_INT_EN);</code>

CloseSPI1 CloseSPI2

Description:	This function turns off the SPI module
Include:	<code>spi.h</code>
Prototype:	<pre>void CloseSPI1(void); void CloseSPI2(void);</pre>
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:	<pre>CloseSPI1.c CloseSPI2.c</pre>
Code Example:	<code>CloseSPI1();</code>

DataRdySPI1 DataRdySPI2

Description:	This function determines if the SPI buffer contains any data to be read.
Include:	<code>spi.h</code>
Prototype:	<code>char DataRdySPI1(void);</code> <code>char DataRdySPI2(void);</code>
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:	<code>DataRdySPI1.c</code> <code>DataRdySPI2.c</code>
Code Example:	<code>while(DataRdySPI1());</code>

ReadSPI1 ReadSPI2

Description:	This function reads the content of the SPI receive buffer (SPIxBUF) register.
Include:	<code>spi.h</code>
Prototype:	<code>unsigned int ReadSPI1(void);</code> <code>unsigned int ReadSPI2(void);</code>
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 SPIxRBF register is returned. If 8-bit communication is enabled, then the lowerbyte 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:	<code>ReadSPI1.c</code> <code>ReadSPI2.c</code>
Code Example:	<code>unsigned int RX_data;</code> <code>RX_data = ReadSPI1();</code>

WriteSPI1 WriteSPI2

Description:	This function writes the data to be transmitted into the transmit buffer (SPIxBUF) register.
Include:	<code>spi.h</code>

WriteSPI1 (Continued)

WriteSPI2

Prototype:	<code>void WriteSPI1(unsigned int data);</code> <code>void WriteSPI2(unsigned int data);</code>
Arguments:	<i>data</i> This is the data to be transmitted which will be stored in SPI buffer.
Remarks:	This function writes the data (byte/word) to be transmitted into the transmit buffer. If 16-bit communication is enabled, the 16-bit value is written to the transmit buffer. If 8-bit communication is enabled, then upper byte is masked and then written to the transmit buffer.
Return Value:	None
Source File:	<code>WriteSPI1.c</code> <code>WriteSPI2.c</code>
Code Example:	<code>WriteSPI1(0x3FFF);</code>

OpenSPI1

OpenSPI2

Description:	This function configures the SPI module
Include:	<code>spi.h</code>
Prototype:	<code>void OpenSPI1(unsigned int config1,</code> <code> unsigned int config2);</code> <code>void OpenSPI2(unsigned int config1,</code> <code> unsigned int config2);</code>
Arguments:	<i>config1</i> This contains the parameters to be configured in the SPIxCON register as defined below: <u>Framed SPI support Enable/Disable</u> <code>FRAME_ENABLE_ON</code> <code>FRAME_ENABLE_OFF</code> <u>Frame Sync Pulse direction control</u> <code>FRAME_SYNC_INPUT</code> <code>FRAME_SYNC_OUTPUT</code> <u>SDO pin control bit</u> <code>DISABLE_SDO_PIN</code> <code>ENABLE_SDO_PIN</code> <u>Word/Byte communication mode</u> <code>SPI_MODEL6_ON</code> <code>SPI_MODEL6_OFF</code> <u>SPI Data Input Sample phase</u> <code>SPI_SMP_ON</code> <code>SPI_SMP_OFF</code> <u>SPI Clock Edge Select</u> <code>SPI_CKE_ON</code> <code>SPI_CKE_OFF</code> <u>SPI slave select enable</u> <code>SLAVE_SELECT_ENABLE_ON</code> <code>SLAVE_SELECT_ENABLE_OFF</code> <u>SPI Clock polarity select</u> <code>CLK_POL_ACTIVE_LOW</code> <code>CLK_POL_ACTIVE_HIGH</code>

OpenSPI1 (Continued) OpenSPI2

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 SPIx-STAT 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_OVERFLOW_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_OVERFLOW_CLR OpenSPI1(config1,
config2);
    
```

putsSPI1 putsSPI2

Description:	This function writes a string of data to be transmitted into the SPI transmit buffer.
Include:	<code>spi.h</code>
Prototype:	<pre>void putsSPI1(unsigned int length, unsigned int *wrptr); void putsSPI2(unsigned int length, unsigned int *wrptr);</pre>
Arguments:	<p><i>length</i> This is the number of data words/bytes to be transmitted.</p> <p><i>wrptr</i> This is the pointer to the string of data to be transmitted.</p>
Return Value:	None
Remarks:	<p>This function writes the specified length of data words/bytes to be transmitted into the transmit buffer.</p> <p>Once the transmit buffer is full, it waits till the data gets transmitted and then writes the next data into the transmit register.</p> <p>The control remains in this function if SPI module is disabled while SPITBF bit is set.</p>
Source File:	<code>putsSPI1.c</code> <code>putsSPI2.c</code>
Code Example:	<code>putsSPI1(10,Txdata_loc);</code>

getsSPI1 getsSPI2

Description:	This function reads a string of data of specified length and stores it into the location specified.
Include:	<code>spi.h</code>
Prototype:	<pre>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);</pre>
Arguments:	<p><i>length</i> This is the length of the string to be received.</p> <p><i>rdptr</i> This is the pointer to the location where the data received have to be stored.</p> <p><i>spi_data_wait</i> This is the timeout count for which the module has to wait before return. If the timeout count is 'N', the actual timeout would be about (19*N - 1) instruction cycles.</p>
Return Value:	<p>This function returns the number of bytes yet to be received.</p> <p>If the return value is a '0', it indicates that the complete string has been received.</p> <p>If the return value is a nonzero, it indicates that the complete string has not been received.</p>
Remarks:	None

getsSPI1 (Continued)

getsSPI2

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

3.15.2 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);

3.15.3 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_OVERFLOW_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_OVERFLOW_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;
}
```

3.16 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.

3.16.1 Individual Functions

CloseQEI

Description:	This function turns off the QEI module
Include:	qe1.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:	qe1.h
Prototype:	void ConfigIntQEI(unsigned int <i>config</i>);
Arguments:	<i>config</i> QEI interrupt priority and enable/disable information as defined below: <u>QEI Interrupt enable/disable</u> QE1_INT_ENABLE QE1_INT_DISABLE

ConfigIntQEI (Continued)

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

OpenQEI (Continued)

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

Idle Mode Operation

QEI_IDLE_STOP
QEI_IDLE_CON

config2 This contains the parameters to be configured in the DFLTxCN 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

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

OpenQEI (Continued)

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();

WriteQEI

Description: This function sets the maximum count value for QEI.
Include: qei.h
Prototype: 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: unsigned int position = 0x3FFF;
 WriteQEI(position);

3.16.2 Individual Macros

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: 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: DisableIntQEI;

SetPriorityIntQEI

Description:	This macro sets priority for QEI interrupt.
Include:	qe1.h
Arguments:	<i>priority</i>
Remarks:	This macro sets QEI interrupt priority bits of interrupt priority control register.
Code Example:	SetPriorityIntQEI(7);

3.16.3 Example of Use

```
#define __dsPIC30F6010__
#include <p30fxxx.h>
#include <qe1.h>
unsigned int pos_value;

void __attribute__((__interrupt__)) _QEInterrupt(void)
{
    PORTDbits.RD1 = 1;    /* turn off LED on RD1 */
    POSCNT = 0;
    IFS2bits.QE1IF = 0;   /* Clear QE1 interrupt flag */
}

int main(void)
{
    unsigned int max_value;
    TRISDbits.TRISD1 = 0;
    PORTDbits.RD1 = 1;    /* turn off LED on RD1 */

    /* Enable QE1 Interrupt and Priority to "1" */
    ConfigIntQE1(QE1_INT_PRI_1 & QE1_INT_ENABLE);

    POSCNT = 0;
    MAXCNT = 0xFFFF;
    OpenQE1(QE1_INT_CLK & QE1_INDEX_RESET_ENABLE &
            QE1_CLK_PRESCALE_256 &
            QE1_GATED_ACC_DISABLE & QE1_INPUTS_NOSWAP &
            QE1_MODE_TIMER & QE1_DIR_SEL_CNTRL &
            QE1_IDLE_CON, 0);
    QEICONbits.UPDN = 1;
    while(1)
    {
        pos_value = ReadQE1();
        if(pos_value >= 0x7FFF)
        {
            PORTDbits.RD1 = 0; /* turn on LED on RD1 */
        }
    }
    CloseQE1();
}
```

3.17 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.

3.17.1 Individual Functions

CloseMCPWM

Description:	This function turns off the Motor Control PWM module
Include:	<code>pwm.h</code>
Prototype:	<code>void closeMCPWM(void);</code>
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:	<code>CloseMCPWM.c</code>
Code Example:	<code>CloseMCPWM();</code>

ConfigIntMCPWM

Description:	This function configures the PWM Interrupts.
Include:	<code>pwm.h</code>
Prototype:	<code>void ConfigIntMCPWM(unsigned int config);</code>
Arguments:	<div> <i>config</i> PWM interrupt priority and enable/disable information as defined below: </div> <div> <u>PWM Interrupt enable/disable</u> PWM_INT_EN PWM_INT_DIS <u>PWM Interrupt priority</u> PWM_INT_PR0 PWM_INT_PR1 PWM_INT_PR2 PWM_INT_PR3 PWM_INT_PR4 PWM_INT_PR5 PWM_INT_PR6 PWM_INT_PR7 <u>Fault A Interrupt enable/disable</u> PWM_FLTA_EN_INT PWM_FLTA_DIS_INT <u>Fault A Interrupt priority</u> 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 </div>

ConfigIntMCPWM (Continued)

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

OpenMCPWM (Continued)

Input prescaler select

PWM_IPCLK_SCALE1
 PWM_IPCLK_SCALE4
 PWM_IPCLK_SCALE16
 PWM_IPCLK_SCALE64

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 synchronisation select

PWM_OSYNC_PWM
 PWM_OSYNC_Tcy

PWM update enable/disable

PWM_UDIS
 PWM_UEN

Return Value None

OpenMCPWM (Continued)

Remarks: This function configures the PTPER, SEVTCMP, PTCON, PWMCON1 and PWMCON2 registers.

Source File: OpenMCPWM.c

Code Example:

```
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);
```

OverrideMCPWM

Description: This function configures the OVDCON register.

Include: pwm.h

Prototype: void OverrideMCPWM(unsigned int *config*);

Arguments: *config* This contains the parameters to be configured in the OVDCON register as defined below:

Output controlled by PWM generator

PWM_GEN_4H
PWM_GEN_3H
PWM_GEN_2H
PWM_GEN_1H
PWM_GEN_4L
PWM_GEN_3L
PWM_GEN_2L
PWM_GEN_1L

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

PWM_POUT_4H
PWM_POUT_4L
PWM_POUT_3H
PWM_POUT_3L
PWM_POUT_2H
PWM_POUT_2L
PWM_POUT_1H
PWM_POUT_1L

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 functions 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 dutycycle 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 = 0xFFFF;
updatedisable = 0;
SetDCMCPWM(dutycyclereg, dutycycle, updatedisable);
  
```

SetMCPWMDeadTimeAssignment

Description:	This function configures the assignment of dead time units to PWM output pairs.
Include:	<code>pwm.h</code>
Prototype:	<pre>void SetMCPWMDeadTimeAssignment (unsigned int config);</pre>
Arguments:	<p><i>config</i> This contains the parameters to be configured in the DTCON2 register as defined below:</p> <p><u>Dead Time Select bits for PWM4 signal</u> PWM_DTS4A_UA PWM_DTS4A_UB PWM_DTS4I_UA PWM_DTS4I_UB Bit defines related to PWM4 is available only for certain devices and the suitable data sheet should be referred to.</p> <p><u>Dead Time Select bits for PWM3 signal</u> PWM_DTS3A_UA PWM_DTS3A_UB PWM_DTS3I_UA PWM_DTS3I_UB</p> <p><u>Dead Time Select bits for PWM2 signal</u> PWM_DTS2A_UA PWM_DTS2A_UB PWM_DTS2I_UA PWM_DTS2I_UB</p> <p><u>Dead Time Select bits for PWM1 signal</u> PWM_DTS1A_UA PWM_DTS1A_UB PWM_DTS1I_UA PWM_DTS1I_UB</p>
Return Value	None
Remarks:	None
Source File:	<code>SetMCPWMDeadTimeAssignment.c</code>
Code Example:	<pre>SetMCPWMDeadTimeAssignment (PWM_DTS3A_UA & PWM_DTS2I_UA & PWM_DTS1I_UA);</pre>

SetMCPWMDeadTimeGeneration

Description:	This function configures dead time values and clock prescalers.
Include:	<code>pwm.h</code>
Prototype:	<pre>void SetMCPWMDeadTimeGeneration(unsigned int config);</pre>
Arguments:	<p><i>config</i> This contains the parameters to be configured in the DTCON1 register as defined below:</p> <p><u>Dead Time Unit B Prescale Select bits</u> PWM_DTBPS8 PWM_DTBPS4 PWM_DTBPS2 PWM_DTBPS1</p>

SetMCPWMDeadTimeGeneration (Continued)

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: 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.

SetMCPWMFaultA (Continued)

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 InputB Enable bits of PWM.

Include: pwm.h

Prototype: 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

SetMCPWMFaultB (Continued)

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: SetMCPWMFaultB(PWM_OVB3L_INACTIVE &
 PWM_FLTB_MODE_LATCH &
 PWM_FLTB2_DIS);

3.17.2 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: 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: 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: SetPriorityIntMCPWM(7);

EnableIntFLTA

Description: This macro enables the FLTA interrupt.
Include: pwm.h

EnableIntFLTA (Continued)

Arguments: None

Remarks: This macro sets FLTA interrupt enable bit of interrupt enable control register.

Code Example: `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: `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*

SetPriorityIntFLTB (Continued)

Remarks: This macro sets FLTB interrupt priority bits of interrupt priority control register.

Code Example: SetPriorityIntFLTB(1);

3.17.3 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 'duty cyclereg' determines the duty cycle
    register(PDCx) to be written */
    unsigned int duty cyclereg;
    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_PRO
        & PWM_FLTB_DIS_INT & PWM_FLTB_INT_PRO);
    ConfigIntMCPWM( config );
    /* Configure PWM to generate square wave of 50% duty cycle */
    duty cyclereg = 1;
    dutycycle = 0x3FFF;
    updatedisable = 0;

    SetDCMCPWM(duty cyclereg,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);
}
```

3.18 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.

3.18.1 Individual Functions

CloseI2C

Description:	This function turns off the I2C module
Include:	<code>i2c.h</code>
Prototype:	<code>void CloseI2C(void);</code>
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:	<code>CloseI2C.c</code>
Code Example:	<code>CloseI2C();</code>

ConfigIntI2C

Description:	This function Configure the I2C Interrupt.
Include:	<code>i2c.h</code>
Prototype:	<code>void ConfigIntI2C(unsigned int config);</code>
Arguments:	<div><i>config</i> I2C interrupt priority and enable/disable information as defined below: <u>I2C master Interrupt enable/disable</u> <code>MI2C_INT_ON</code> <code>MI2C_INT_OFF</code> I2C slave Interrupt enable/disable <code>SI2C_INT_ON</code> <code>SI2C_INT_OFF</code> <u>I2C master Interrupt priority</u> <code>MI2C_INT_PRI_7</code> <code>MI2C_INT_PRI_6</code> <code>MI2C_INT_PRI_5</code> <code>MI2C_INT_PRI_4</code> <code>MI2C_INT_PRI_3</code> <code>MI2C_INT_PRI_2</code> <code>MI2C_INT_PRI_1</code> <code>MI2C_INT_PRI_0</code> <u>I2C slave Interrupt priority</u> <code>SI2C_INT_PRI_7</code> <code>SI2C_INT_PRI_6</code> <code>SI2C_INT_PRI_5</code> <code>SI2C_INT_PRI_4</code> <code>SI2C_INT_PRI_3</code> <code>SI2C_INT_PRI_2</code> <code>SI2C_INT_PRI_1</code> <code>SI2C_INT_PRI_0</code></div>
Return Value	None
Remarks:	This function clears the Interrupt flag bits, sets the interrupt priorities of master and slave and enables/disables the interrupt.

