



16-BIT LANGUAGE TOOLS LIBRARIES

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

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

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

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. 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 Microchip intellectual property rights.

Trademarks

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

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

Analog-for-the-Digital Age, Application Maestro, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, Linear Active Thermistor, MPASM, MPLIB, MPLINK, MPSIM, PICKit, PICDEM, PICDEM.net, PICLAB, PICTail, PowerCal, PowerInfo, PowerMate, PowerTool, Real ICE, rFLAB, rfPICDEM, Select Mode, Smart Serial, SmartTel, Total Endurance, UNI/O, WiperLock and Zena are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

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

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

© 2005, 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, nonvolatile 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	1
Chapter 1. Library Overview	
1.1 Introduction	7
1.2 OMF-Specific Libraries/Start-up Modules	7
1.3 Start-up Code	8
1.4 DSP Library	8
1.5 16-Bit Peripheral Libraries	8
1.6 Standard C Libraries (with Math Functions)	8
1.7 MPLAB C30 Built-in Functions	8
Chapter 2. DSP Library	
2.1 Introduction	9
2.2 Using the DSP Library	10
2.3 Vector Functions	13
2.4 Window Functions	26
2.5 Matrix Functions	31
2.6 Filtering Functions	38
2.7 Transform Functions	58
2.8 Control Functions	73
2.9 Miscellaneous Functions	78
Chapter 3. 16-Bit Peripheral Libraries	
3.1 Introduction	81
3.2 Using the 16-Bit Peripheral Libraries	82
3.3 External LCD Functions	82
3.4 CAN Functions	89
3.5 ADC12 Functions	103
3.6 ADC10 Functions	110
3.7 Timer Functions	117
3.8 Reset/Control Functions	125
3.9 I/O Port Functions	128
3.10 Input Capture Functions	133
3.11 Output Compare Functions	139
3.12 UART Functions	149
3.13 DCI Functions	158
3.14 SPI Functions	166
3.15 QEI Functions	174
3.16 PWM Functions	179
3.17 I2C™ Functions	191

16-Bit Language Tools Libraries

Chapter 4. Standard C Libraries with Math Functions

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

Chapter 5. MPLAB C30 Built-in Functions

5.1 Introduction	383
5.2 Built-In Function List	384

Appendix A. ASCII Character Set401

Index403

Worldwide Sales and Service424

Preface

NOTICE TO CUSTOMERS

All documentation becomes dated, and this manual is no exception. Microchip tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our web site (www.microchip.com) to obtain the latest documentation available.

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

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

INTRODUCTION

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

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

Items discussed in this chapter include:

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

16-Bit Language Tools Libraries

ABOUT THIS GUIDE

Document Layout

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

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

Conventions Used in this Guide

This manual uses the following documentation conventions:

DOCUMENTATION CONVENTIONS

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

16-Bit Language Tools Libraries

RECOMMENDED READING

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

README Files

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

dsPIC[®] Language Tools Getting Started (DS70094)

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

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

A guide to using the 16-bit assembler, MPLAB ASM30, 16-bit linker, MPLAB LINK30 and various 16-bit utilities, including MPLAB LIB30 archiver/librarian.

MPLAB[®] C30 C Compiler User's Guide (DS51284)

A guide to using the 16-bit C compiler. MPLAB LINK30 is used with this tool.

dsPIC30F Family Overview (DS70043)

An overview of the dsPIC30F devices and architecture.

dsPIC30F/33F Programmer's Reference Manual (DS70157)

Programmer's guide to dsPIC30F/33F devices. Includes the programmer's model and instruction set.

Microchip Web Site

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

TROUBLESHOOTING

See the README files for information on common problems not addressed in this document.

THE MICROCHIP WEB SITE

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

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

DEVELOPMENT SYSTEMS CUSTOMER CHANGE NOTIFICATION SERVICE

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

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

The Development Systems product group categories are:

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

16-Bit Language Tools Libraries

CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

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

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

Technical support is available through the web site at: <http://support.microchip.com>

Chapter 1. Library Overview

1.1 INTRODUCTION

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

1.1.1 Assembly Code Applications

Free versions of the 16-bit language tool libraries are available from the Microchip web site. DSP and 16-bit peripheral libraries are provided with object files and source code. A math library containing functions from the standard C header file `<math.h>` is provided as an object file only. The complete standard C library is provided with the MPLAB C30 C compiler.

1.1.2 C Code Applications

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

1.1.3 Chapter Organization

This chapter is organized as follows:

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

1.2 OMF-SPECIFIC LIBRARIES/START-UP MODULES

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

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

There are two ways to select the OMF:

1. Set an environment variable called `PIC30_OMF` for all tools.
2. Select the OMF on the command line when invoking the tool, i.e., `-omf=omf` or `-momf=omf`.

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

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

16-Bit Language Tools Libraries

1.3 START-UP CODE

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

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

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

1.4 DSP LIBRARY

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

1.5 16-BIT PERIPHERAL LIBRARIES

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

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

1.6 STANDARD C LIBRARIES (WITH MATH FUNCTIONS)

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

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

A typical C application will require all three libraries.

1.7 MPLAB C30 BUILT-IN FUNCTIONS

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

Chapter 2. DSP Library

2.1 INTRODUCTION

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

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

2.1.1 Assembly Code Applications

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

2.1.2 C Code Applications

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

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

2.1.3 Chapter Organization

This chapter is organized as follows:

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

16-Bit Language Tools Libraries

2.2 USING THE DSP LIBRARY

2.2.1 Building with the DSP Library

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

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

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

2.2.2 Memory Models

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

2.2.3 DSP Library Function Calling Convention

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

The working registers W0 to W7 are treated as scratch memory, and their values may not be preserved after the function call. On the other hand, if any of the working registers W8 to W13 are used by a function, the working register is first saved, the register is used and then its original value is restored upon function return. The return value of a (non void) function is available in working register W0 (also referred to as WREG). When needed, the run time software stack is used following the C system stack rules described in the “MPLAB[®] C30 Compiler User’s Guide”. Based on these guidelines, the object modules of the DSP Library can be linked to either a C program, an assembly program or a program which combines code in both languages.

2.2.4 Data Types

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

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

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

The `fractional` data type is used to represent data that has 1 sign bit, and 15 fractional bits. Data which uses this format is commonly referred to as “1.15” data.

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

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

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

2.2.5 Data Memory Usage

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

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

2.2.6 CORCON Register Usage

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

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

- DSP multipliers 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

16-Bit Language Tools Libraries

For a detailed explanation of the CORCON register and its effects, refer to the “dsPIC30F Family Reference Manual” (DS70046).

2.2.7 Overflow and Saturation Handling

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

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

2.2.8 Integrating with Interrupts and an RTOS

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

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

2.2.9 Rebuilding the DSP Library

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

2.3 VECTOR FUNCTIONS

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

2.3.1 Fractional Vector Operations

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

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

The one dimensional arrangement of a vector accommodates to the memory storage model of the device, so that the 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 16-bit device.

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

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

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

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

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

- a) The sum of sizes of all the vectors involved in a particular operation falls within the range of available data memory for the target device.
- b) In the case of binary operations, the cardinalities of both operand vectors *must* obey the rules of vector algebra (particularly, see remarks for the `VectorConvolve` and `VectorCorrelate` functions).
- c) The destination vector *must* be large enough to accept the results of an operation.

2.3.2 User Considerations

- a) No boundary checking is performed by these functions. Out of range cardinalities (including zero length vectors) as well as nonconforming use of source vector sizes in binary operations may produce unexpected results.
- b) The vector addition and subtraction operations could lead to saturation if the sum of corresponding elements in the source vector(s) is greater than $1-2^{-15}$ or smaller than -1.0 . Analogously, the vector dot product and power operations could lead to saturation if the sum of products is greater than $1-2^{-15}$ or smaller than -1.0 .
- c) It is recommended that the STATUS Register (SR) be examined after completion of each function call. In particular, users can inspect the SA, SB and SAB flags after the function returns to determine if saturation occurred.
- d) All the functions have been designed to operate on fractional vectors allocated in default RAM memory space (X-Data or Y-Data).
- e) Operations which return a destination vector can be nested, so that for instance if:
a = Op1 (b, c), with b = Op2 (d), and c = Op3 (e, f), then
a = Op1 (Op2 (d), Op3 (e, f))

2.3.3 Additional Remarks

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

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

As another example, you may be interested in replacing `numElems` elements of a destination vector located between `N` and `N+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.

2.3.4 Individual Functions

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

VectorAdd

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

Include: `dsp.h`

Prototype:

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

Arguments:

<code>numElems</code>	number of elements in source vectors
<code>dstV</code>	pointer to destination vector
<code>srcV1</code>	pointer to source one vector
<code>srcV2</code>	pointer to source two vector

Return Value: Pointer to base address of destination vector.

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

Source File: `vadd.s`

Function Profile: System resources usage:

<code>W0..W4</code>	used, not restored
<code>ACCA</code>	used, not restored
<code>CORCON</code>	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):
 $17 + 3(numElems)$

16-Bit Language Tools Libraries

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:

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

Return Value: Pointer to base address of destination vector.