ConfigIntI2C (Continued)

Source File: ConfigIntI2C.c
Code Example: ConfigIntI2C(MI2C_INT_ON & MI2C_INT_PRI_3
& SI2C_INT_ON & SI2C_INT_PRI_5);

AckI2C

Description: Generates I2C bus Acknowledge condition.
Include: i2c.h
Prototype: void AckI2C(void);
Arguments: None
Return Value: None
Remarks: This function generates an I2C 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 I2C 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 I2C control register and Transmit status bit I2C Status register are clear. The IdleI2C function is required since the hardware I2C peripheral does not allow for spooling of bus sequence. The I2C peripheral must be in idle state before an I2C 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:	<code>i2c.h</code>						
Prototype:	<code>unsigned int MastergetsI2C(unsigned int <i>length</i>, unsigned char *<i>rdptr</i>, unsigned int <i>i2c_data_wait</i>);</code>						
Arguments:	<table><tr><td><i>length</i></td><td>Number of bytes to read from I2C device.</td></tr><tr><td><i>rdptr</i></td><td>Character type pointer to dsPIC Ram for Storage of data read from I2C device</td></tr><tr><td><i>i2c_data_wait</i></td><td>This is the timeout count for which the module has to wait before return. If the timeout count is 'N', the actual timeout would be about (20*N - 1) instruction cycles.</td></tr></table>	<i>length</i>	Number of bytes to read from I2C device.	<i>rdptr</i>	Character type pointer to dsPIC Ram for Storage of data read from I2C device	<i>i2c_data_wait</i>	This is the timeout count for which the module has to wait before return. If the timeout count is 'N', the actual timeout would be about (20*N - 1) instruction cycles.
<i>length</i>	Number of bytes to read from I2C device.						
<i>rdptr</i>	Character type pointer to dsPIC Ram for Storage of data read from I2C device						
<i>i2c_data_wait</i>	This is the timeout count for which the module has to wait before return. If the timeout count is 'N', the actual timeout 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 <i>i2c_data_wait</i> time out value						
Remarks:	This routine reads a predefined data string from the I2C bus.						
Source File:	<code>MastergetsI2C.c</code>						
Code Example:	<pre>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);</pre>						

MasterputsI2C

Description:	This function is used to write out a data string to the I2C bus.		
Include:	<code>i2c.h</code>		
Prototype:	<code>unsigned int MasterputsI2C(unsigned char *<i>wrptr</i>);</code>		
Arguments:	<table><tr><td><i>wrptr</i></td><td>Character type pointer to data objects in dsPIC Ram. The data objects are written to the I2C device.</td></tr></table>	<i>wrptr</i>	Character type pointer to data objects in dsPIC Ram. The data objects are written to the I2C device.
<i>wrptr</i>	Character type pointer to data objects in dsPIC Ram. The data objects are written to the I2C device.		
Return Value	This function returns -3 if a write collison occurred.This function returns '0' if the null characer 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 <code>MasterputcI2C</code> function. The actual called function body is termed <code>MasterWriteI2C</code> . <code>MasterWriteI2C</code> and <code>MasterputcI2C</code> refer to the same function via a <code>#define</code> statement in the <code>i2c.h</code>		
Source File:	<code>MasterputsI2C.c</code>		
Code Example:	<pre>unsigned char string[] = " MICROCHIP "; unsigned char *wrptr; wrptr = string; MasterputsI2C(wrptr);</pre>		

MasterReadI2C

Description:	This function is used to read a single byte from I2C bus
Include:	<code>i2c.h</code>

MasterReadI2C (Continued)

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: `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`

OpenI2C (Continued)

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

Return Value

None

Remarks:

This function configures the I2C control register and I2C Baud rate generator register.

Source File:

OpenI2C.c

Code Example:

OpenI2C();

RestartI2C

Description:

Generates I2C bus restart condition.

Include:

i2c.h

Prototype:

void RestartI2C(void);

RestartI2C (Continued)

Arguments:	None
Return Value	None
Remarks:	This function generates an I2C bus Restart condition.
Source File:	RestartI2C.c
Code Example:	RestartI2C();

SlavegetsI2C

Description:	This function reads pre-determined data string length from the I2C bus.				
Include:	i2c.h				
Prototype:	<pre>unsigned int SlavegetsI2C(unsigned char *rdptr, unsigned int i2c_data_wait);</pre>				
Arguments:	<table><tr><td><i>rdptr</i></td><td>Character type pointer to dsPIC Ram for Storage of data read from I2C device</td></tr><tr><td><i>i2c_data_wait</i></td><td>This is the timeout count for which the module has to wait before return. If the timeout count is 'N', the actual timeout would be about (20*N - 1) instruction cycles.</td></tr></table>	<i>rdptr</i>	Character type pointer to dsPIC Ram for Storage of data read from I2C device	<i>i2c_data_wait</i>	This is the timeout count for which the module has to wait before return. If the timeout count is 'N', the actual timeout would be about (20*N - 1) instruction cycles.
<i>rdptr</i>	Character type pointer to dsPIC Ram for Storage of data read from I2C device				
<i>i2c_data_wait</i>	This is the timeout count for which the module has to wait before return. If the timeout count is 'N', the actual timeout would be about (20*N - 1) instruction cycles.				
Return Value	Returns the number of bytes received from the I2C bus.				
Remarks:	This routine reads a predefined data string from the I2C bus.				
Source File:	SlavegetsI2C.c				
Code Example:	<pre>unsigned char string[12]; unsigned char *rdptr; rdptr = string; i2c_data_out = 0x11; SlavegetsI2C(rdptr, i2c_data_wait);</pre>				

SlaveputsI2C

Description:	This function is used to write out a data string to the I2C bus.		
Include:	i2c.h		
Prototype:	<pre>unsigned int SlaveputsI2C(unsigned char *wrptr);</pre>		
Arguments:	<table><tr><td><i>wrptr</i></td><td>Character type pointer to data objects in dsPIC Ram. The data objects are written to the I2C device.</td></tr></table>	<i>wrptr</i>	Character type pointer to data objects in dsPIC Ram. The data objects are written to the I2C device.
<i>wrptr</i>	Character type pointer to data objects in dsPIC 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:	<pre>unsigned char string[] ="MICROCHIP"; unsigned char *rdptr; rdptr = string; SlaveputsI2C(rdptr);</pre>		

SlaveReadI2C

Description:	This function is used to read a single byte from the I2C bus.
Include:	i2c.h

SlaveReadI2C (Continued)

Prototype:	<code>unsigned char SlaveReadI2C(void);</code>
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 <code>SlavegetcI2C</code> .
Source File:	<code>SlaveReadI2C.c</code>
Code Example:	<pre>unsigned char value; value = SlaveReadI2C();</pre>

SlaveWriteI2C

Description:	This function is used to write out a single byte to the I2C bus.
Include:	<code>i2c.h</code>
Prototype:	<code>void SlaveWriteI2C(unsigned char data_out);</code>
Arguments:	<code>data_out</code> 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 <code>SlaveputcI2C</code> .
Source File:	<code>SlaveWriteI2C.c</code>
Code Example:	<code>SlaveWriteI2C('a');</code>

StartI2C

Description:	Generates I2C bus start condition.
Include:	<code>i2c.h</code>
Prototype:	<code>void StartI2C(void);</code>
Arguments:	None
Return Value	None
Remarks:	This function generates a I2C bus start condition.
Source File:	<code>StartI2C.c</code>
Code Example:	<code>StartI2C();</code>

StopI2C

Description:	Generates I2C bus stop condition.
Include:	<code>i2c.h</code>
Prototype:	<code>void StopI2C(void);</code>
Arguments:	None
Return Value	None
Remarks:	This function generates a I2C bus stop condition.
Source File:	<code>StopI2C.c</code>
Code Example:	<code>StopI2C();</code>

3.18.2 Individual Macros

EnableIntMI2C

Description:	This macro enables the master I2C interrupt.
Include:	<code>i2c.h</code>
Arguments:	None
Remarks:	This macro sets master I2C enable bit of interrupt enable control register.
Code Example:	<code>EnableIntMI2C;</code>

DisableIntMI2C

Description:	This macro disables the master I2C interrupt.
Include:	<code>i2c.h</code>
Arguments:	None
Remarks:	This macro clears master I2C interrupt enable bit of interrupt enable control register.
Code Example:	<code>DisableIntMI2C;</code>

SetPriorityIntMI2C

Description:	This macro sets priority for master I2C interrupt.
Include:	<code>i2c.h</code>
Arguments:	<i>priority</i>
Remarks:	This macro sets master I2C interrupt priority bits of interrupt priority control register.
Code Example:	<code>SetPriorityIntMI2C(1);</code>

EnableIntSI2C

Description:	This macro enables the slave I2C interrupt.
Include:	<code>i2c.h</code>
Arguments:	None
Remarks:	This macro sets slave I2C enable bit of interrupt enable control register.
Code Example:	<code>EnableIntSI2C;</code>

DisableIntSI2C

Description:	This macro disables the slave I2C interrupt.
Include:	<code>i2c.h</code>
Arguments:	None
Remarks:	This macro clears slave I2C interrupt enable bit of interrupt enable control register.
Code Example:	<code>DisableIntSI2C;</code>

SetPriorityIntSI2C

Description:	This macro sets priority for master I2C interrupt.
Include:	i2c.h
Arguments:	<i>priority</i>
Remarks:	This macro sets master I2C interrupt priority bits of interrupt priority control register.
Code Example:	SetPriorityIntSI2C(4);

3.18.3 Example of Use

```
#define __dsPIC30F6014__
#include <p30fxxx.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.a` and `libm.a` (math functions).

Additionally, some dsPIC standard C library helper functions, and standard functions that must be modified for use with dsPIC devices, are in the library `libpic30.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:\pic30_tools`) contains the following subdirectories with library-related files:

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

4.2 HIGHLIGHTS

This chapter is organized as follows:

- Using the Standard C Libraries

libc.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.a

- `<math.h>` mathematical functions

libpic30.a

- `pic30-libs`

4.3 USING THE STANDARD C LIBRARIES

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

4.3.1 Header Files

All standard C library entities are declared or defined in one or more standard headers (See list in **Section 4.2 “Highlights”**.) 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.

Every function in the library is declared in a standard header.

4.3.2 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.a`, `libm.a`, and `libpic30.a`.

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* and *MPLAB C30 C Compiler User's Guide* for more information on the heap.

4.4 <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 <code>stderr</code> and the program is aborted.
Include:	<code><assert.h></code>
Prototype:	<code>void assert(int expression);</code>
Argument:	<code>expression</code> The expression to test.

assert (Continued)

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:

```
#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
```

4.5 <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: `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 a zero.

Remarks: Alphanumeric characters are included within the ranges A - Z, a - z, or 0 - 9.

isalnum (Continued)

Example:

```
#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: 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.

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

isctrl

Description:	Test for a control character.
Include:	<ctype.h>
Prototype:	int isctrl(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:	<pre>#include <ctype.h> /* for isctrl */ #include <stdio.h> /* for printf */ int main(void) { char ch; ch = 'B'; if (isctrl(ch)) printf("B is a control character\n"); else printf("B is NOT a control character\n"); ch = '\t'; if (isctrl(ch)) printf("A tab is a control character\n"); else printf("A tab is NOT a control character\n"); }</pre> <p>Output: B is NOT a control character A tab is a control character</p>

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'.

isdigit (Continued)

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.

isgraph (Continued)

Example:

```
#include <ctype.h> /* for isgraph */
#include <stdio.h> /* for printf */

int main(void)
{
    int ch;

    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'.

islower (Continued)

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:

```
#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");
}
```

Standard C Libraries with Math Functions

isprint (Continued)

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: #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').

isspace (Continued)

Example:

```
#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: 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:

```
#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");
}
```

isupper (Continued)

Output:

B is upper case

b is NOT upper case

isxdigit

Description: Test for a hexadecimal digit.**Include:** <ctype.h>**Prototype:** 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:

```
#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.

tolower (Continued)

Example:

```
#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

toupper

Description: Convert a character to an upper case alphabetical character.

Include: <ctype.h>

Prototype: 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:

```
#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));
}
```

toupper (Continued)

Output:

b changes to upper case B

B remains upper case B

@ has no upper case, so @ is returned

4.6 <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 startup, `errno` is zero. Library functions will never set `errno` to zero.

EDOM

Description: Represents a domain error.**Include:** `<errno.h>`**Remarks:** EDOM represents a domain error, which occurs when an input argument is outside the domain in which the function is defined.

ERANGE

Description: Represents an overflow or underflow error.**Include:** `<errno.h>`**Remarks:** ERANGE represents an overflow or underflow error, which occurs when a result is too large or too small to be stored.

errno

Description: Contains the value of an error when an error occurs in a function.**Include:** `<errno.h>`**Remarks:** The variable `errno` is set to a non-zero integer value by a library function when an error occurs. At program startup, `errno` is set to zero. `Errno` should be reset to zero prior to calling a function that sets it.

4.7 <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.

DBL_DIG

Description: Number of decimal digits of precision in a double precision floating point value**Include:** `<float.h>`**Value:** 6 by default, 15 if the switch `-fno-short-double` is used

DBL_DIG (Continued)

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_EPSILON

Description: The difference between 1.0 and the next larger representable double precision floating point value

Include: `<float.h>`

Value: 1.192093e-07 by default, 2.220446e-16 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_MANT_DIG

Description: Number of base-`FLT_RADIX` digits in a double precision floating-point significand

Include: `<float.h>`

Value: 24 by default, 53 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

Description: Maximum finite double precision floating point value

Include: `<float.h>`

Value: 3.402823e+38 by default, 1.797693e+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_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>`

Standard C Libraries with Math Functions

DBL_MAX_EXP (Continued)

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.

DBL_MIN_EXP

Description: Minimum negative integer value for a double precision floating point exponent in base `FLT_RADIX`

Include: `<float.h>`

Value: -125 by default, -1021 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.

FLT_DIG

Description: Number of decimal digits of precision in a single precision floating point value

Include: `<float.h>`

Value: 6

FLT_EPSILON

Description: The difference between 1.0 and the next larger representable single precision floating point value

Include: `<float.h>`

Value: 1.192093e-07

FLT_MANT_DIG

Description: Number of base-FLT_RADIX digits in a single precision floating-point significand
Include: <float.h>
Value: 24

FLT_MAX

Description: Maximum finite single precision floating point value
Include: <float.h>
Value: 3.402823e+38

FLT_MAX_10_EXP

Description: Maximum integer value for a single precision floating point exponent in base 10
Include: <float.h>
Value: 38

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

Standard C Libraries with Math Functions

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

LDBL_MIN_EXP

Description: Minimum negative integer value for a long double precision floating point exponent in base `FLT_RADIX`
Include: `<float.h>`
Value: -1021

4.8 <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.

CHAR_BIT

Description: Number of bits to represent type `char`
Include: `<limits.h>`
Value: 8

CHAR_MAX

Description: Maximum value of a `char`
Include: `<limits.h>`
Value: 127

CHAR_MIN

Description: Minimum value of a `char`
Include: `<limits.h>`

Standard C Libraries with Math Functions

CHAR_MIN (Continued)

Value: -128

INT_MAX

Description: Maximum value of an `int`

Include: `<limits.h>`

Value: 32767

INT_MIN

Description: Minimum value of an `int`

Include: `<limits.h>`

Value: -32768

LLONG_MAX

Description: Maximum value of a `long int`

Include: `<limits.h>`

Value: 9223372036854775807

LLONG_MIN

Description: Minimum value of a `long int`

Include: `<limits.h>`

Value: -9223372036854775808

LONG_MAX

Description: Maximum value of a `long int`

Include: `<limits.h>`

Value: 2147483647

LONG_MIN

Description: Minimum value of a `long int`

Include: `<limits.h>`

Value: -2147483648

MB_LEN_MAX

Description: Maximum number of bytes in a multibyte character

Include: `<limits.h>`

Value: 1

SCHAR_MAX

Description: Maximum value of a signed char
Include: <limits.h>
Value: 127

SCHAR_MIN

Description: Minimum value of a signed char
Include: <limits.h>
Value: -128

SHRT_MAX

Description: Maximum value of a short int
Include: <limits.h>
Value: 32767

SHRT_MIN

Description: Minimum value of a short int
Include: <limits.h>
Value: -32768

UCHAR_MAX

Description: Maximum value of an unsigned char
Include: <limits.h>
Value: 255

UINT_MAX

Description: Maximum value of an unsigned int
Include: <limits.h>
Value: 65535

ULLONG_MAX

Description: Maximum value of a long unsigned int
Include: <limits.h>
Value: 18446744073709551615

ULONG_MAX

Description: Maximum value of a long unsigned int
Include: <limits.h>
Value: 4294967295

USHRT_MAX

Description:	Maximum value of an unsigned short int
Include:	<limits.h>
Value:	65535

4.9 <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

Description:	A type that is an array used by <code>setjmp</code> and <code>longjmp</code> to save and restore the program environment.
Include:	<setjmp.h>
Prototype:	<code>typedef int jmp_buf[_NSETJMP];</code>
Remarks:	<code>_NSETJMP</code> is defined as <code>16 + 2</code> that represents 16 registers and a 32-bit return address.

setjmp

Description:	A macro that saves the current state of the program for later use by <code>longjmp</code> .
Include:	<setjmp.h>
Prototype:	<code>#define setjmp(jmp_buf env)</code>
Argument:	<code>env</code> variable where environment is stored
Return Value:	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. Note: If the argument <code>val</code> from <code>longjmp</code> is 0, <code>setjmp</code> returns 1.
Example:	See <code>longjmp</code> .

longjmp

Description:	A function that restores the environment saved by <code>setjmp</code> .
Include:	<code><setjmp.h></code>
Prototype:	<code>void longjmp(jmp_buf env, int val);</code>
Arguments:	<code>env</code> variable where environment is stored <code>val</code> value to be returned to <code>setjmp</code> call.
Remarks:	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.