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

Source File: `vcon.s`

VectorConvolve (Continued)

Function Profile: System resources usage:

W0..W7	used, not restored
W8..W10	saved, used, restored
ACCA	used, not restored
CORCON	saved, used, restored

DO and REPEAT instruction usage:

- 2 level DO instructions
- no REPEAT instructions

Program words (24-bit instructions):

58

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

For $N = numElements1$, and $M = numElements2$,

$$28 + 13M + 6 \sum_{m=1}^M m + (N - M)(7 + 3M), \text{ for } M < N$$

$$28 + 13M + 6 \sum_{m=1}^M m, \text{ for } M = N$$

VectorCopy

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

Include: `dsp.h`

Prototype:

```
extern fractional* VectorCopy (
    int numElements,
    fractional* dstV,
    fractional* srcV
);
```

Arguments:

<code>numElements</code>	number of elements in source 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: The destination vector *must* already exist. Destination vectors *must* have, at least, `numElements` elements, but could be longer. This function can be computed in place. See Additional Remarks at the end of the section for comments on this mode of operation.

Source File: `vcopy.s`

Function Profile: System resources usage:

W0..W3	used, not restored
--------	--------------------

DO and REPEAT instruction usage:

- no DO instructions
- 1 level REPEAT instructions

Program words (24-bit instructions):

6

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

$12 + numElements$

16-Bit Language Tools Libraries

VectorCorrelate

Description:	<p>VectorCorrelate computes the correlation between two source vectors, and stores the result in a destination vector. The result is computed as follows:</p> $r(n) = \sum_{k=0}^{N-1} x(k)y(k+n), \text{ for } 0 \leq n < N + M - 1$ <p>where $x(k)$ = source one vector of size N, $y(k)$ = source two vector of size M (with $M \leq N$).</p>										
Include:	<code>dsp.h</code>										
Prototype:	<pre>extern fractional* VectorCorrelate (int numElems1, int numElems2, fractional* dstV, fractional* srcV1, fractional* srcV2);</pre>										
Arguments:	<table><tr><td><code>numElems1</code></td><td>number of elements in source one vector</td></tr><tr><td><code>numElems2</code></td><td>number of elements in source two vector</td></tr><tr><td><code>dstV</code></td><td>pointer to destination vector</td></tr><tr><td><code>srcV1</code></td><td>pointer to source one vector</td></tr><tr><td><code>srcV2</code></td><td>pointer to source two vector</td></tr></table>	<code>numElems1</code>	number of elements in source one vector	<code>numElems2</code>	number of elements in source two vector	<code>dstV</code>	pointer to destination vector	<code>srcV1</code>	pointer to source one vector	<code>srcV2</code>	pointer to source two vector
<code>numElems1</code>	number of elements in source one vector										
<code>numElems2</code>	number of elements in source two vector										
<code>dstV</code>	pointer to destination vector										
<code>srcV1</code>	pointer to source one vector										
<code>srcV2</code>	pointer to source two vector										
Return Value:	Pointer to base address of destination vector.										
Remarks:	<p>The number of elements in the source two vector <i>must</i> be less than or equal to the number of elements in the source one vector.</p> <p>The destination vector <i>must</i> already exist, with exactly <code>numElems1+numElems2-1</code> number of elements.</p> <p>This function can be self applicable.</p> <p>This function uses VectorConvolve.</p>										
Source File:	<code>vcor.s.s</code>										
Function Profile:	<p>System resources usage:</p> <table><tr><td><code>W0..W7</code></td><td>used, not restored,</td></tr></table> <p>plus resources from VectorConvolve</p> <p>DO and REPEAT instruction usage:</p> <table><tr><td>1 level DO instructions</td></tr><tr><td>no REPEAT instructions,</td></tr><tr><td>plus DO/REPEAT instructions from</td></tr></table> <code>VectorConvolve</code>	<code>W0..W7</code>	used, not restored,	1 level DO instructions	no REPEAT instructions,	plus DO/REPEAT instructions from					
<code>W0..W7</code>	used, not restored,										
1 level DO instructions											
no REPEAT instructions,											
plus DO/REPEAT instructions from											
	<p>Program words (24-bit instructions):</p> <table><tr><td>14,</td></tr></table> <p>plus program words from VectorConvolve</p>	14,									
14,											
	<p>Cycles (including C-function call and return overheads):</p> <table><tr><td>$19 + \text{floor}(M / 2) * 3$, with $M = \text{numElems2}$,</td></tr></table> <p>plus cycles from VectorConvolve.</p>	$19 + \text{floor}(M / 2) * 3$, with $M = \text{numElems2}$,									
$19 + \text{floor}(M / 2) * 3$, with $M = \text{numElems2}$,											
	<p>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.</p>										

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:

<code>numElems</code>	number of elements in source vectors
<code>srcV1</code>	pointer to source one vector
<code>srcV2</code>	pointer to source two vector

Return Value: Value of the sum of products.

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

Source File: `vdot.s`

Function Profile: System resources usage:

W0..W2	used, not restored
W4..W5	used, not restored
ACCA	used, not restored
CORCON	saved, used, restored

DO and REPEAT instruction usage:

1 level DO instructions
no REPEAT instructions

Program words (24-bit instructions):

13

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

$17 + 3(numElems)$

VectorMax

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

Include: `dsp.h`

Prototype:

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

Arguments:

<code>numElems</code>	number of elements in source vector
<code>srcV</code>	pointer to source vector
<code>maxIndex</code>	pointer to holder for index of (last) maximum element

Return Value: Maximum value in vector.

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

Source File: `vmax.s`

16-Bit Language Tools Libraries

VectorMax (Continued)

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

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

Program words (24-bit instructions):
13

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

VectorMin

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

Include: dsp.h

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

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

Return Value: Minimum value in vector.

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

Source File: vmin.s

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

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

Program words (24-bit instructions):
13

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

VectorMultiply

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

Include: dsp.h

Prototype:

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

Arguments:

<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.

Remarks: This operation is also known as vector element-by-element multiplication.

This function can be computed in place.

This function can be self applicable.

Source File: vmul.s

Function Profile: System resources usage:

W0..W5	used, not restored
ACCA	used, not restored
CORCON	saved, used, restored

DO and REPEAT instruction usage:

1 level DO instructions

no REPEAT instructions

Program words (24-bit instructions):

14

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

$17 + 4(numElems)$

VectorNegate

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

Include: dsp.h

Prototype:

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

Arguments:

<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: vneg.s

16-Bit Language Tools Libraries

VectorNegate (Continued)

Function Profile: System resources usage:

W0..W5	used, not restored
ACCA	used, not restored
CORCON	saved, used, restored

DO and REPEAT instruction usage:

- 1 level DO instructions
- no REPEAT instructions

Program words (24-bit instructions):

16

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

$19 + 4(numElems)$

VectorPower

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

Include: `dsp.h`

Prototype:

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

Arguments:

<code>numElems</code>	number of elements in source vector
<code>srcV</code>	pointer to source vector

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

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

Source File: `vpow.s`

Function Profile: System resources usage:

W0..W2	used, not restored
W4	used, not restored
ACCA	used, not restored
CORCON	saved, used, restored

DO and REPEAT instruction usage:

- no DO instructions
- 1 level REPEAT instructions

Program words (24-bit instructions):

12

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

$16 + 2(numElems)$

VectorScale

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

Include: `dsp.h`

Prototype:

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

Arguments:

<code>numElems</code>	number of elements in source vector
<code>dstV</code>	pointer to destination vector
<code>srcV</code>	pointer to source vector
<code>sclVal</code>	value by which to scale vector elements

Return Value: Pointer to base address of destination vector.

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

Source File: `vscl.s`

Function Profile: System resources usage:

W0..W5	used, not restored
ACCA	used, not restored
CORCON	saved, used, restored

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

Program words (24-bit instructions):
 14

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

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:

<code>numElems</code>	number of elements in source vectors
<code>dstV</code>	pointer to destination vector
<code>srcV1</code>	pointer to source one vector (minuend)
<code>srcV2</code>	pointer to source two vector (subtrahend)

Return Value: Pointer to base address of destination vector.

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

16-Bit Language Tools Libraries

VectorSubtract (Continued)

Source File: vsub.s

Function Profile: System resources usage:

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

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

Program words (24-bit instructions):
14

Cycles (including C-function call and return overheads):
 $17 + 4(numElements)$

VectorZeroPad

Description: VectorZeroPad copies the source vector into the beginning of the (already existing) destination vector, and then fills with zeros the remaining `numZeros` elements of destination vector:
 $dstV[n] = srcV[n], 0 \leq n < numElements$
 $dstV[n] = 0, numElements \leq n < numElements + numZeros$

Include: dsp.h

Prototype:

```
extern fractional* VectorZeroPad (  
    int numElements,  
    int numZeros,  
    fractional* dstV,  
    fractional* srcV  
);
```

Arguments:

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

Source File: vzpad.s

VectorZeroPad (Continued)

Function Profile: System resources usage:
W0..W6 used, not restored
plus resources from `VectorCopy`

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

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

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

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

16-Bit Language Tools Libraries

2.4 WINDOW FUNCTIONS

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

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

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

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

2.4.1 User Considerations

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

2.4.2 Individual Functions

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

BartlettInit

Description:	BartlettInit initializes a Bartlett window of length <i>numElems</i> .
Include:	dsp.h
Prototype:	<pre>extern fractional* BartlettInit (int numElems, fractional* window);</pre>
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 <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: *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: 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.txt" in pic30_tools\src\dsp for this information.