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.

sig_atomic_t

Description:	A type used by a signal handler
Include:	<code><signal.h></code>
Prototype:	<code>typedef int sig_atomic_t;</code>

SIG_DFL

Description:	Used as the second argument and/or the return value for <code>signal</code> to specify that the default handler should be used for a specific signal.
Include:	<code><signal.h></code>

SIG_ERR

Description:	Used as the return value for <code>signal</code> when it cannot complete a request due to an error.
Include:	<code><signal.h></code>

SIG_IGN

Description: Used as the second argument and/or the return value for `signal` to specify that the signal should be ignored.

Include: `<signal.h>`

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

SIGFPE (Continued)

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”.

Standard C Libraries with Math Functions

SIGSEGV

Description:	Signals invalid access to storage.
Include:	<signal.h>
Prototype:	#define SIGSEGV
Remarks:	SIGSEGV is used as an argument for <code>raise</code> and/or <code>signal</code> . 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 <code>signal</code> for an example of a user defined function.
Example:	<pre>#include <signal.h> /* for raise, SIGSEGV */ #include <stdio.h> /* for printf */ int main(void) { raise(SIGSEGV); printf("Program never reaches here."); }</pre> <p>Output: SEGV</p> <p>Explanation: SEGV stands for "invalid storage access".</p>

SIGTERM

Description:	Signals a termination request
Include:	<signal.h>
Prototype:	#define SIGTERM
Remarks:	SIGTERM is used as an argument for <code>raise</code> and/or <code>signal</code> . 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 <code>signal</code> for an example of a user defined function.
Example:	<pre>#include <signal.h> /* for raise, SIGTERM */ #include <stdio.h> /* for printf */ int main(void) { raise(SIGTERM); printf("Program never reaches here."); }</pre> <p>Output: TERM</p> <p>Explanation: TERM stands for "termination request".</p>

raise

Description:	Reports a synchronous signal.
Include:	<signal.h>
Prototype:	<code>int raise(int sig);</code>
Argument:	<i>sig</i> signal name
Return Value:	Returns a 0 if successful; otherwise, returns a nonzero value.

raise (Continued)

Remarks: raise sends the signal identified by sig to the executing program.

Example:

```
#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__))
_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: <signal.h>

Prototype: void (*signal(int sig, void(*func)(int)))(int);

Arguments: sig signal name

signal (Continued)

func function to be executed

Return Value: Returns the previous value of *func*.

Example:

```
#include <signal.h> /* for signal, raise, SIGINT,
SIGILL, SIG_IGN, and SIGFPE */
#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.");
}
```

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 <code>va_list</code> declares a variable that will refer to each argument in a variable-length argument list.
Include:	<code><stdarg.h></code>
Example:	See <code>va_arg</code> .

`va_arg`

Description:	Gets the current argument
Include:	<code><stdarg.h></code>
Prototype:	<code>#define va_arg(va_list ap, Ty)</code>
Argument:	<code>ap</code> pointer to list of arguments <code>Ty</code> type of argument to be retrieved
Return Value:	Returns the current argument
Remarks:	<code>va_start</code> must be called before <code>va_arg</code> .
Example:	<pre>#include <stdio.h> /* for printf */ #include <stdarg.h> /* for va_arg, va_start, va_list, va_end */</pre>

va_arg (Continued)

```
void tprint(const char *fmt, ...)
{
    va_list ap;

    va_start(ap, fmt);
    while (*fmt)
    {
        switch (*fmt)
        {
            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

va_end

Description:	Ends the use of <i>ap</i> .
Include:	<stdarg.h>
Prototype:	#define va_end(va_list ap)
Argument:	<i>ap</i> pointer to list of arguments

va_end (Continued)

Remarks: After a call to `va_end`, the argument list pointer `ap` is considered to be invalid. Further calls to `va_arg` should not be made until the next `va_start`. In MPLAB C30, `va_end` does nothing, so this call is not necessary but should be used for readability and portability.

Example: See `va_arg`.

va_start

Description: Sets the argument pointer `ap` to first optional argument in the variable-length argument list

Include: `<stdarg.h>`

Prototype: `#define va_start(va_list ap, last_arg)`

Argument: `ap` pointer to list of arguments
`last_arg` last named argument before the optional arguments

Example: See `va_arg`.

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)`

offsetof (Continued)

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 startup 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.

FILE

Description:	Stores information for a file stream.
Include:	<code><stdio.h></code>

fpos_t

Description:	Type of a variable used to store a file position.
Include:	<code><stdio.h></code>

Standard C Libraries with Math Functions

size_t

Description: The result type of the `sizeof` operator.
Include: `<stdio.h>`

_IOFBF

Description: Indicates full buffering.
Include: `<stdio.h>`
Remarks: Used by the function `setvbuf`.

_IOLBF

Description: Indicates line buffering.
Include: `<stdio.h>`
Remarks: Used by the function `setvbuf`.

_IONBF

Description: Indicates no buffering.
Include: `<stdio.h>`
Remarks: Used by the function `setvbuf`.

BUFSIZ

Description: Defines the size of the buffer used by the function `setbuf`.
Include: `<stdio.h>`
Value: 512

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 */
```

clearerr (Continued)

```
int main(void)
{
    FILE *myfile;

    if ((myfile = fopen("samclearerr.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 * <i>stream</i>);
Argument:	<i>stream</i> 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.

fclose (Continued)

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 nonzero if stream is at the end of file; otherwise, returns zero.

Example: `#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 );
    }
}
```

feof (Continued)

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 nonzero 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

fflush

Description:	Flushes 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.

fgetc

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|
|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:	<i>stream</i> target stream <i>pos</i> 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:	<pre>/* 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); } }</pre> <p>Output: Bytes read: program opens a file Bytes read: and reads bytes at Bytes read: program opens a file</p>

fgets

Description:	Get a string from a stream						
Include:	<stdio.h>						
Prototype:	<code>char *fgets(char *s, int n, FILE *stream);</code>						
Arguments:	<table><tr><td><i>s</i></td><td>pointer to the storage string</td></tr><tr><td><i>n</i></td><td>maximum number of characters to read</td></tr><tr><td><i>stream</i></td><td>pointer to the open stream.</td></tr></table>	<i>s</i>	pointer to the storage string	<i>n</i>	maximum number of characters to read	<i>stream</i>	pointer to the open stream.
<i>s</i>	pointer to the storage string						
<i>n</i>	maximum number of characters to read						
<i>stream</i>	pointer to the open stream.						
Return Value:	Returns a pointer to the string <i>s</i> 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 <i>s</i> until it has read <i>n</i> -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 <i>fgets</i> sets the error indicator, the array contents are indeterminate.						
Example:	<pre>#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 ", s); } fclose(buf); } }</pre> <p>Input: Contents of afile.txt (used as input): Short Longer string</p> <p>Output: Short Longer string </p>						

fopen

Description:	Opens a file.				
Include:	<stdio.h>				
Prototype:	<code>FILE *fopen(const char *filename, const char *mode);</code>				
Arguments:	<table><tr><td><i>filename</i></td><td>name of the file</td></tr><tr><td><i>mode</i></td><td>type of access permitted</td></tr></table>	<i>filename</i>	name of the file	<i>mode</i>	type of access permitted
<i>filename</i>	name of the file				
<i>mode</i>	type of access permitted				

fopen (Continued)

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");
    }
}
```

fopen (Continued)

```
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:	<stdio.h>
Prototype:	int fprintf(FILE *stream, const char *format, ...);
Arguments:	<i>stream</i> pointer to the stream in which to output data <i>format</i> 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.

fprintf (Continued)

Example:

```
#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 <i>c</i> , FILE * <i>stream</i>);
Arguments:	<i>c</i> character to be written <i>stream</i> 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.

fputc (Continued)

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:

<i>s</i>	string to be written
<i>stream</i>	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: <stdio.h>

Prototype: size_t fread(void *ptr, size_t size, size_t nelem, FILE *stream);

Arguments:

<i>ptr</i>	pointer to the storage buffer
<i>size</i>	size of item

fread (Continued)

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:

```
#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\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");
}
```

fread (Continued)

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: `FILE *freopen(const char *filename, const char *mode, FILE *stream);`

Arguments:

<i>filename</i>	name of the new file
<i>mode</i>	type of access permitted
<i>stream</i>	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.

freopen (Continued)

Example: `#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:

<i>stream</i>	pointer to the open stream from which to read data
<i>format</i>	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`.

fscanf (Continued)

Example: `#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\n", s);
        fscanf(myfile, "%s", s);
        printf("%s\n", s);
        fscanf(myfile, "%s", s);
        printf("%s\n", s);
        fscanf(myfile, "%d", &x);
        printf("%d\n", x);
        fscanf(myfile, "%s", s);
        printf("%s\n", s);
        fscanf(myfile, "%c", a);
        printf("%c\n", 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: `int fseek(FILE *stream, long offset, int mode);`

Arguments:

<i>stream</i>	stream in which to move the file pointer.
<i>offset</i>	value to add to the current position
<i>mode</i>	type of seek to perform

fseek (Continued)

Return Value: Returns 0 if successful; otherwise, returns a non-zero value and set 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: #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, "
                    "this is the middle and "
                    "this is the end.");

        y = fseek(myfile, 0L, SEEK_SET);
        if (y)
            perror("Fseek failed");
        else
        {
            fgets(s, 22, myfile);
            printf("\n%s\n\n", s);
        }

        y = fseek(myfile, 2L, SEEK_CUR);
        if (y)
            perror("Fseek failed");
        else
        {
            fgets(s, 70, myfile);
            printf("\n%s\n\n", s);
        }

        y = fseek(myfile, -16L, SEEK_END);
        if (y)
            perror("Fseek failed");
        else
        {
            fgets(s, 70, myfile);
            printf("\n%s\n", s);
        }
        fclose(myfile);
    }
}
```

fseek (Continued)

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

Description: Sets the stream's file position.

Include: `<stdio.h>`

Prototype: `int fsetpos(FILE *stream, const fpos_t *pos);`

Arguments:

<i>stream</i>	target stream
<i>pos</i>	position-indicator storage as returned by an earlier call to <code>fgetpos</code>

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`.

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];
```

fsetpos (Continued)

```
        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:	<i>stream</i> stream in which to get the current file position
Return Value:	Returns the position of the file pointer if successful; otherwise, returns -1.
Example:	<pre>#include <stdio.h> /* for ftell, fread, fprintf, printf, fopen, fclose, sizeof, FILE, NULL */ int main(void) { FILE *myfile; char s[75]; long y;</pre>

ftell (Continued)

```
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:	<table><tr><td><i>ptr</i></td><td>pointer to the storage buffer</td></tr><tr><td><i>size</i></td><td>size of item</td></tr><tr><td><i>nelem</i></td><td>maximum number of items to be read</td></tr><tr><td><i>stream</i></td><td>pointer to the open stream</td></tr></table>	<i>ptr</i>	pointer to the storage buffer	<i>size</i>	size of item	<i>nelem</i>	maximum number of items to be read	<i>stream</i>	pointer to the open stream
<i>ptr</i>	pointer to the storage buffer								
<i>size</i>	size of item								
<i>nelem</i>	maximum number of items to be read								
<i>stream</i>	pointer to the open stream								
Return Value:	Returns the number of complete elements successfully written, which will be less than <i>nelem</i> only if a write error is encountered.								
Remarks:	The function writes characters to a given stream from a buffer pointed to by <i>ptr</i> up to <i>nelem</i> elements whose size is specified by <i>size</i> . 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.								

fwrite (Continued)

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");

    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");
}
```

fwrite (Continued)

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 */`

```
int main(void)
{
    FILE *buf;
    char y;
```

getc (Continued)

```
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: #include <stdio.h> /* for getchar, printf */

```
int main(void)
{
    char y;

    y = getchar();
    printf("%c|", y);
    y = getchar();
    printf("%c|", y);
    y = getchar();
    printf("%c|", y);
    y = getchar();
    printf("%c|", y);
    y = getchar();
    printf("%c|", y);
    y = getchar();
    printf("%c|", 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 <code>stdin</code> .
Include:	<code><stdio.h></code>
Prototype:	<code>char *gets(char *s);</code>
Argument:	<code>s</code> pointer to the storage string
Return Value:	Returns a pointer to the string <code>s</code> if successful; otherwise, returns a null pointer
Remarks:	The function reads characters from the stream <code>stdin</code> and stores them into the string pointed to by <code>s</code> 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 <code>gets</code> sets the error indicator, the array contents are indeterminate.
Example:	<pre>#include <stdio.h> /* for gets, printf */ int main(void) { char y[50]; gets(y) ; printf("Text: %s\n", y); }</pre> <p>Input: Contents of <code>UartIn.txt</code> (used as <code>stdin</code> input for simulator): Short Longer string</p> <p>Output: Text: Short</p>

perror

Description:	Prints an error message to <code>stderr</code> .
Include:	<code><stdio.h></code>
Prototype:	<code>void perror(const char *s);</code>
Argument:	<code>s</code> string to print
Return Value:	None.
Remarks:	The string <code>s</code> is printed followed by a colon and a space. Then an error message based on <code>errno</code> is printed followed by an newline

perror (Continued)

Example:

```
#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.

printf (Continued)

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.

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, 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:

```
#include <stdio.h> /* for printf */

int main(void)
{
    /* print a character right justified in a 3 */
    /* character space. */
    printf("%3c\n", 'a');
```

printf (Continued)

```
/* 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("%.4o\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
0x1c
1.100000E+20
-3.35
0.02
```

putc

Description:	Puts a character to the stream.
Include:	<stdio.h>
Prototype:	int putc(int <i>c</i> , FILE * <i>stream</i>);
Arguments:	<i>c</i> character to be written <i>stream</i> 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.

putc (Continued)

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:

```
T|h|i|s| |i|s| |t|e|x|t|
|
```

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 <code>stdout</code> .
Include:	<code><stdio.h></code>
Prototype:	<code>int puts(const char *s);</code>
Argument:	<i>s</i> string to be written
Return Value:	Returns a non-negative value if successful; otherwise, returns EOF.
Remarks:	The function writes characters to the stream <code>stdout</code> . A newline character is appended. The terminating null character is not written to the stream.
Example:	<pre>#include <stdio.h> /* for puts */ int main(void) { char buf[] = "This is text\n"; puts(buf); puts(" "); }</pre> <p>Output: This is text </p>

remove

Description:	Deletes the specified file.
Include:	<code><stdio.h></code>
Prototype:	<code>int remove(const char *filename);</code>
Argument:	<i>filename</i> name of file to be deleted.
Return Value:	Returns 0 if successful, -1 if not.
Remarks:	If <i>filename</i> does not exist or is open, <code>remove</code> will fail.
Example:	<pre>#include <stdio.h> /* for remove, printf */ int main(void) { if (remove("myfile.txt") != 0) printf("Cannot remove file"); else printf("File removed"); }</pre> <p>Output: File removed</p>

rename

Description:	Renames the specified file.
Include:	<code><stdio.h></code>
Prototype:	<code>int rename(const char *old, const char *new);</code>
Arguments:	<i>old</i> pointer to the old name <i>new</i> pointer to the new name.

rename (Continued)

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:	<pre>#include <stdio.h> /* for rename, printf */ int main(void) { if (rename("myfile.txt", "newfile.txt") != 0) printf("Cannot rename file"); else printf("File renamed"); }</pre> <p>Output: File renamed</p>

rewind

Description:	Resets the file pointer to the beginning of the file.
Include:	<stdio.h>
Prototype:	void rewind(FILE *stream);
Argument:	<i>stream</i> stream to reset the file pointer
Remarks:	The function calls <code>fseek(stream, 0L, SEEK_SET)</code> and then clears the error indicator for the given stream.
Example:	<pre>#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); } }</pre> <p>Output: I have 10 cookies. I ate 10 cookies.</p>

scanf

Description:	Scans formatted text from <code>stdin</code> .
Include:	<code><stdio.h></code>
Prototype:	<code>int scanf(const char *format, ...);</code>
Argument:	<i>format</i> 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 failure is encountered before the first.
Remarks:	<p>Each format specifier begins with a percent sign followed by optional fields and a required type as shown here:</p> <p style="margin-left: 40px;"><code>%[*][width][modifier]type</code></p> <p style="margin-left: 40px;">*</p> <p style="margin-left: 80px;">indicates assignment suppression. This will cause the input field to be skipped and no assignment made.</p> <p style="margin-left: 40px;">width</p> <p style="margin-left: 80px;">specify the maximum number of input characters to match for the conversion not including white space that can be skipped.</p> <p style="margin-left: 40px;">modifier</p> <p style="margin-left: 80px;">h modifier - used with type d, i, o, u, x, X; converts the value to a short int or unsigned short int.</p> <p style="margin-left: 80px;">h modifier - used with n; specifies that the pointer points to a short int</p> <p style="margin-left: 80px;">l modifier - used with type d, i, o, u, x, X; converts the value to a long int or unsigned long int</p> <p style="margin-left: 80px;">l modifier - used with n; specifies that the pointer points to a long int</p> <p style="margin-left: 80px;">l modifier - used with c; specifies a wide character</p> <p style="margin-left: 80px;">l modifier - used with type e, E, f, F, g, G; converts the value to a double</p> <p style="margin-left: 80px;">ll modifier - used with type d, i, o, u, x, X; converts the value to a long long int or unsigned long long int</p> <p style="margin-left: 80px;">ll modifier - used with n; specifies that the pointer points to a long long int</p> <p style="margin-left: 80px;">L modifier - used with e, E, f, g, G; converts the value to a long double</p>

scanf (Continued)