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

16-Bit Language Tools Libraries

HammingInit

Description: HammingInit initializes a Hamming window of length *numElems*.

Include: dsp.h

Prototype:

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

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

Return Value: Pointer to base address of initialized window.

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

Source File: inithamm.c

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

DO and REPEAT instruction usage:
None

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

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

HanningInit

Description: HanningInit initializes a Hanning window of length *numElems*.

Include: dsp.h

Prototype:

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

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

Return Value: Pointer to base address of initialized window.

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

Source File: inithann.c

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

DO and REPEAT instruction usage:
None

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

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

KaiserInit

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

Include: dsp.h

Prototype:

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

Arguments:

<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 *must* already exist, with exactly *numElems* number of elements.

Source File: initkais.c

Function Profile: System resources usage:

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

DO and REPEAT instruction usage:
None

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

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

VectorWindow

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

Include: dsp.h

Prototype:

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

Arguments:

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

Source File: dowindow.s

16-Bit Language Tools Libraries

VectorWindow (Continued)

Function Profile: System resources usage:
resources from `VectorMultiply`

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

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

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

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

2.5 MATRIX FUNCTIONS

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

2.5.1 Fractional Matrix Operations

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

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

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

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

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

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

16-Bit Language Tools Libraries

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

- a) The sum of sizes of all the matrices involved in a particular operation falls within the range of available data memory for the target device.
- b) In the case of binary operations the number of rows and columns of the operand matrices *must* obey the rules of vector algebra; i.e., for matrix addition and subtraction the two matrices must have the same number of rows and columns, while for matrix multiplication, the number of columns of the first operand must be the same as the number of rows of the second operand. The source matrix to the inversion operation must be square (the same number of rows as of columns), and non-singular (its determinant different than zero).
- c) The destination matrix *must* be large enough to accept the results of an operation.

2.5.2 User Considerations

- a) No boundary checking is performed by these functions. Out of range dimensions (including zero row and/or zero column matrices) as well as nonconforming use of source matrix sizes in binary operations may produce unexpected results.
- b) The matrix addition and subtraction operations could lead to saturation if the sum of corresponding elements in the source(s) matrix(ces) is greater than $1-2^{-15}$ or smaller than -1.
- c) The matrix multiplication operation could lead to saturation if the sum of products of corresponding row and column sets results in a value greater than $1-2^{-15}$ or smaller than -1.
- d) It is recommended that the STATUS Register (SR) is examined after completion of each function call. In particular, users can inspect the SA, SB and SAB flags after the function returns to determine if saturation occurred.
- e) All the functions have been designed to operate on fractional matrices allocated in default RAM memory space (X-Data or Y-Data).
- f) Operations which return a destination matrix can be nested, so that for instance if:

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

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

2.5.3 Additional Remarks

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

For instance, while computing the `MatrixMultiply` function, the dimensions of the intervening matrices does not necessarily need to be $\{numRows1, numCols1Rows2\}$ for source one matrix, $\{numCols1Rows2, numCols2\}$ for source two matrix, and $\{numRows1, numCols2\}$ for destination matrix. In fact, all that is needed is that their sizes are large enough so that during computation the pointers do 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.

2.5.4 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:

<i>numRows</i>	number of rows in source matrices
<i>numCols</i>	number of columns in source matrices
<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 $srcM1[r][c] + srcM2[r][c]$ is larger than $1-2^{-15}$, this operation results in saturation for the (r, c) -th element. This function can be computed in place. This function can be self applicable.

Source File: `madd.s`

Function Profile: System resources usage:

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

DO and REPEAT instruction usage:

- 1 level DO instructions
- no REPEAT instructions

Program words (24-bit instructions):
14

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

16-Bit Language Tools Libraries

MatrixMultiply

Description:	<p>MatrixMultiply performs the matrix multiplication between the source one and source two matrices, and places the result in the destination matrix. Symbolically:</p> $\text{dstM}[i][j] = \sum_k (\text{srcM1}[i][k])(\text{srcM2}[k][j])$ <p>where: $0 \leq i < \text{numRows1}$ $0 \leq j < \text{numCols2}$ $0 \leq k < \text{numCols1Rows2}$</p>												
Include:	<p>dsp.h</p>												
Prototype:	<pre>extern fractional* MatrixMultiply (int numRows1, int numCols1Rows2, int numCols2, fractional* dstM, fractional* srcM1, fractional* srcM2);</pre>												
Arguments:	<table><tr><td><i>numRows1</i></td><td>number of rows in source one matrix</td></tr><tr><td><i>numCols1Rows2</i></td><td>number of columns in source one matrix; which <i>must</i> be the same as number of rows in source two matrix</td></tr><tr><td><i>numCols2</i></td><td>number of columns in source two matrix</td></tr><tr><td><i>dstM</i></td><td>pointer to destination matrix</td></tr><tr><td><i>srcM1</i></td><td>pointer to source one matrix</td></tr><tr><td><i>srcM2</i></td><td>pointer to source two matrix</td></tr></table>	<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
<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:	<p>If the absolute value of</p> $\sum_k (\text{srcM1}[i][k])(\text{srcM2}[k][j])$ <p>is larger than $1-2^{-15}$, this operation results in saturation for the (i, j)-th element.</p> <p>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.</p>												
Source File:	<p>mmul.s</p>												
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></table> <p>DO and REPEAT instruction usage:</p> <ul style="list-style-type: none">2 level DO instructionsno REPEAT instructions <p>Program words (24-bit instructions):</p> <p>35</p> <p>Cycles (including C-function call and return overheads):</p> $36 + \text{numRows1} * (8 + \text{numCols2} * (7 + 4 * \text{numCols1Rows2}))$	W0..W7	used, not restored	W8..W13	saved, used, restored	ACCA	used, not restored	CORCON	saved, used, restored				
W0..W7	used, not restored												
W8..W13	saved, used, restored												
ACCA	used, not restored												
CORCON	saved, used, restored												

MatrixScale

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

Include: `dsp.h`

Prototype:

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

Arguments:

<code>numRows</code>	number of rows in source matrix
<code>numCols</code>	number of columns in source matrix
<code>dstM</code>	pointer to destination matrix
<code>srcM</code>	pointer to source matrix
<code>sclVal</code>	value by which to scale matrix elements

Return Value: Pointer to base address of destination matrix.

Remarks: This function can be computed in place.

Source File: `mscl.s`

Function Profile: System resources usage:

W0..W5	used, not restored
ACCA	used, not restored
CORCON	saved, used, restored

DO and REPEAT instruction usage:

1 level DO instructions
no REPEAT instructions

Program words (24-bit instructions):

14

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

$20 + 3(\text{numRows} * \text{numCols})$

MatrixSubtract

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

Include: `dsp.h`

Prototype:

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

Arguments:

<code>numRows</code>	number of rows in source matrix(ces)
<code>numCols</code>	number of columns in source matrix(ces)
<code>dstM</code>	pointer to destination matrix
<code>srcM1</code>	pointer to source one matrix (minuend)
<code>srcM2</code>	pointer to source two matrix (subtrahend)

Return Value: Pointer to base address of destination matrix.

16-Bit Language Tools Libraries

MatrixSubtract (Continued)

Remarks: If the absolute value of $srcM1[r][c] - srcM2[r][c]$ is larger than $1-2^{-15}$, this operation results in saturation for the (r, c) -th element. This function can be computed in place. This function can be self applicable.

Source File: `msub.s`

Function Profile: System resources usage:

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

DO and REPEAT instruction usage:

- 1 level DO instructions
- no REPEAT instructions

Program words (24-bit instructions):
15

Cycles (including C-function call and return overheads):
 $20 + 4(numRows * numCols)$

MatrixTranspose

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

Include: `dsp.h`

Prototype:

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

Arguments:

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

Source File: `mtrp.s`

Function Profile: System resources usage:

W0..W5	used, not restored
--------	--------------------

DO and REPEAT instruction usage:

- 2 level DO instructions
- no REPEAT instructions

Program words (24-bit instructions):
14

Cycles (including C-function call and return overheads):
 $16 + numCols * (6 + (numRows-1) * 3)$

2.5.5 Matrix Inversion

The result of inverting a non-singular, square, fractional matrix is another square matrix (of the same dimension) whose element values are not necessarily constrained to the discrete fractional set $\{-1, \dots, 1-2^{-15}\}$. 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:

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

Arguments: `numRowCols` number of rows and columns in (square) source matrix

`dstM` pointer to destination matrix

`srcM` pointer to source matrix

Required for internal use:

`pivotFlag` pointer to a length `numRowsCols` vector

`swappedRows` pointer to a length `numRowsCols` vector

`swappedCols` pointer to a length `numRowsCols` vector

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

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

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

This function can be computed in place.