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 read so far. No characters are scanned.
[...]	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:

```
#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 * <i>stream</i> , char * <i>buf</i>);
Arguments:	<i>stream</i> pointer to the open stream <i>buf</i> user allocated buffer
Remarks:	setbuf must be called after fopen but before any other function calls that operate on the stream. If <i>buf</i> is a null pointer, setbuf calls the function setvbuf(<i>stream</i> , 0, _IONBF, BUFSIZ) for no buffering; otherwise setbuf calls setvbuf(<i>stream</i> , <i>buf</i> , _IOFBF, BUFSIZ) for full buffering with a buffer of size BUFSIZ. See setvbuf.
Example:	<pre>#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); } }</pre> <p>Output: myfile1 has no buffering myfile2 has full buffering</p>

setvbuf

Description:	Defines the stream to be buffered and the buffer size.
Include:	<stdio.h>
Prototype:	int setvbuf(FILE * <i>stream</i> , char * <i>buf</i> , int <i>mode</i> , size_t <i>size</i>);
Arguments:	<i>stream</i> pointer to the open stream <i>buf</i> user allocated buffer <i>mode</i> type of buffering <i>size</i> size of buffer
Return Value:	Returns 0 if successful

setvbuf (Continued)

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:

```
#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:

<i>s</i>	storage string for output
<i>format</i>	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.

sprintf (Continued)

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:

<i>s</i>	storage string for input
<i>format</i>	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.

Example:

```
#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);
}
```

sscanf (Continued)

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:	<pre>#include <stdio.h> /* for tmpfile, printf, FILE, NULL */ int main(void) { FILE *mytmpfile; if ((mytmpfile = tmpfile()) == NULL) printf("Cannot create temporary file"); else printf("Temporary file was created"); }</pre> Output: Temporary file was created

tmpnam

Description:	Creates a unique temporary filename
Include:	<stdio.h>
Prototype:	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.

tmpnam (Continued)

Example:

```
#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: 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.

ungetc (Continued)

Example: `#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: `<stdio.h>`

`<stdarg.h>`

Prototype: `int vfprintf(FILE *stream, const char *format, va_list ap);`

Arguments:

<i>stream</i>	pointer to the open stream
<i>format</i>	format control string
<i>ap</i>	pointer to a list of arguments

Return Value: Returns number of characters generated or a negative number if an error occurs.

vfprintf (Continued)

Remarks:	<p>The format argument has the same syntax and use that it has in <code>printf</code>.</p> <p>To access the variable length argument list, the <code>ap</code> variable must be initialized by the macro <code>va_start</code> and may be reinitialized by additional calls to <code>va_arg</code>. This must be done before the <code>vfprintf</code> function is called. Invoke <code>va_end</code> after the function returns. For more details see <code>stdarg.h</code>.</p>
Example:	<pre>#include <stdio.h> /* for vfprintf, fopen, fclose, printf, FILE, NULL */ #include <stdarg.h> /* for va_start, va_list, va_end */ 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); }</pre> <p>Output: Contents of <code>afile.txt</code> Error: The letter 'a' is not an integer value. Error: Requires 3 or more characters.</p>

vprintf

Description:	Prints formatted text to <code>stdout</code> using a variable length argument list
Include:	<code><stdio.h></code> <code><stdarg.h></code>
Prototype:	<code>int vprintf(const char *format, va_list ap);</code>
Arguments:	<i>format</i> format control string <i>ap</i> pointer to a list of arguments

vprintf (Continued)

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:

```
#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.

vsprintf (Continued)

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 errormsg(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;

    errormsg("The letter '%c' is not %s\n", 'a',
            "an integer value.");
    errormsg("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.

div_t

Description: A type that holds a quotient and remainder of a signed integer division with operands of type `int`.

Include: `<stdlib.h>`

Prototype: `typedef struct { int quot, rem; } div_t;`

Remarks: This is the structure type returned by the function `div`.

ldiv_t

Description: A type that holds a quotient and remainder of a signed integer division with operands of type `long`.

Include: `<stdlib.h>`

ldiv_t (Continued)

Prototype: `typedef struct { long quot, rem; } ldiv_t;`

Remarks: This is the structure type returned by the function `ldiv`.

size_t

Description: The type of the result of the `sizeof` operator.

Include: `<stdlib.h>`

wchar_t

Description: A type that holds a wide character value.

Include: `<stdlib.h>`

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: void abort(void);

Remarks: abort will cause the processor to reset.

Example:

```
#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.

abs (Continued)

Example:

```
#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:

```
#include <stdio.h> /* for scanf, printf */
#include <stdlib.h> /* for atexit, exit */

void good_msg(void);
void bad_msg(void);
void end_msg(void);
```

atexit (Continued)

```
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

Standard C Libraries with Math Functions

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:	<p>The number may consist of the following:</p> <p>[whitespace] [sign] digits [.digits] [{ e E } [sign] digits]</p> <p>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 strtod(s, 0, 0) except it does no error checking so errno will not be set.</p>
Example:	<pre>#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); }</pre> <p>Output:</p> <pre>String = "1.28" float = 1.280000 String = "27.835:e2" float = 2783.500000 String = "Number1" float = 0.000000</pre>

atoi

Description:	Converts a string to an integer.
Include:	<stdlib.h>
Prototype:	int atoi(const char *s);
Argument:	s string to be converted
Return Value:	Returns the converted integer if successful; otherwise, returns 0.

atoi (Continued)

Remarks: The number may consist of the following:
[whitespace] [sign] digits
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:

```
#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\" \t int = %d\n", a, x);

    x = atoi(b);
    printf("String = \"%s\" \t int = %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: 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:
[whitespace] [sign] digits
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:

```
#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:

<i>key</i>	object to search for
<i>base</i>	pointer to the start of the search data
<i>nelem</i>	number of elements
<i>size</i>	size of elements
<i>cmp</i>	pointer to the comparison function
<i>ck</i>	pointer to the key for the search
<i>ce</i>	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 `comp` uses ascending order.

bsearch (Continued)

Example:

```
#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
```


Standard C Libraries with Math Functions

calloc

Description:	Allocates an array in memory and initializes the elements to 0.
Include:	<stdlib.h>
Prototype:	void *calloc(size_t <i>nelem</i> , size_t <i>size</i>);
Arguments:	<i>nelem</i> number of elements <i>size</i> 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:

```
/* 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:	div_t div(int <i>numer</i> , int <i>denom</i>);
Arguments:	<i>numer</i> numerator <i>denom</i> denominator
Return Value:	Returns the quotient and the remainder.

div (Continued)

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:

```
#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\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\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\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\n", z.quot, z.rem);

    x = 7;
    y = 0;
    printf("For div(%d, %d)\n", x, y);
    z = div(x, y);
    printf("The quotient is %d and the "
           "remainder is %d\n\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: 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:

```
#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:	<i>ptr</i> 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:	<pre>#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"); } }</pre> <p>Output: Memory allocated Memory freed</p>

getenv

Description:	Get a value for an environment variable.
Include:	<stdlib.h>
Prototype:	char *getenv(const char *name);
Argument:	<i>name</i> 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:

```
#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:

```
#include <stdio.h> /* for printf */
#include <stdlib.h> /* for labs */

int main(void)
{
    long i;

    i = 123456;
    printf("The absolute value of %7ld is %6ld\n",
        i, labs(i));

    i = -246834;
    printf("The absolute value of %7ld is %6ld\n",
        i, labs(i));

    i = 0;
    printf("The absolute value of %7ld is %6ld\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:	<code>ldiv_t ldiv(long numer, long denom);</code>
Arguments:	<i>numer</i> numerator <i>denom</i> 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:	<pre>#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);</pre>

ldiv (Continued)

```
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);
}
```

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");
    }
}
```

malloc (Continued)

Output:

Memory allocated

Memory freed

mblen

Description: Gets the length of a multibyte character. (See Remarks.)**Include:** <stdlib.h>**Prototype:** 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:** 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:** 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>

qsort (Continued)

Prototype: `void qsort(void *base, size_t nelem, size_t size,
 int (*cmp)(const void *e1, const void *e2));`

Arguments:

<i>base</i>	pointer to the start of the array
<i>nelem</i>	number of elements
<i>size</i>	size of the elements
<i>cmp</i>	pointer to the comparison function
<i>e1</i>	pointer to the key for the search
<i>e2</i>	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 `cmp` uses ascending order.

Example:

```
#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:	<pre>#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()); }</pre> <p>Output: Number = 21422 Number = 2061 Number = 16443 Number = 11617 Number = 9125</p> <p>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.</p>

realloc

Description:	Reallocates memory to allow a size change.
Include:	<stdlib.h>
Prototype:	void *realloc(void *ptr, size_t size);
Arguments:	<i>ptr</i> points to previously allocated memory <i>size</i> 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:

```
#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: <stdlib.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.

srand (Continued)

Example:

```
#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: <stdlib.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:

```
#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 );
}
```

strtod (Continued)

```
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\n", end );

    x = strtol(b, &end, 4);
    printf("String = \"%s\" long = %ld\n", b, x );
    printf("Stopped at: %s\n\n", end );
}
```

strtol (Continued)

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: 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:

```
#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 = \"%s\"   long = %lu\n", a, x );
    printf("Stopped at: %s\n\n", end );

    x = strtoul(b, &end, 0);
    printf("String = \"%s\"   long = %lu\n", b, x );
    printf("Stopped at: %s\n\n", end );

    x = strtoul(c, &end, 0);
    printf("String = \"%s\"   long = %lu\n", c, x );
    printf("Stopped at: %s\n\n", end );
}
```

strtoul (Continued)

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: 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: /* 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: 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: size_t wcstombs(char *s, const wchar_t *wcs,
 size_t n);

Arguments: s points to the multibyte string

wcstombs (Continued)

	<i>wcs</i>	points to the wide character string
	<i>n</i>	the number of characters to convert
Return Value:	Returns the number of characters stored excluding the null character.	
Remarks:	<i>wcstombs</i> converts <i>n</i> 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.

size_t

Description:	The type of the result of the <code>sizeof</code> operator.
Include:	<code><string.h></code>

NULL

Description:	The value of a null pointer constant.
Include:	<code><string.h></code>

memchr

Description:	Locates a character in a buffer.
Include:	<code><string.h></code>
Prototype:	<code>void *memchr(const void *s, int c, size_t n);</code>
Arguments:	<i>s</i> pointer to the buffer <i>c</i> character to search for <i>n</i> number of characters to check
Return Value:	Returns a pointer to the location of the match if successful; otherwise, returns null.
Remarks:	<code>memchr</code> stops when it finds the first occurrence of <i>c</i> or after searching <i>n</i> number of characters.
Example:	<pre>#include <string.h> /* for memchr, NULL */ #include <stdio.h> /* for printf */ int main(void) { char buf1[50] = "What time is it?"; char ch1 = 'i', ch2 = 'y'; char *ptr; int res;</pre>

memchr (Continued)

```
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("\n");

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:	int memcmp(const void *s1, const void *s2, size_t n);
Arguments:	<div><i>s1</i> first buffer</div> <div><i>s2</i> second buffer</div> <div><i>n</i> number of characters to compare</div>
Return Value:	Returns a positive number if <i>s1</i> is greater than <i>s2</i> , zero if <i>s1</i> is equal to <i>s2</i> , or a negative number if <i>s1</i> is less than <i>s2</i> .
Remarks:	This function compares the first <i>n</i> characters in <i>s1</i> to the first <i>n</i> characters in <i>s2</i> and returns a value indicating whether the buffers are less than, equal to or greater than each other.

memcmp (Continued)

Example:

```
#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\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");

    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:	<i>dst</i> buffer to copy characters to <i>src</i> buffer to copy characters from <i>n</i> number of characters to copy
Return Value:	Returns <i>dst</i> .
Remarks:	memcpy copies <i>n</i> characters from the source buffer <i>src</i> to the destination buffer <i>dst</i> . If the buffers overlap, the behavior is undefined.
Example:	<pre>#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\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); }</pre>

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 <i>n</i> 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:	<i>s1</i> buffer to copy characters to (destination) <i>s2</i> buffer to copy characters from (source)

memmove (Continued)

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:

```
#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\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.

memset (Continued)

Example:

```
#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\", \'?\',4);\n", buf1, ch1);
    memset(buf1, ch1, 4);
    printf("buf1 after memset: %s\n", buf1);

    printf("\n");
    printf("memset(\"%s\", \'y\',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: <string.h>

Prototype: 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:

```
#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\n", strlen(buf1));
    printf("buf2 : %s\n", buf2);
    printf("\t(%d characters)\n\n", strlen(buf2));
}
```

strcat (Continued)

```
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 hereWhere is the time?
      (28 characters)

buf1 after strcat of "Why?":
      We're hereWhere 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:	<i>s</i> pointer to the string <i>c</i> 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 <i>s</i> to find the first occurrence of the character <i>c</i> .
Example:	<pre>#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;</pre>

strchr (Continued)

```
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:	<string.h>
Prototype:	int strcmp(const char *s1, const char *s2);
Arguments:	<i>s1</i> first string <i>s2</i> second string
Return Value:	Returns a positive number if <i>s1</i> is greater than <i>s2</i> , zero if <i>s1</i> is equal to <i>s2</i> , or a negative number if <i>s1</i> is less than <i>s2</i> .
Remarks:	This function compares successive characters from <i>s1</i> and <i>s2</i> until they are not equal or the null terminator is reached.
Example:	<pre>#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;</pre>

strcmp (Continued)

```
printf("buf1 : %s\n", buf1);
printf("buf2 : %s\n", buf2);
printf("buf3 : %s\n\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");
}
```

Output:

```
buf1 : Where is the time?
buf2 : Where did they go?
buf3 : Why?
```

```
buf2 comes before buf1
```

```
buf1 comes before buf3
```

```
"Why?" and buf3 are equal
```

strcoll

Description:	Compares one string to another. (See Remarks.)
Include:	<string.h>
Prototype:	int strcoll(const char *s1, const char *s2);
Arguments:	<i>s1</i> first string <i>s2</i> second string
Return Value:	Using the locale-dependent rules, it returns a positive number if <i>s1</i> is greater than <i>s2</i> , zero if <i>s1</i> is equal to <i>s2</i> , or a negative number if <i>s1</i> is less than <i>s2</i> .

Standard C Libraries with Math Functions

strcoll (Continued)

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\n", buf3);

    strcpy(buf1, buf2);
    printf("buf1 after strcpy of buf2: \n\t%s\n\n",
        buf1);

    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

Description: Calculate the number of consecutive characters at the beginning of a string that are not contained in a set of characters.