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

Function Profile: System resources usage:

W0..W7 used, not restored

W8, W14 saved, used, restored

DO and REPEAT instruction usage:

None

Program words (24-bit instructions):

See the file "readme.txt" in `pic30_tools\src\dsp` for this information.

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

See the file "readme.txt" in `pic30_tools\src\dsp` for this information.

2.6 FILTERING FUNCTIONS

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

2.6.1 Fractional Filter Operations

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

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

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

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

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

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

2.6.2 FIR and IIR Filter Implementations

The properties of a filter depend on the value distribution of its coefficients. In particular, two types of filters are of special interest: Finite Impulse Response (FIR) filters, for which $a[m] = 0$ when $1 \leq m < M$, and Infinite Impulse Response (IIR) filters, those such that $a[0] \neq 0$, and $a[m] \neq 0$ for some m in $\{1, \dots, M\}$. 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.

2.6.3 Single Sample Filtering

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

2.6.4 User Considerations

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

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

16-Bit Language Tools Libraries

2.6.5 Individual Functions

In what follows, the individual functions implementing filtering operations are described. For further discussions on digital filters, please consult Alan Oppenheim and Ronald 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:	<pre>typedef struct { int numCoeffs; fractional* coeffsBase; fractional* coeffsEnd; int coeffsPage; fractional* delayBase; fractional* delayEnd; fractional* delay; } FIRStruct;</pre>														
Parameters:	<table><tr><td><i>numCoeffs</i></td><td>number of coefficients in filter (also M)</td></tr><tr><td><i>coeffsBase</i></td><td>base address for filter coefficients (also h)</td></tr><tr><td><i>coeffsEnd</i></td><td>end address for filter coefficients</td></tr><tr><td><i>coeffsPage</i></td><td>coefficients buffer page number</td></tr><tr><td><i>delayBase</i></td><td>base address for delay buffer</td></tr><tr><td><i>delayEnd</i></td><td>end address for delay buffer</td></tr><tr><td><i>delay</i></td><td>current value of delay pointer (also d)</td></tr></table>	<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)
<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:	<p>Number of coefficients in filter is M.</p> <p>Coefficients, $h[m]$, defined in $0 \leq m < M$, either within X-Data or program memory.</p> <p>Delay buffer $d[m]$, defined in $0 \leq m < M$, <i>only</i> in Y-Data.</p> <p>If coefficients are stored in X-Data space, <i>coeffsBase</i> points to the actual address where coefficients are allocated. If coefficients are stored in program memory, <i>coeffsBase</i> 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 <code>psvoffset()</code>.</p> <p><i>coeffsEnd</i> is the address in X-Data space (or offset if in program memory) of the last byte of the filter coefficients buffer.</p> <p>If coefficients are stored in X-Data space, <i>coeffsPage</i> must be set to 0xFF00 (defined value <code>COEFFS_IN_DATA</code>). 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 <code>psvpage()</code>.</p> <p><i>delayBase</i> points to the actual address where the delay buffer is allocated.</p> <p><i>delayEnd</i> is the address of the last byte of the filter delay buffer.</p>														

FIRStruct (Continued)

When the coefficients and delay buffers are implemented as circular increasing modulo buffers, both *coeffsBase* and *delayBase* *must* be aligned to a 'zero' power of two address (*coeffsEnd* and *delayEnd* are odd addresses). Whether these buffers are implemented as circular increasing modulo buffers or not is indicated in the remarks section of each FIR filter function description.

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

FIR

Description:	FIR applies an FIR filter to the sequence of source samples, places the results in the sequence of destination samples, and updates the delay values.								
Include:	dsp.h								
Prototype:	<pre>extern fractional* FIR (int numSamps, fractional* dstSamps, fractional* srcSamps, FIRStruct* filter);</pre>								
Arguments:	<table> <tr> <td><i>numSamps</i></td> <td>number of input samples to filter (also N)</td> </tr> <tr> <td><i>dstSamps</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> </table>	<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
<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:	<p>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.)</p>								
Source File:	fir.s								

16-Bit Language Tools Libraries

FIR (Continued)

Function Profile: System resources usage:

W0..W6	used, not restored
W8, W10	saved, used, restored
ACCA	used, not restored
CORCON	saved, used, restored
MODCON	saved, used, restored
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.