Include: `<string.h>`

Prototype: `size_t strcspn(const char *s1, const char *s2);`

strcspn (Continued)

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 */
```

```
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.

strerror (Continued)

Example:

```
#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:

```
#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\n",
        strlen(str1));
    printf("str2 : %s\n", str2);
    printf("\t(string length = %d characters)\n\n",
        strlen(str2));
    printf("str3 : %s\n", str3);
    printf("\t(string length = %d characters)\n\n\n",
        strlen(str3));
}
```

strlen (Continued)

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:

<i>s1</i>	destination string to copy to
<i>s2</i>	source string to copy from
<i>n</i>	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:

```
#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?";

    printf("buf1 : %s\n", buf1);
    printf("\t(%d characters)\n\n", strlen(buf1));
    printf("buf2 : %s\n", buf2);
    printf("\t(%d characters)\n\n", strlen(buf2));
    printf("buf3 : %s\n", buf3);
    printf("\t(%d characters)\n\n\n", strlen(buf3));

    strncat(buf1, buf2, 6);
    printf("buf1 after strncat of 6 characters "
           "of buf2: \n\t%s\n", buf1);
    printf("\t(%d characters)\n", strlen(buf1));

    printf("\n");

    strncat(buf1, buf2, 25);
    printf("buf1 after strncat of 25 characters "
           "of buf2: \n\t%s\n", buf1);
    printf("\t(%d characters)\n", strlen(buf1));
```

strncat (Continued)

```
printf("\n");

strncat(buf1, buf3, 4);
printf("buf1 after strncat of 4 characters "
      "of buf3: \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)

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:	<i>s1</i> first string <i>s2</i> second string <i>n</i> number of characters to compare
Return Value:	Returns a positive number if <i>s1</i> is greater than <i>s2</i> , zero if <i>s1</i> is equal to <i>s2</i> , or a negative number if <i>s1</i> is less than <i>s2</i> .
Remarks:	strncmp returns a value based on the first character that differs between <i>s1</i> and <i>s2</i> . Characters that follow a null character are not compared.

strncmp (Continued)

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");

    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(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));
```

strncpy (Continued)

```
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)
```

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:	char *strpbrk(const char *s1, const char *s2);
Arguments:	<i>s1</i> pointer to the string to be searched <i>s2</i> pointer to characters to search for

strpbrk (Continued)

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: `#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");

    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

strrchr (Continued)

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);

    printf("\n");

    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: 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*.

strspn (Continued)

Remarks: This function stops searching when a character from *s1* is not in *s2*.

Example:

```
#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);
}
```

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.

strstr (Continued)

Example:

```
#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\n", str1);
    printf("str2 : %s\n", str2);
    printf("str3 : %s\n\n", str3);

    ptr = strstr(str1, str2);
    if (ptr != NULL)
    {
        res = ptr - str1 + 1;
        printf("\"%s\" found at position %d\n",
            str2, res);
    }
    else
        printf("\"%s\" not found\n", str2);

    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:	<string.h>
Prototype:	char *strtok(char *s1, const char *s2);
Arguments:	s1 pointer to the null terminated string to be searched

strtok (Continued)

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.

Example:

```
#include <string.h> /* for strtok, NULL */
#include <stdio.h> /* for printf */

int main(void)
{
    char str1[30] = "Here, on top of the world!";
    char delim[5] = ", .";
    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:	<i>s1</i> destination string <i>s2</i> source string to be transformed <i>n</i> number of characters to transform
Return Value:	Returns the length of the transformed string not including the terminating null character. If <i>n</i> is zero, the string is not transformed (<i>s1</i> may be a point null in this case) and the length of <i>s2</i> is returned.
Remarks:	If the return value is greater than or equal to <i>n</i> , the content of <i>s1</i> is indeterminate. Since MPLAB C30 does not support alternate locales, the transformation is equivalent to <code>strcpy</code> , except that the length of the destination string is bounded by <i>n</i> -1.

4.16 <TIME.H> DATE AND TIME FUNCTIONS

The header file `time.h` consists of types, macros and functions that manipulate time.

clock_t

Description:	Stores processor time values.
Include:	<time.h>
Prototype:	typedef long clock_t

size_t

Description:	The type of the result of the <code>sizeof</code> operator.
Include:	<time.h>

struct tm

Description:	Structure used to hold the time and date (calendar time).
Include:	<time.h>
Prototype:	<pre>struct tm { int tm_sec; /*seconds after the minute (0 to 61) */ /* allows for up to two leap seconds */ int tm_min; /* minutes after the hour (0 to 59) */ int tm_hour; /* hours since midnight (0 to 23) */ int tm_mday; /* day of month (1 to 31) */ int tm_mon; /* month (0 to 11 where January = 0) */ int tm_year; /* years since 1900 */ int tm_wday; /* day of week (0 to 6 where Sunday = 0) */ int tm_yday; /* day of year (0 to 365 where January 1 = 0) */ int tm_isdst; /* Daylight Savings Time flag */ }</pre>
Remarks:	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.

time_t

Description: Represents calendar time values.
Include: `<time.h>`
Prototype: `typedef long time_t`

CLOCKS_PER_SEC

Description: Number of processor clocks per second.
Include: `<time.h>`
Prototype: `#define CLOCKS_PER_SEC`
Value: 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: `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

asctime (Continued)

Example:

```
#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: 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.

Example:

```
#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:	<i>tod</i> 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:	<pre>#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)); }</pre>

Output:

Day and time Sat Feb 1 02:30:30 2003

difftime

Description:	Find the difference between two times.
Include:	<time.h>
Prototype:	double difftime(time_t t1, time_t t0);
Arguments:	<i>t1</i> ending time <i>t0</i> beginning time
Return Value:	Returns the number of seconds between <i>t1</i> and <i>t0</i> .
Remarks:	By default, MPLAB C30 returns the time as instruction cycles so difftime returns the number of ticks between <i>t1</i> and <i>t0</i> .

difftime (Continued)

Example:

```
#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: 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:

```
#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:	<i>tod</i> 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 <code>localtime</code> and <code>gmtime</code> will be equivalent except <code>localtime</code> will return <code>tm_isdst</code> (Daylight Savings Time flag) as -1 to indicate that the status of Daylight Savings Time is not known.
Example:	<pre>#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)); }</pre> <p>Output: Local time = Mon Oct 20 16:43:02 2003</p>

mktime

Description:	Converts local time to a calendar value.
Include:	<time.h>
Prototype:	time_t mktime(struct tm *tptr);
Argument:	<i>tptr</i> a pointer to the time structure
Return Value:	Returns the calendar time encoded as a value of <code>time_t</code> .
Remarks:	If the calendar time cannot be represented, the function returns -1, cast as a <code>time_t</code> (i.e. <code>(time_t)-1</code>).

mktime (Continued)

Example:

```
#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: `<time.h>`

Prototype: `size_t strftime(char *s, size_t n, const char *format, const struct tm *tptr);`

Arguments:

<i>s</i>	output string
<i>n</i>	maximum length of string
<i>format</i>	format-control string
<i>tptr</i>	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)
- %I** hour of the day (01-12)
- %j** day of the year (001-366)
- %m** month of the year (01-12)
- %M** minute of the hour (00-59)
- %p** AM/PM designator
- %S** second of the minute (00-61)
allowing for up to two leap seconds

strftime (Continued)

%U week number of the year where Sunday is the first day of week 1 (00-53)
%w weekday where Sunday is day 0 (0-6)
%W week number of the year where Monday is the first day of week 1 (00-53)
%x appropriate date representation
%X appropriate time representation
%y year without century (00-99)
%Y year with century
%Z time zone (possibly abbreviated) or no characters if time zone is unavailable
%% percent character %

Example:

```
#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:	<i>tod</i> 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.

time (Continued)

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: `double acos (double x);`

Argument: `x` value between -1 and 1 for which to return the arccosine

Return Value: Returns the arc cosine in radians in the range of 0 to pi (inclusive).

acos (Continued)

Remarks: A domain error occurs if x is less than -1 or greater than 1.

Example:

```
#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);

    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 π (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;
```

acosf (Continued)

```
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 = 0.0F;
y = acosf (x);
if (errno)
    perror("Error");
printf("The arccosine of %f is %f\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:	<math.h>
Prototype:	double asin (double x);
Argument:	x value between -1 and 1 for which to return the arcsine
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:	

```
#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);
}
```

asin (Continued)

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: <math.h>

Prototype: 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:

```
#include <math.h> /* for asinf */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */
```

```
int main(void)
{
    float x, y;

    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: double atan (double x);

Argument: x value for which to return the arctangent

atan (Continued)

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:

```
#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 arctangent

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:

```
#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 arctangent function of y/x .
Include:	<math.h>
Prototype:	double atan2 (double y , double x);
Arguments:	y y value for which to return the arctangent x x value for which to return the arctangent
Return Value:	Returns the arctangent in radians in the range of $-\pi$ to π (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 \pm infinity.

Example:

```
#include <math.h> /* for atan2 */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

#include <math.h> /* for atan2 */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    double x, y, z;

    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);
}
```

atan2 (Continued)

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 arctangent
x *x* value for which to return the arctangent

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:

```
#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 %f/%f is %f\n\n",
        y, x, z);

    errno = 0;
    x = 0.0F;
    y = -1.0F;
    z = atan2f (y, x);
    if (errno)
        perror("Error");
    printf("The arctangent of %f/%f is %f\n\n",
        y, x, z);
```

atan2f (Continued)

```
    errno = 0;
    x = 0.0F;
    y = 0.0F;
    z = atan2f (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
```

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:

```
#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:	<pre>#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\n", x[i], y); } }</pre> <p>Output:</p> <pre>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</pre>

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:	<pre>#include <math.h> /* for cos */ #include <stdio.h> /* for printf, perror */ #include <errno.h> /* for errno */ int main(void) { double x,y;</pre>

cos (Continued)

```
    errno = 0;
    x = -1.0;
    y = cos (x);
    if (errno)
        perror("Error");
    printf("The cosine of %f is %f\n\n", x, y);

    errno = 0;
    x = 0.0;
    y = cos (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
```

cosf

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);
}
```

cosf (Continued)

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: <math.h>

Prototype: 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:

```
#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\n",
           x, y);

    errno = 0;
    x = 0.0;
    y = cosh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of %f is %f\n\n",
           x, y);

    errno = 0;
    x = 720.0;
    y = cosh (x);
    if (errno)
        perror("Error");
    printf("The hyperbolic cosine of %f is %f\n\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:

```
#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>

exp (Continued)

Prototype:	<code>double exp (double x);</code>
Argument:	<code>x</code> value for which to return the exponential
Return Value:	Returns the exponential of <code>x</code> . On an overflow, <code>exp</code> returns <code>inf</code> and on an underflow <code>exp</code> returns 0.
Remarks:	A range error occurs if the magnitude of <code>x</code> is too large.
Example:	<pre>#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\n", x, y); errno = 0; x = 1E3; y = exp (x); if (errno) perror("Error"); printf("The exponential of %f is %f\n\n", x, y); errno = 0; x = -1E3; y = exp (x); if (errno) perror("Error"); printf("The exponential of %f is %f\n\n", x, y); }</pre> <p>Output:</p> <pre>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</pre>

expf

Description:	Calculates the exponential function of <code>x</code> (e raised to the power <code>x</code> where <code>x</code> is a single precision floating point value).
Include:	<code><math.h></code>
Prototype:	<code>float expf (float x);</code>
Argument:	<code>x</code> floating point value for which to return the exponential
Return Value:	Returns the exponential of <code>x</code> . On an overflow, <code>expf</code> returns <code>inf</code> and on an underflow <code>exp</code> returns 0.

Standard C Libraries with Math Functions

expf (Continued)

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.

fabs (Continued)

Example:

```
#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:

```
#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: <math.h>

Prototype: double floor (double x);

floor (Continued)

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:

```
#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 ceiling 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: `<math.h>`

Prototype: `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:

```
#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);
    }
}
```

floorf (Continued)

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:	<math.h>
Prototype:	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 nonzero, the result will have the same sign as x and the magnitude of the result will be less than the magnitude of y .
Example:	<pre>#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); errno = 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);</pre>

fmod (Continued)

```
    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 .

fmodf (Continued)

Remarks: If $y = 0$, a domain error occurs. If y is nonzero, the result will have the same sign as x and the magnitude of the result will be less than the magnitude of y .

Example:

```
#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\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\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\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\n", x, y, z);

    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);
```

fmodf (Continued)

```
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;

    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", 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", n);
}
```

frexp (Continued)

```
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", 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:	float frexpf (float x, int *exp);
Arguments:	<div><div><div>x</div><div>floating point value for which to return the fraction and exponent</div></div><div><div>*exp</div><div>pointer to a stored integer exponent</div></div></div>
Return Value:	Returns the fraction, <i>exp</i> points to the exponent. If <i>x</i> 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:	<pre>#include <math.h> /* for frexpf */ #include <stdio.h> /* for printf */ int main(void) { float x,y; int n; x = 0.15F; y = frexpf (x, &n); printf("For frexpf of %f\n the fraction is %f\n ", x, y); printf(" and the exponent is %d\n\n", n); x = -2.5F; y = frexpf (x, &n); printf("For frexpf of %f\n the fraction is %f\n ", x, y); printf(" and the exponent is %d\n\n", n); }</pre>

frexpf (Continued)

```
x = 0.0F;
y = frexpf (x, &n);
printf("For frexpf of %f\n the fraction is %f\n ",
      x, y);
printf(" and the exponent is %d\n\n", n);
}
```

Output:

```
For frexpf of 0.150000
the fraction is 0.600000
and the exponent is -2

For frexpf of -2.500000
the fraction is -0.625000
and the exponent is 2

For frexpf 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: 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:

```
#include <math.h> /* for ldexp */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    double x,y;
    int n;

    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);
}
```

ldexp (Continued)

```
    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

ldexpf

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 $x * 2^{ex}$. On an overflow, ldexp returns inf and on an underflow, ldexp returns 0.

Remarks: A range error will occur on overflow or underflow.

Example: #include <math.h> /* for ldexpf */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

```
int main(void)
{
    float x,y;
    int n;
```

ldexpf (Continued)

```
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\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\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\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.

log (Continued)

Example:

```
#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 $x \leq 0$.

log10 (Continued)

Example:

```
#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\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\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\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:	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 $x \leq 0$.

log10f (Continued)

Example:

```
#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:	<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 $x \leq 0$.

logf (Continued)

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\n",
          x, y);

    errno = 0;
    x = 0.0F;
    y = logf (x);
    if (errno)
        perror("Error");
    printf("The natural logarithm of %f is %f\n\n",
          x, y);

    errno = 0;
    x = -2.0F;
    y = logf (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

modf

Description:	Splits a double precision floating point value into fractional and integer parts.
Include:	<math.h>
Prototype:	double modf(double <i>x</i> , double * <i>pint</i>);
Arguments:	<i>x</i> double precision floating point value <i>pint</i> pointer to a stored the integer part
Return Value:	Returns the signed fractional part and <i>pint</i> 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.

modf (Continued)

Example:

```
#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 %f the fraction is %f\n ", x, y);
    printf("    and the integer is %0.f\n\n", n);

    x = -15.2121;
    y = modf (x, &n);
    printf("For %f the fraction is %f\n ", x, y);
    printf("    and the integer is %0.f\n\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:

```
#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", 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", n);
}
```

modff (Continued)

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

pow

Description: Calculates x raised to the power y .

Include: `<math.h>`

Prototype: `double pow(double x, double y);`

Arguments: x the base
 y the exponent

Return Value: Returns x raised to the power y (x^y).

Remarks: If y is 0 `pow` returns 1. If x is 0.0 and y is less than 0 `pow` returns inf and a domain error occurs. If the result overflows or underflows, a range error occurs.

Example:

```
#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);
```

pow (Continued)

```
    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: <math.h>

Prototype: 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:

```
#include <math.h> /* for powf */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

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\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\n ", x, y, z);
}
```

powf (Continued)

```
    errno = 0;
    x = 0.0F;
    y = -3.0F;
    z = powf (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
```

```
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 x is a NaN or infinity.

Example:

```
#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\n", x, y);

    errno = 0;
    x = 0.0;
    y = sin (x);
    if (errno)
        perror("Error");
    printf("The sine of %f is %f\n\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:

```
#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\n", x, y);

    errno = 0;
    x = 0.0F;
    y = sinf (x);
    if (errno)
        perror("Error");
    printf("The sine of %f is %f\n\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: 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.

sinh (Continued)

Example:

```
#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:	float sinh (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.

sinhf (Continued)

Example:

```
#include <math.h> /* for sinh */
#include <stdio.h> /* for printf, perror */
#include <errno.h> /* for errno */

int main(void)
{
    float x, y;

    errno = 0;
    x = -1.0F;
    y = sinh(x);
    if (errno)
        perror("Error");
    printf("The hyperbolic sine of %f is %f\n\n",
           x, y);

    errno = 0;
    x = 0.0F;
    y = sinh(x);
    if (errno)
        perror("Error");
    printf("The hyperbolic sine of %f is %f\n\n",
           x, y);
}
```

Output:

```
The hyperbolic sine of -1.000000 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);
}
```

sqrt (Continued)

```
    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: 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:

```
#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);
}
```

sqrtf (Continued)

```
    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:

```
#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:

```
#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.

tanh (Continued)

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: <math.h>

Prototype: 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:

```
#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.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.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.a`.

4.18.1 Rebuilding the libpic30.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.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.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:	<code>void _exit (int status);</code>

_exit (Continued)

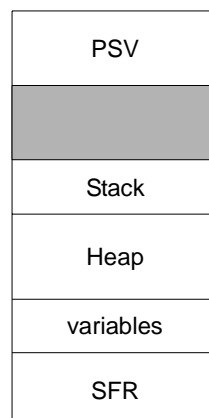
Argument:	<i>status</i> exit status
Remarks:	This is a helper function called by the <code>exit()</code> Standard C Library function.
Default Behavior:	As distributed, this function flushes stdout and terminates. The parameter status is the same as that passed to the <code>exit()</code> standard C library function.
File:	<code>_exit.c</code>

brk

Description:	Set the end of the process's data space.
Include:	None
Prototype:	<code>int brk(void *endds)</code>
Argument:	<i>endds</i> pointer to the end of the data segment
Return Value:	Returns '0' if successful, '-1' if not.
Remarks:	<p><code>brk()</code> 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.</p> <p>Newly allocated space is uninitialized.</p> <p>This helper function is used by the Standard C Library function <code>malloc()</code>.</p>

brk (Continued)

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: `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:	<code>char *getenv(const char *s);</code>
Argument:	<i>s</i> 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

seek

Description:	Move a file pointer to a specified location.
Include:	None
Prototype:	<code>long lseek(int handle, long offset, int origin);</code>
Argument:	<i>handle</i> refers to an opened file <i>offset</i> the number of characters from the origin <i>origin</i> the position from which to start the seek. <i>origin</i> may be one of the following values (as defined in <code>stdio.h</code>): SEEK_SET - Beginning of file. SEEK_CUR - Current position of file pointer. SEEK_END - End of file.
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 <code>fgetpos()</code> , <code>ftell()</code> , <code>fseek()</code> , <code>fsetpos()</code> , and <code>rewind()</code> .
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:	<code>int open(const char *name, int access, int mode);</code>
Argument:	<i>name</i> name of the file to be opened <i>access</i> access method to open file <i>mode</i> 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.

Standard C Libraries with Math Functions

open (Continued)

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 <code>fopen()</code> and <code>freopen()</code> .
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 <code>errno</code> is set to the value of the symbolic constant <code>EFOPEN</code> defined in <code><errno.h></code> .
File:	<code>open.c</code>

read

Description:	Read data from a file.						
Include:	None						
Prototype:	<pre>int read(int handle, void * buffer, unsigned int len);</pre>						
Argument:	<table><tr><td><i>handle</i></td><td>handle referring to an opened file</td></tr><tr><td><i>buffer</i></td><td>points to the storage location for read data</td></tr><tr><td><i>len</i></td><td>the maximum number of characters to read</td></tr></table>	<i>handle</i>	handle referring to an opened file	<i>buffer</i>	points to the storage location for read data	<i>len</i>	the maximum number of characters to read
<i>handle</i>	handle referring to an opened file						
<i>buffer</i>	points to the storage location for read data						
<i>len</i>	the maximum number of characters to read						
Return Value:	Returns the number of characters read, which may be less than <i>len</i> if there are fewer than <i>len</i> 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 <code>fgetc()</code> , <code>fgets()</code> , <code>fread()</code> , and <code>gets()</code> .						
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:	<code>read.c</code>						

remove

Description:	Remove a file.
Include:	<code><stdio.h></code>
Prototype:	<pre>int remove(const char *filename);</pre>

remove (Continued)

Argument:	<i>filename</i> 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:	<code>remove.c</code>

rename

Description:	Rename a file or directory.
Include:	<code><stdio.h></code>
Prototype:	<code>int rename(const char *oldname, const char *newname);</code>
Argument:	<i>oldname</i> pointer to the old name <i>newname</i> pointer to the new name
Return Value:	Returns '0' if it is successful. On an error, the function returns a nonzero 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:	<code>rename.c</code>

sbrk

Description:	Extend the process's data space by a given increment.
Include:	None
Prototype:	<code>void * sbrk(int incr);</code>
Argument:	<i>incr</i> number of characters to increment/decrement
Return Value:	Return the start of the new space allocated, or '-1' for errors.
Remarks:	<p><code>sbrk()</code> adds <i>incr</i> characters to the break value and changes the allocated space accordingly. <i>incr</i> can be negative, in which case the amount of allocated space is decreased.</p> <p><code>sbrk()</code> 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.</p> <p>This is a helper function called by the Standard C Library function <code>malloc()</code>.</p>

sbrk (Continued)

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.

write (Continued)

Remarks:	<p>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.</p> <p>This is a helper function called by the Standard C Library function <code>fflush()</code>.</p>
Default Behavior:	<p>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.</p>
File:	<p><code>write.c</code></p>

Appendix A. ASCII Character Set

TABLE A-1: ASCII CHARACTER SET

		Most Significant Character							
Least Significant Character	HEX	0	1	2	3	4	5	6	7
	0	NUL	DLE	Space	0	@	P	'	p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	"	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	\$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	Bell	ETB	'	7	G	W	g	w
	8	BS	CAN	(8	H	X	h	x
	9	HT	EM)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[k	{
	C	FF	FS	,	<	L	\	l	
	D	CR	GS	-	=	M]	m	}
	E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL	

NOTES:

Index

Symbols

#define	4, 64, 65, 86, 93, 131, 146, 166
#if	190
#include	4, 64, 174
%, Percent	231, 236, 237, 299
-, Dash	237
\f, Form Feed	181
\n, Newline	181, 203, 204, 213, 218, 229, 234
\r, Carriage Return	181
\t, Horizontal Tab	181
\v, Vertical Tab	181
^, Caret	237
__FILE__	175
__LINE__	175
_exit	339
_IOBF	205, 238, 239
_IOLBF	205, 239
_IONBF	205, 238, 239
_MathError	198
_NSETJMP	193

Numerics

0x	183, 230, 267, 268
----------	--------------------

A

Abnormal Termination Signal	195
abort	175, 248
abs	248
Absolute Value	
Double Floating Point	313
Integer	248
Long Integer	259
Single Floating Point	314
Absolute Value Function	
abs	248
fabs	313
fabsf	314
labs	259
Access Mode	
Binary	214
Text	214
AckI2C	165
acos	300
acosf	301
ADC, 10-Bit	
Busy	89
Close	89
Configure Interrupt	89
Disable Interrupt Macro	94
Enable Interrupt Macro	94
Example of Use	95
Open	90
Read	93

Set Channel	93
Set Interrupt Priority Macro	94
Start Conversion	90
Stop Sampling	93
ADC, 12-Bit	
Busy	82
Close	82
Configure Interrupt	83
Disable Interrupt Macro	87
Enable Interrupt Macro	87
Example of Use	88
Open	83
Read	86
Set Channel	86
Set Interrupt Priority Macro	87
Start Conversion	83
Stop Sampling	86
Allocate Memory	261
calloc	255
Free	258
realloc	264
Alphabetic Character	
Defined	176
Test for	176
Alphanumeric Character	
Defined	175
Test for	175
AM/PM	298
Append	275, 282
arccosine	
Double Floating Point	300
Single Floating Point	301
arcsine	
Double Floating Point	302
Single Floating Point	303
arctangent	
Double Floating Point	303
Single Floating Point	304
arctangent of y/x	
Double Floating Point	305
Single Floating Point	306
Argument List	200, 243, 244, 245
Arithmetic Error Message	195
ASCII Character Set	347
asctime	293
asin	302
assert	174
assert.h	174
Assignment Suppression	236
Asterisk	230, 231, 236
atan	303

dsPIC™ Language Tools Libraries

atan2	305	CANxSetTXMode	77
atan2f	306	Caret (^)	237
atanf	304	Carriage Return	181
atexit	249, 257	ceil	307
atof	251	ceilf	308
atoi	251	ceiling	
atol	252	Double Floating Point	307
B		Single Floating Point	308
BartlettInit	20	char	
Base	267, 268	Maximum Value	190
10	186, 187, 188, 189, 190, 324, 325	Minimum Value	190
2	189	Number of Bits	190
e	323, 326	CHAR_BIT	190
FLT_RADIX	186, 187, 188, 190	CHAR_MAX	190
Binary		CHAR_MIN	190
Base	189	Character Array	237
Mode	214, 241	Character Case Mapping	
Search	253	Lower Case Alphabetic Character	183
Streams	203	Upper Case Alphabetic Character	184
Bitfields	203	Character Case Mapping Functions	
BitReverseComplex	52	tolower	183
BlackmanInit	21	toupper	184
BORStatReset	106	Character Handling, See ctype.h	
brk	340, 345	Character Input/Output Functions	
bsearch	253	fgetc	211
Buffer Size	205, 238	fgets	213
BufferEmptyDCI	134	fputc	216
Buffering Modes	239	fputs	217
Buffering, See File Buffering		getc	227
BUFSIZ	205, 238	getchar	228
BusyADC10	89	gets	229
BusyADC12	82	putc	232
BusyUartx	126	putchar	233
BusyXLCD	64	puts	234
C		ungetc	242
C Locale	175, 193	Character Testing	
Calendar Time	292, 293, 295, 296, 297, 299	Alphabetic Character	176
calloc	255, 258	Alphanumeric Character	175
CAN Interrupts		Control Character	177
Configure	79	Decimal Digit	177
Disable Macro	80	Graphical Character	178
Enable Macro	80	Hexadecimal Digit	183
Set Priority Macro	81	Lower Case Alphabetic Character	179
CAN, Example of Use	81	Printable Character	180
CANxAbortAll	69	Punctuation Character	181
CANxGetRXErrorCount	70	Upper Case Alphabetic Character	182
CANxGetTXErrorCount	70	White-Space Character	181
CANxInitialize	78	Character Testing Functions	
CANxIsBusOff	70	isalnum	175
CANxIsRXPassive	71	isalpha	176
CANxIsRXReady	71	iscntrl	177
CANxIsTXPassive	72	isdigit	177
CANxIsTXReady	72	isgraph	178
CANxReceiveMessage	72	islower	179
CANxSendMessage	73	isprint	180
CANxSetFilter	74	ispunct	181
CANxSetMask	75	isspace	181
CANxSetOperationMode	75	isupper	182
CANxSetOperationModeNoWait	76	isxdigit	183
CANxSetRXMode	77	Characters	

Alphabetic	176	ConfigIntTimerx	97
Alphanumeric	175	ConfigIntTimerxx	97
Control	177	ConfigIntUARTx	127
Convert to Lower Case Alphabetic	183	ConfigINTx	107
Convert to Upper Case Alphabetic	184	Control Character	
Decimal Digit	177	Defined	177
Graphical	178	Test for	177
Hexadecimal Digit	183	Control Transfers	193
Lower Case Alphabetic	179	Conversion	230, 236, 240
Printable	180	Convert	
Punctuation	181	Character to Multibyte Character	269
Upper Case Alphabetic	182	Multibyte Character to Wide Character	262
White-Space	181	Multibyte String to Wide Character String	262
Classifying Characters	175	String to Double Floating Point	251, 266
clearerr	207	String to Integer	251
Clearing Error Indicator	207	String to Long Integer	252, 267
clock	294	String to Unsigned Long Integer	268
clock_t	292, 294	To Lower Case Alphabetic Character	183
CLOCKS_PER_SEC	293	To Upper Case Alphabetic Character	184
close	341	Wide Character String to Multibyte String	269
CloseADC10	89	ConvertADC10	90
CloseADC12	82	ConvertADC12	83
CloseCapturex	111	Copying Functions	
CloseDCI	134	memcpy	273
CloseI2C	164	memmove	273
CloseINTx	107	memset	274
CloseMCPWM	153	strcpy	279
CloseOCx	117	strncpy	285
CloseQEI	148	cos	308
CloseSP1x	141	cosf	309
CloseTimerx	96	CosFactorInit	52
CloseTimerxx	96	cosh	310
CloseUARTx	126	coshf	311
Common Definitions, See stddef.h		cosine	
Compare Strings	277	Double Floating Point	308
Comparison Function	253, 263	Single Floating Point	309
Comparison Functions		crt0, crt1	1
memcmp	271	ctime	295
strcmp	277	ctype.h	175
strcoll	278	isalnum	175
strncmp	283	iscntrl	177
strxfrm	292	isdigit	177
Compiler Options		isgraph	178
-fno-short-double	204	isalpha	176
-msmart-io	204	islower	179
Concatenation Functions		ispring	180
strcat	275	ispunct	181
strncat	282	isspace	181
ConfigCNPullups	108	isupper	182
ConfigIntADC10	89	isxdigit	183
ConfigIntADC12	83	tolower	183
ConfigIntCANx	79	toupper	184
ConfigIntCapturex	111	Current Argument	200
ConfigIntCN	108	Customer Notification Service	xiv
ConfigIntDCI	135	Customer Support	xv
ConfigIntI2C	164	Customized Function	258
ConfigIntIOCx	117	D	
ConfigIntMCPWM	153	Dash (-)	237
ConfigIntQEI	148	DataRdyDCI	135
ConfigIntSplx	141	DataRdyI2C	165

dsPIC™ Language Tools Libraries

DataRdySPIx	142	DisableIntQEI	151
DataRdyUARTx	127	DisableIntSI2C	171
Date and Time	298	DisableIntSPIx	146
Date and Time Functions, See time.h		DisableIntTx	102
Day of the Month	292, 293, 298	DisableIntUxTX	132
Day of the Week	292, 293, 298	DisableINTx	110
Day of the Year	292, 298	div	246, 255
Daylight Savings Time	292, 296, 297	div_t	246
dbl_dig	185	Divide	
dbl_epsilon	186	Integer	255
dbl_mant_dig	186	Long Integer	260
dbl_max	186	Divide by Zero	195, 198, 256
dbl_max_10_exp	186	Document	
dbl_max_exp	186	Conventions	xii
dbl_min	187	Layout	xi
dbl_min_10_exp	187	Numbering Conventions	xii
dbl_min_exp	187	Updates	xii
DCI Functions		Domain Error 185, 300, 301, 302, 303, 305, 306, 308,	
Close DCI	134	309, 316, 318, 323, 324, 325, 326, 331, 332, 334, 335,	
Configure DCI	136	336, 337	
Configure DCI Interrupt	135	dot	231
DCI RX Buffer Status	135	Double Precision Floating Point	
DCI TX Buffer Status	134	Machine Epsilon	186
Example of Use	139	Maximum Exponent (base 10)	186
Read DCI RX Buffer	138	Maximum Exponent (base 2)	186
Write DCI TX Buffer	138	Maximum Value	186
DCI Macros		Minimum Exponent (base 10)	187
Disable DCI Interrupt	139	Minimum Exponent (base 2)	187
Enable DCI Interrupt	139	Minimum Value	187
Set DCI Interrupt Priority	139	Number of Binary Digits	186
DCT	53	Number of Decimal Digits	185
DCTIP	54	double Type	204
Deallocate Memory	258, 264	Dream Function	230
debugging logic errors	174	DSP Libraries	3
Decimal	231, 237, 267, 268	dsPIC Peripheral Libraries	63
Decimal Digit		E	
Defined	177	edom	185, 300
Number Of	185, 187, 189	Ellipses (...)	200, 237
Test for	177	Empty Binary File	214
Decimal Point	230	Empty Text File	214
Default Handler	194	EnableCNx	109
Diagnostics, See assert.h		EnableIntADC	87, 94
difftime	295	EnableIntCANx	80
Digit, Decimal, See Decimal Digit		EnableIntDCI	139
Digit, Hexadecimal, See Hexadecimal Digit		EnableIntFLTA	161
Direct Input/Output Functions		EnableIntFLTB	162
fread	217	EnableIntICx	115
fwrite	225	EnableIntIuTX	132
DisableCNx	109	EnableIntMCPWM	161
DisableIntADC	87, 94	EnableIntMI2C	171
DisableIntCANx	80	EnableIntOCx	124
DisableIntDCI	139	EnableIntQEI	151
DisableInterrupts	106	EnableIntSI2C	171
DisableIntFLTA	162	EnableIntSPIx	146
DisableIntFLTB	162	EnableIntTx	102
DisableIntIC1	115	EnableIntUxRX	132
DisableIntIuRX	132	EnableINTx	110
DisableIntMCPWM	161	End Of File	205
DisableIntMI2C	171	Indicator	204
DisableIntOCx	124	Seek	222

Test For	209	exponential function	
Environment Function		Double Floating Point	311
getenv	258	Single Floating Point	312
Environment Variable	342	F	
EOF	205	fabs	313
erange	185, 300	fabsf	314
errno	185, 300	fclose	208, 341
errno.h	185, 300, 343	feof	207, 209
edom	185	error	207, 210
erange	185	fflush	211, 346
errno	185	FFTComplex	56
Error Codes	185, 280	FFTComplexIP	58
Error Conditions	300	fgetc	211, 343
Error Handler	256	fgetpos	212, 342
Error Handling Functions		fgets	213, 343
clearerr	207	Field Width	230
feof	209	FILE	204
ferror	210	File Access Functions	
perror	229	fclose	208
Error Indicator	204	fflush	211
Error Indicators		fopen	213
Clearing	207, 235	freopen	219
End Of File	207, 213	setbuf	238
Error	207, 213	setvbuf	238
Test For	210	File Access Modes	204, 214
Error Signal	194	File Buffering	
Errors, See errno.h		Fully Buffered	204, 205
Errors, Testing For	185	Line Buffered	204, 205
Example of Use		Unbuffered	204, 205
ADC, 10-Bit	95	File Operations	
ADC, 12-Bit	88	Remove	234
CAN	81	Rename	234
DCI	139	File Positioning Functions	
I2C	172	fgetpos	212
Input Capture	116	fseek	221
Output Compare	125	fsetpos	223
PWM	163	ftell	224
QEI Functions	152	rewind	235
SPI	147	FILENAME_MAX	205
Timers	103	File-Position Indicator ...	204, 211, 212, 216, 218, 223, 225
UART	133	Files, Maximum Number Open	206
Exception Error	256	Filtering Functions	31
exit	241, 247, 249, 257, 340	FIR	34
EXIT_FAILURE	247	FIRDecimate	35
EXIT_SUCCESS	247	FIRDelayInit	35
exp	311	FIRInterpDelayInit	37
expf	312	FIRInterpolate	36
Exponential and Logarithmic Functions		FIRLattice	38
exp	311	FIRLMS	39
expf	312	FIRLMSNorm	40
frexp	319	FIRStruct	33
frexpf	320	FIRStructInit	42
ldexp	321	IIRCanonic	43
ldexpf	322	IIRCanonicInit	44
log	323	IIRLattice	44
log10	324	IIRLattice OCTAVE model	49
log10f	325	IIRLatticeInit	46
logf	326	IIRTransposed	47
modf	327	IIRTransposedInit	48
modff	328		

dsPIC™ Language Tools Libraries

FIR	34	flt_radix	189
FIRDecimate	35	FLT_RADIX Digit	
FIRDelayInit	35	Number Of	186, 188, 189
FIRInterpDelayInit	37	flt_rounds	189
FIRInterpolate	36	Flush	211, 257
FIRLattice	38	fmod	316
FIRLMS	39	fmodf	317
FIRLMSNorm	40	-fno-short-double	185, 186, 187, 204
FIRStruct	33	fopen	204, 213, 239, 343
FIRStructInit	42	FOPEN_MAX	206
flags	230	Form Feed	181
float.h	185	Format Specifiers	230, 236
dbl_dig	185	Formatted Input/Output Functions	
dbl_mant_dig	186	fprintf	215
dbl_max	186	fscanf	220
dbl_max_10_exp	186	Formatted I/O Routines	204
dbl_max_exp	186	Formatted Input/Output Functions	
dbl_min	187	printf	230
dbl_min_10_exp	187	scanf	236
dbl_min_exp	187	sprintf	239
flr_epsilon	187	sscanf	240
flt_dig	187	vfprintf	243
flt_epsilon	186	vprintf	244
flt_mant_dig	188	vsprintf	245
flt_max	188	Formatted Text	240
flt_max_10_exp	188	Printing	239
flt_max_exp	188	fpos_t	204
flt_min	188	fprintf	204, 215
flt_min_10_exp	188	fputc	216
flt_min_exp	188	fputs	217
flt_radix	189	fraction and exponent function	
flt_rounds	189	Double Floating Point	319
ldbl_dig	189	Single Floating Point	320
ldbl_epsilon	189	Fraction Digits	230
ldbl_mant_dig	189	fread	217, 343
ldbl_max	189	free	258
ldbl_max_10_exp	189	Free Memory	258
ldbl_max_exp	190	freopen	204, 219, 343
ldbl_min	190	frexp	319
ldbl_min_10_exp	190	frexpf	320
ldbl_min_exp	190	fscanf	204, 220
Floating Point		fseek	221, 242, 342
Limits	185	fsetpos	223, 242, 342
No Conversion	204	ftell	224, 342
Types, Properties Of	185	Full Buffering	238, 239
Floating Point, See float.h		Fully Buffered	204, 205
Floating-Point Error Signal	195	fwrite	225
floor	314	G	
Double Floating Point	314	getc	227
Single Floating Point	315	getchar	228
floorf	315	getcSPIx	146
flr_max	188	getcUARTx	131
flt_dig	187	getenv	258, 342
flt_epsilon	187	gets	229, 343
flt_mant_dig	188	getsSPIx	145
flt_max_10_exp	188	getsUARTx	130
flt_max_exp	188	GMT	296
flt_min	188	gmtime	296, 297
flt_min_10_exp	188	Graphical Character	
flt_min_exp	188	Defined	178

Test for	178	Configure CN Pull-Ups	108
Greenwich Mean Time	296	Configure INT	107
H		Disable INT	107
h modifier	231, 236	I/O Port Macros	
HammingInit	21	Disable CN Interrupts	109
Handler		Disable Interrupts	110
Default	194	Enable CN Interrupts	109
Error	256	Enable Interrupts	110
Interrupt	198	Set Interrupts Priority	110
Nested	194	I2C Functions	
Signal	194, 199	Acknowledge I2C	165
Signal Type	194	Close I2C.	164
Handling		Configure I2C	167
Interrupt Signal	198	Configure I2C Interrupt	164
HanningInit	22	Data Ready I2C	165
Header Files		Example of Use	172
assert.h	174	Idle I2C	165
ctype.h	175	Master I2C Get String	166
errno.h	185, 300, 343	Master I2C Put String	166
float.h	185	Not Acknowledge I2C	167
limits.h	190	Read Master I2C.	166
locale.h	193	Read Slave I2C	169
math.h	300	Restart I2C	168
setjmp.h	193	Slave I2C Get String	169
signal.h	194	Slave I2C Put String	169
stdarg.h	200	Start I2C	170
stddef.h	202	Stop I2C	170
stdio.h	203, 343, 344	Write Master I2C.	167
stdlib.h	246, 342, 345	Write Slave I2C	170
string.h	270	I2C Macros	
time.h	292, 345	Disable Master I2C Interrupt	171
Heap	341	Disable Slave I2C Interrupt	171
Helper Functions	339	Enable Master I2C Interrupt	171
Hexadecimal	231, 237, 267, 268	Enable Slave I2C Interrupt	171
Hexadecimal Conversion	230	Set Master I2C Interrupt Priority	171
Hexadecimal Digit		Set Slave I2C Interrupt Priority	172
Defined	183	Idexp	321
Test for	183	Idexpf	322
Horizontal Tab	181	IdleI2C	165
Hour	292, 293, 298	IFFTComplex	59
HUGE_VAL	300	IFFTComplexIP	60
hyperbolic cosine		Ignore Signal	195
Double Floating Point	310	IIRCanonic	43
Single Floating Point	311	IIRCanonicInit	44
Hyperbolic Functions		IIRLattice	44
cosh	310	IIRLattice OCTAVE model	49
coshf	311	IIRLatticeInit	46
sinh	332	IIRTransposed	47
sinhf	333	IIRTransposedInit	48
tanh	337	Illegal Instruction Signal	196
tanhf	338	Implementation-Defined Limits, See limits.h	
hyperbolic sine		Indicator	
Double Floating Point	332	End Of File	204, 205
Single Floating Point	333	Error	204, 210
hyperbolic tangent		File Position ...	204, 211, 212, 216, 218, 223, 225
Double Floating Point	337	Infinity	300
Single Floating Point	338	Input and Output, See stdio.h	
I		Input Capture Functions	
I/O Port Functions		Close Input Capture	111
Configure CN Interrupts	108	Configure Input Capture	113

dsPIC™ Language Tools Libraries

Configure Input Capture Interrupt	111	LCD, External	64
Example of Use	116	Busy	64
Read All Input Captures	114	Example of Use	69
Input Capture Macros		Open	65
Disable Capture Interrupt	115	Put String	66
Enable Capture Interrupt	115	Read Address	66
Set Capture Interrupt Priority	115	Read Data	66
Input Formats	204	Set Character Generator Address	67
Instruction Cycles	296, 297, 299	Set Display Data Address	67
int		Write Command	68
Maximum Value	191	Write Data	67
Minimum Value	191	lconv, struct	193
INT_MAX	191	ldbl_dig	189
INT_MIN	191	ldbl_epsilon	189
Integer Limits	190	ldbl_mant_dig	189
Internal Error Message	280	ldbl_max	189
Internet Address	xiv	ldbl_max_10_exp	189
Interrupt Handler	198	ldbl_max_exp	190
Interrupt Signal	196	ldbl_min	190
Interrupt Signal Handling	198	ldbl_min_10_exp	190
Interruption Message	196	ldbl_min_exp	190
Invalid Executable Code Message	196	ldiv	247, 260
Invalid Storage Request Message	197	ldiv_t	246
Inverse Cosine, See arccosine		Leap Second	292, 298
Inverse Sine, See arcsine		Left-Justify	230
Inverse Tangent, See arctangent		libpic30.a, Rebuilding	339
isalnum	175	Libraries	
isBOR	103	DSP	3
isctrl	177	dsPIC Peripheral	63
isdigit	177	Standard C	173
isgraph	178	Standard C Math	300
isalpha	176	Limits	
islower	179	Floating Point	185
isLVD	104	Integer	190
isMCLR	104	limits.h	190
isPOR	104	CHAR_BITS	190
isprint	180	CHAR_MAX	190
ispunct	181	CHAR_MIN	190
isspace	181	INT_MAX	191
isupper	182	INT_MIN	191
isWDTTO	104	LLONG_MAX	191
isWDTWU	105	LLONG_MIN	191
isWU	105	LONG_MAX	191
isxdigit	183	LONG_MIN	191
J		MB_LEN_MAX	191
jmp_buf	193	SCHAR_MAX	192
Justify	230	SCHAR_MIN	192
K		SHRT_MAX	192
KaiserInit	23	SHRT_MIN	192
L		UCHAR_MAX	192
L modifier	231, 236	UINT_MAX	192
l modifier	231, 236	ULLONG_MAX	192
L_tmpnam	206, 241	ULONG_MAX	192
labs	259	USHRT_MAX	193
LC_ALL	193	Line Buffered	204, 205
LC_COLLATE	193	Line Buffering	239
LC_CTYPE	193	ll modifier	231, 236
LC_MONETARY	193	LLONG_MAX	191
LC_NUMERIC	193	LLONG_MIN	191
LC_TIME	193	load exponent function	

Double Floating Point	321	MasterWriteI2C	167
Single Floating Point	322	Math Exception Error	256
Local Time	295, 297	math.h	300
Locale, C	175, 193	acos.	300
Locale, Other	193	acosf.	301
locale.h	193	asin.	302
localeconv	193	asinf.	303
Localization, See locale.h		atan.	303
localtime	295, 296, 297	atan2.	305
Locate Character	276	atan2f.	306
log	323	atanf.	304
log10	324	ceil	307
log10f	325	ceilf	308
logarithm function		cos	308
Double Floating Point	324	cosf	309
Single Floating Point	325	cosh	310
logarithm function, natural		coshf	311
Double Floating Point	323	exp	311
Single Floating Point	326	expf	312
logf	326	fabs	313
logic errors, debugging	174	fabsf	314
Long Double Precision Floating Point		floor	314
Machine Epsilon	189	floorf	315
Maximum Exponent (base 10)	189	fmod	316
Maximum Exponent (base 2)	190	fmodf	317
Maximum Value	189	frexp	319
Minimum Exponent (base 10)	190	frexpf	320
Minimum Exponent (base 2)	190	HUGE_VAL.	300
Minimum Value	190	ldexp	321
Number of Binary Digits	189	ldexpf	322
Number of Decimal Digits	189	log	323
long double Type	204	log10	324
long int		log10f	325
Maximum Value	191	logf	326
Minimum Value	191	modf	327
long long int		modff	328
Maximum Value	191	pow	329
Minimum Value	191	powf	330
long long unsigned int		sin	331
Maximum Value	192	sinf	332
long unsigned int		sinh	332
Maximum Value	192	sinhf	333
LONG_MAX	191	sqrt	334
LONG_MIN	191	sqrtf	335
longjmp	194	tan	336
Lower Case Alphabetic Character		tanf	337
Convert To	183	tanh	337
Defined	179	tanhf	338
Test for	179	Mathematical Functions, See math.h	
M		Matrix Functions	24
Machine Epsilon		MatrixAdd	26
Double Floating Point	186	MatrixInvert	30
Long Double Floating Point	189	MatrixMultiply	27
Single Floating Point	187	MatrixScale	28
Magnitude	300, 312, 313, 316, 318, 332, 333	MatrixSubtract	28
malloc	258, 261, 340, 344	MatrixTranspose	29
Mapping Characters	175	Maximum	
MastergetsI2C	166	Multibyte Character	247
MasterputsI2C	166	Maximum Value	
MasterReadI2C	166	Double Floating Point Exponent (base 10)	186

Double Floating Point Exponent (base 2)	186
Long Double Floating Point Exponent (base 10) .	189
Long Double Floating Point Exponent (base 2) ...	190
Multibyte Character	191
rand	247
Single Floating Point Exponent (base 10)	188
Single Floating Point Exponent (base 2)	188
Type char	190
Type Double	186
Type int	191
Type Long Double	189
Type long int	191
Type long long int	191
Type long long unsigned int	192
Type long unsigned int	192
Type short int	192
Type signed char	192
Type Single	188
Type unsigned char	192
Type unsigned int	192
Type unsigned short int	193
MB_CUR_MAX	247
MB_LEN_MAX	191
mblen	262
mbstowcs	262
mbtowc	262
memchr	270
memcmp	271
memcpy	273
memmove	273
Memory	
Allocate	255, 261
Deallocate	258
Free	258
Reallocate	264
memset	274
Message	
Arithmetic Error	195
Interrupt	196
Invalid Executable Code	196
Invalid Storage Request	197
Termination Request	197
Microchip Internet Web Site	xiv
Minimum Value	
Double Floating Point Exponent (base 10)	187
Double Floating Point Exponent (base 2)	187
Long Double Floating Point Exponent (base 10) .	190
Long Double Floating Point Exponent (base 2) ...	190
Single Floating Point Exponent (base 10)	188
Single Floating Point Exponent (base 2)	188
Type char	190
Type Double	187
Type int	191
Type Long Double	190
Type long int	191
Type long long int	191
Type short int	192
Type signed char	192
Type Single	188
Type unsigned char	192
Type unsigned int	192
Type unsigned short int	193
Minute	292, 293, 298
mktime	297
modf	327
modff	328
modulus function	
Double Floating Point	327
Single Floating Point	328
Month	292, 293, 298
-msmart-io	204
Multibyte Character	247, 262, 269
Maximum Value	191
Multibyte String	262, 269
N	
NaN	300
NDEBUG	174
Nearest Integer Functions	
ceil	307
ceilf	308
floor	314
floorf	315
Nested Signal Handler	194
Newline	181, 203, 204, 213, 218, 229, 234
No Buffering	204, 205, 238, 239
Non-Local Jumps, See setjmp.h	
NotAckI2C	167
NULL	193, 202, 206, 247, 270, 293
O	
Octal	231, 237, 267, 268
Octal Conversion	230
offsetof	202
open	342
OpenADC10	90
OpenADC12	83
OpenCapturex	113
OpenDCI	136
OpenI2C	167
OpenMCPWM	154
OpenOCx	119
OpenQEI	149
OpenSPIx	143
OpenTimerx	98
OpenTimerxx	99
OpenUART	128
OpenXLCD	65
Output Compare Functions	
Close Compare	117
Configure Compare	119
Configure Compare Interrupt	117
Example of Use	125
Read Compare Duty Cycle	121
Read Compare Duty Cycle - PWM mode	120
Set Compare Duty Cycle	123
Set Compare Duty Cycle - PWM mode	122
Output Compare Macros	
Disable Compare Interrupt	124
Enable Compare Interrupt	124
Set Compare Interrupt Priority	125

Output Formats	204
Overflow Errors	185, 300, 312, 321, 322, 329, 330
Overlap	273, 275, 279, 282, 285
P	
Pad Characters	230
Percent	231, 236, 237, 299
Peripheral Libraries, dsPIC	63
perror	229
pic30-libs	339
_exit	339
brk	340
close	341
getenv	342
open	342
read	343
remove	343
rename	344
sbrk	344
seek	342
system	345
time	345
write	345
Plus Sign	230
Pointer, Temporary	264
PORStatReset	106
pow	329
power function	
Double Floating Point	329
Single Floating Point	330
Power Functions	
pow	329
powf	330
powf	330
precision	231
Prefix	230
prefix	183
Print Formats	204
Printable Character	
Defined	180
Test for	180
printf	204, 230
Processor Clocks per Second	293
Processor Time	292, 294
Pseudo-Random Number	264, 265
ptrdiff_t	202
Punctuation Character	
Defined	181
Test for	181
Pushed Back	242
putc	232
putchar	233
putcSPIx	146
putcUARTx	131
putsXLCD	66
puts	234
putsSPIx	145
putsUARTx	131
putsXLCD	66
PvoverrideMCPWM	156
PWM Functions	

Close PWM	153
Configure Override	156
Configure PWM	154
Configure PWM Interrupt	153
Example of Use	163
Set PWM Dead Time Assignment	158
Set PWM Dead Time Generation	158
Set PWM Duty Cycle	157
Set PWM FaultA.	159
Set PWM FaultB.	160
PWM Macros	
Disable FLTA Interrupt	162
Disable FLTB Interrupt	162
Disable PWM Interrupt	161
Enable FLTA Interrupt	161
Enable FLTB Interrupt	162
Enable PWM Interrupt	161
Set FLTA Interrupt Priority	162
Set FLTB Interrupt Priority	162
Set PWM Interrupt Priority	161
Q	
QEI Functions	
Close QEI	148
Configure QEI	149
Configure QEI Interrupt	148
Example of Use	152
Read QEI Position Count	151
Write QEI Position Count	151
QEI Macros	
Disable QEI Interrupt	151
Enable QEI Interrupt	151
Set QEI Interrupt Priority	152
qsort	253, 262
Quick Sort	262
R	
Radix	189
raise	194, 195, 196, 197, 199
rand	264, 265
RAND_MAX	247, 264
Range	237
Range Error .. 185, 267, 268, 310, 311, 312, 313, 321, 322,	329, 330, 332, 333
read	343
ReadADC10	93
ReadADC12	86
ReadAddrXLCD	66
ReadCapture	114
ReadDataXLCD	66
ReadDCI	138
ReadDCOCxPWM	120
ReadQEI	151
ReadRegOCx	121
ReadSPIx	142
ReadTimerx	100
ReadTimerxx	101
ReadUARTx	129
realloc	258, 264
Reallocate Memory	264
Rebuilding the libpic30.a library	339
References	xiii

dsPIC™ Language Tools Libraries

Registered Functions	249, 257	SetDCOCxPWM	122
remainder		SetDDRamAddr	67
Double Floating Point	316	setjmp	193
Single Floating Point	317	setjmp.h	193
Remainder Functions		jmp_buf	193
fmod	316	longjmp	194
fmodf	317	setjmp	193
remove	234, 343	setlocale	193
rename	234, 344	SetMCPWMDeadTimeAssignment	158
Reset	248, 269	SetMCPWMDeadTimeGeneration	158
Reset File Pointer	235	SetMCPWMFaultA	159
Reset/Control Functions		SetMCPWMFaultB	160
Low Voltage Detect	104	SetPointIntUxRX	132
Master Clear Reset	104	SetPriorityIntADC	87, 94
Reset from Brown-out	103	SetPriorityIntCANx	81
Reset from Power-on	104	SetPriorityIntDCI	139
Wake-up from Sleep	105	SetPriorityIntFLTA	162
Watchdog Timer Time-out	104	SetPriorityIntFLTB	162
Watchdog Timer Wake-up	105	SetPriorityIntICx	115, 125
Reset/Control Macros		SetPriorityIntMCPWM	161
Disable All Interrupts	106	SetPriorityIntMI2C	171
Disable Watchdog Timer	106	SetPriorityIntQEI	152
Enable Watchdog Timer	106	SetPriorityIntSI2C	172
Reset BOR bit	106	SetPriorityIntSPIx	146
Reset POR bit	106	SetPriorityIntTx	102
RestartI2C	168	SetPriorityIntUxTX	133
rewind	235, 242, 342	SetPriorityIntx	110
Rounding Mode	189	SetPulseOCx	123
S		setvbuf	204, 205, 238
sainf	303	short int	
sbrk	341, 344	Maximum Value	192
Scan Formats	204	Minimum Value	192
scanf	204, 236	SHRT_MAX	192
Scanning	240	SHRT_MIN	192
SCHAR_MAX	192	sig_atomic_t	194
SCHAR_MIN	192	SIG_DFL	194
Search Functions		SIG_ERR	194
memchr	270	SIG_IGN	195
strchr	276	SIGABRT	195
strcspn	279	SIGFPE	195
strpbrk	286	SIGILL	196
strrchr	287	SIGINT	196
strspn	288	Signal	
strstr	289	Abnormal Termination	195
strtok	290	Error	194
Second	292, 293, 295, 298	Floating-Point Error	195
Seed	264, 265	Ignore	195
Seek		Illegal Instruction	196
From Beginning of File	222	Interrupt	196
From Current Position	222	Reporting	197
From End Of File	222	Termination Request	197
seek	342	signal	195, 196, 197, 198, 199
SEEK_CUR	206, 222	Signal Handler	194, 199
SEEK_END	206, 222	Signal Handler Type	194
SEEK_SET	206, 222	Signal Handling, See signal.h	
setbuf	204, 205, 238	signal.h	194
SetCGRamAddr	67	raise	197
SetChanADC10	93	sig_atomic_t	194
SetChanADC12	86	SIG_DFL	194
SetDCMCPWM	157	SIG_ERR	194

SIG_IGN	195	Disable SPI Interrupt	146
SIGABRT	195	Enable SPI Interrupt	146
SIGFPE	195	Set SPI Interrupt Priority	146
SIGILL	196	sprintf	204, 239
signal	198	sqrt	334
SIGSEGV	197	sqrtf	335
SIGTERM	197	square root function	
signed char		Double Floating Point	334
Maximum Value	192	Single Floating Point	335
Minimum Value	192	Square Root Functions	
SIGSEGV	197	sqrt	334
SIGTERM	197	sqrtf	335
sim30 simulator	339	srand	265
sin	331	sscanf	204, 240
sine		Stack	341
Double Floating Point	331	Standard C Library	173
Single Floating Point	332	Standard C Locale	175
sinf	332	Standard Error	204, 206
singal.h		Standard Input	204, 206
SIGINIT	196	Standard Output	204, 207
Single Precision Floating Point		StartI2C	170
Machine Epsilon	187	Startup	204
Maximum Exponent (base 10)	188	Module, Alternate	1
Maximum Exponent (base 2)	188	Module, Primary	1
Maximum Value	188	stdarg.h	200
Minimum Exponent (base 10)	188	va_arg	200
Minimum Exponent (base 2)	188	va_end	201
Minimum Value	188	va_list	200
Number of Binary Digits	188	va_start	202
Number of Decimal Digits	187	stddef.h	202
sinh	332	NULL	202
sinhf	333	offsetof	202
size	231	ptrdiff_t	202
size_t	202, 205, 247, 270, 292	size_t	202
sizeof	202, 205, 247, 270, 292	wchar_t	202
SlavegetsI2C	169	stderr	175, 204, 206, 229
SlaveputsI2C	169	stdin	204, 206, 228, 229, 236
SlaveReadI2C	169	stdio.h	203, 343, 344
SlaveWriteI2C	170	_IOFBF	205
Sort, Quick	262	_IOLBF	205
Source File Name	175	_IONBF	205
Source Line Number	175	BUFSIZ	205
Space	230	clearerr	207
Space Character		EOF	205
Defined	181	fclose	208
Test for	181	feof	209
Specifiers	230, 236	ferror	210
SPI Functions		fflush	211
Close SPI	141	fgetc	211
Configure SPI	143	fgetpos	212
Configure SPI Interrupt	141	fgets	213
Example of Use	147	FILE	204
Read SPI RX Buffer	142	FILENAME_MAX	205
SPI Buffer Status	142	fopen	213
SPI Get Character	146	FOPEN_MAX	206
SPI Get String	145	fpos_t	204
SPI Put Character	146	fprintf	215
SPI Put String	145	fputc	216
Write SPI TX Buffer	142	fputs	217
SPI Macros		fread	217

freopen	219	mbstowcs	262
fscanf	220	mbtowc	262
fseek	221	NULL	247
fsetpos	223	qsort	262
ftell	224	rand	264
fwrite	225	RAND_MAX	247
getc	227	realloc	264
getchar	228	size_t	247
gets	229	srand	265
L_tmpnam	206	strtod	266
NULL	206	strtol	267
perror	229	strtoul	268
printf	230	system	269
putc	232	wchar_t	247
putchar	233	wctomb	269
puts	234	wxstombs	269
remove	234	stdout	204, 206, 207, 230, 233, 234
rename	234	StopI2C	170
rewind	235	StopSampADC10	93
scanf	236	StopSampADC12	86
SEEK_CUR	206	strcat	275
SEEK_END	206	strchr	276
SEEK_SET	206	strcmp	277
setbuf	238	strcoll	278
setvbuf	238	strcpy	279
size_t	205	strcspn	279
sprintf	239	Streams	203
sscanf	240	Binary	203
stderr	206	Buffering	238
stdin	206	Closing	208, 257
stdout	207	Opening	213
TMP_MAX	207	Reading From	227
tmpfile	241	Text	203
tmpnam	241	Writing To	225, 232
ungetc	242	strerror	280
vfprintf	243	strftime	298
vprintf	244	String	
vsprintf	245	Length	281
stdlib.h	246, 342, 345	Search	289
abort	248	Transform	292
abs	248	String Functions, See string.h	
atexit	249	string.h	270
atof	251	memchr	270
atoi	251	memcmp	271
atol	252	memcpy	273
bsearch	253	memmove	273
calloc	255	memset	274
div	255	NULL	270
div_t	246	size_t	270
exit	257	strcat	275
EXIT_FAILURE	247	strchr	276
EXIT_SUCCESS	247	strcmp	277
free	258	strcoll	278
getenv	258	strcpy	279
labs	259	strcspn	279
ldiv	260	strerror	280
ldiv_t	246	strlen	281
malloc	261	strncat	282
MB_CUR_MAX	247	strncmp	283
mblen	262	strncpy	285

strpbrk	286	NULL	293
strchr	287	size_t	292
strspn	288	strftime	298
strstr	289	struct tm	292
strtok	290	time	299
strxfrm	292	time_t	293
strlen	281	time_t	293, 297, 299
strncat	282	Timer Functions	
strncmp	283	Close	96
strncpy	285	Close 32-bit	96
strpbrk	286	Configure Interrupt	97
strchr	287	Configure Interrupt 32-bit	97
strspn	288	Example of Use	103
strstr	289	Open	98
strtod	251, 266	Open 32-bit	99
strtok	290	Read	100
strtol	252, 267	Read 32-bit	101
strtoul	268	Write	101
struct lconv	193	Write 32-bit	101
struct tm	292	Timer Macros	
strxfrm	292	Disable Interrupt	102
Substrings	291	Enable Interrupt	102
Subtracting Pointers	202	Set Interrupt Priority	102
Successful Termination	247	TMP_MAX	207
system	269, 345	tmpfile	241
T		tmpnam	241
Tab	181	Tokens	291
tan	336	tolower	183
tanf	337	toupper	184
tangent		Transferring Control	193
Double Floating Point	336	Transform Functions	50
Single Floating Point	337	BitReverseComplex	52
tanh	337	CosFactorInit	52
tanhf	338	DCT	53
Temporary		DCTIP	54
File	241, 257	FFTComplex	56
Filename	206, 241	FFTComplexIP	58
Pointer	264	IFFTComplex	59
Termination		IFFTComplexIP	60
Request Message	197	TwidFactorInit	62
Request Signal	197	Transform String	292
Successful	247	Trigonometric Functions	
Unsuccessful	247	acos	300
Text Mode	214	acosh	301
Text Streams	203	asin	302
Ticks	293, 294, 295	asinh	303
time	299, 345	atan	303
Time Difference	295	atan2	305
Time Structure	292, 298	atan2f	306
Time Zone	299	atanf	304
time.h	292, 345	cos	308
asctime	293	cosh	309
clock	294	sin	331
clock_t	292	sinh	332
CLOCKS_PER_SEC	293	tan	336
ctime	295	tanh	337
difftime	295	Troubleshooting	xiii
gmtime	296	TwidFactorInit	62
localtime	297	type	231, 237
mktime	297		

U

UART Functions

Close UART	126
Configure UART	128
Configure UART Interrupt	127
Example of Use	133
Read UART	129
UART Buffer Status	127
UART Get Character	131
UART Get String	130
UART Put Character	131
UART Put String	131
UART Status	126
Write UART	130

UART Macros

Disable UART RX Interrupt	132
Disable UART TX Interrupt	132
Enable UART RX Interrupt	132
Enable UART TX Interrupt	132
Set UART RX Interrupt Priority	132
Set UART TX Interrupt Priority	133

UCHAR_MAX

UINT_MAX

ULLONG_MAX

ULONG_MAX

Underflow Errors ... 185, 300, 312, 321, 322, 329, 330

ungetc

Universal Time Coordinated

unsigned char

Maximum Value	192
---------------------	-----

unsigned int

Maximum Value	192
---------------------	-----

unsigned short int

Maximum Value	193
---------------------	-----

Unsuccessful Termination

Upper Case Alphabetic Character

Convert To	184
Defined	182
Test for	182

USHRT_MAX

UTC

Utility Functions, See stdlib.h

V

va_arg

va_end

va_list

va_start

Variable Argument Lists, See stdarg.h

Variable Length Argument List 200, 202, 243, 244, 245

Vector Functions

VectorAdd

VectorConvolve

VectorCopy

VectorCorrelate

VectorDotProduct

VectorMax

VectorMin

VectorMultiply

VectorNegate

VectorPower

VectorScale

VectorSubtract

VectorWindow

VectorZeroPad

VERBOSE_DEBUGGING

Vertical Tab

vfprintf

vprintf

vsprintf

W

wchar_t

wcstombs

wctomb

WDTSWDisable

WDTSWEnable

Week

White Space

White-Space Character

Defined	181
---------------	-----

Test for	181
----------------	-----

wide

Wide Character

Wide Character String

Wide Character Value

Width

width

Window Functions

BartlettInit	20
--------------------	----

BlackmanInit	21
--------------------	----

HammingInit	21
-------------------	----

HanningInit	22
-------------------	----

KaiserInit	23
------------------	----

VectorWindow	23
--------------------	----

write

WriteCmdXLCD

WriteDataXLCD

WriteDCI

WriteQEI

WriteSP1x

WriteTimerx

WriteTimerxx

WriteUARTx

WWW Address

Y

Year

Z

Zero

Zero, divide by



WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office

2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: <http://www.microchip.com>

Atlanta

3780 Mansell Road, Suite 130
Alpharetta, GA 30022
Tel: 770-640-0034
Fax: 770-640-0307

Boston

2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848
Fax: 978-692-3821

Chicago

333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071
Fax: 630-285-0075

Dallas

4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423
Fax: 972-818-2924

Detroit

Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo

2767 S. Albright Road
Kokomo, IN 46902
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles

18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888
Fax: 949-263-1338

Phoenix

2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966
Fax: 480-792-4338

San Jose

1300 Terra Bella Avenue
Mountain View, CA 94043
Tel: 650-215-1444

Toronto

6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Australia

Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing

Unit 706B
Wan Tai Bei Hai Bldg.
No. 6 Chaoyangmen Bei Str.
Beijing, 100027, China
Tel: 86-10-85282100
Fax: 86-10-85282104

China - Chengdu

Rm. 2401-2402, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-86766200
Fax: 86-28-86766599

China - Fuzhou

Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506
Fax: 86-591-7503521

China - Hong Kong SAR

Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Shanghai

Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700
Fax: 86-21-6275-5060

China - Shenzhen

Rm. 1812, 18/F, Building A, United Plaza
No. 5022 Binhe Road, Futian District
Shenzhen 518033, China
Tel: 86-755-82901380
Fax: 86-755-8295-1393

China - Shunde

Room 401, Hongjian Building
No. 2 Fengxiangnan Road, Ronggui Town
Shunde City, Guangdong 528303, China
Tel: 86-765-8395507 Fax: 86-765-8395571

China - Qingdao

Rm. B505A, Fullhope Plaza,
No. 12 Hong Kong Central Rd.
Qingdao 266071, China
Tel: 86-532-5027355 Fax: 86-532-5027205

India

Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan

Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471-6166 Fax: 81-45-471-6122

Korea

168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5932 or
82-2-558-5934

Singapore

200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-6334-8870 Fax: 65-6334-8850

Taiwan

Kaohsiung Branch
30F - 1 No. 8
Min Chuan 2nd Road
Kaohsiung 806, Taiwan
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan

Taiwan Branch
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Austria

Durisolstrasse 2
A-4600 Wels
Austria
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

Denmark

Regus Business Centre
Lautrup høj 1-3
Ballerup DK-2750 Denmark
Tel: 45-4420-9895 Fax: 45-4420-9910

France

Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - 1er Etage
91300 Massy, France
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany

Steinheilstrasse 10
D-85737 Ismaning, Germany
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy

Via Quasimodo, 12
20025 Legnano (MI)
Milan, Italy
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands

P. A. De Biesbosch 14
NL-5152 SC Drunen, Netherlands
Tel: 31-416-690399
Fax: 31-416-690340

United Kingdom

505 Eskdale Road
Wokingham
Berkshire, England RG41 5TU
Tel: 44-118-921-5869
Fax: 44-118-921-5820

11/24/03