Example

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

FIRDecimate

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

Include: dsp.h

Prototype:

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

Arguments:

<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.

FIRDecimate (Continued)

Remarks: Number of coefficients in filter is M, with M an integer multiple of R. Coefficients, $h[m]$, defined in $0 \leq m < M$, not implemented as a circular modulo buffer.
 Delay, $d[m]$, defined in $0 \leq m < M$, not implemented as a circular modulo buffer.
 Source samples, $x[n]$, defined in $0 \leq n < NR$.
 Destination samples, $y[n]$, defined in $0 \leq n < N$.
 (See also `FIRStruct`, `FIRStructInit`, and `FIRDelayInit`.)

Source File: `firdecim.s`

Function Profile: System resources usage:

W0..W7	used, not restored
W8..W12	saved, used, restored
ACCA	used, not restored
CORCON	saved, used, restored
PSVPAG	saved, used, restored (only if coefficients in P memory)

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

Program words (24-bit instructions):
 48

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

FIRDelayInit

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

Include: `dsp.h`

Prototype:

```
extern void FIRDelayInit (
    FIRStruct* filter
);
```

Arguments: `filter` pointer to `FIRStruct` filter structure.

Remarks: See description of `FIRStruct` structure above.
Note: FIR interpolator's delay is initialized by function `FIRInterpDelayInit`.

Source File: `firdelay.s`

Function Profile: System resources usage:

W0..W2	used, not restored
--------	--------------------

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

Program words (24-bit instructions):
 7

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

16-Bit Language Tools Libraries

FIRInterpolate

Description:	<p>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, $y[n] = x[n/R]$.</p> <p>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.</p>										
Include:	<p>dsp.h</p>										
Prototype:	<pre>extern fractional* FIRInterpolate (int numSamps, fractional* dstSamps, fractional* srcSamps, FIRStruct* filter, int rate);</pre>										
Arguments:	<p><i>numSamps</i> number of input samples (also N, $N = Rp$, p integer) <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>rate</i> rate of interpolation (upsampling factor, also R)</p>										
Return Value:	<p>Pointer to base address of destination samples.</p>										
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 FIRStruct, FIRStructInit, and FIRInterpDelayInit.)</p>										
Source File:	<p>firinter.s</p>										
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> <ul style="list-style-type: none">2 level DO instructions1 level REPEAT instructions <p>Program words (24-bit instructions):</p> <p>63</p> <p>Cycles (including C-function call and return overheads):</p> <ul style="list-style-type: none">$45 + 6(M / R) + N(14 + M / R + 3M + 5R)$, or$48 + 6(M / R) + N(14 + M / R + 4M + 5R)$ if coefficients in P memory.	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: FIRInterpDelayInit initializes to zero the delay values in an FIRStruct filter structure, optimized for use with an FIR interpolating filter.

Include: dsp.h

Prototype:

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

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

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

Source File: firintdl.s

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

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

Program words (24-bit instructions):
 13

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

FIRLattice

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

Include: dsp.h

Prototype:

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

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

Return Value: Pointer to base address of destination samples.

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

Source File: firlatt.s

16-Bit Language Tools Libraries

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:

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

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: Number of coefficients in filter is M.
Coefficients, $h[m]$, defined in $0 \leq m < M$, implemented as a circular increasing modulo buffer.
delay, $d[m]$, defined in $0 \leq m < M-1$, implemented as a circular increasing modulo buffer.
Source samples, $x[n]$, defined in $0 \leq n < N$.
Reference samples, $r[n]$, defined in $0 \leq n < N$.
Destination samples, $y[n]$, defined in $0 \leq n < N$.
Adaptation:

$$h_m[n] = h_m[n - 1] + \mu * (r[n] - y[n]) * x[n - m],$$
for $0 \leq n < N, 0 \leq m < M$.
The operation could result in saturation if the absolute value of $(r[n] - y[n])$ is greater than or equal to one.
Filter coefficients *must not* be allocated in program memory, because in that case their values could not be adapted. If filter coefficients are detected as allocated in program memory the function returns NULL.
(See also FIRStruct, FIRStructInit and FIRDelayInit.)

Source File: `firlms.s`

Function Profile: System resources usage:

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

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

Program words (24-bit instructions):
76

Cycles (including C-function call and return overheads):
 $61 + N(13 + 5M)$

FIRLMSNorm

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

Include: `dsp.h`

FIRLMSNorm (Continued)

Prototype:

```
extern fractional* FIRLMSNorm (  
    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:

$$h_m[n] = h_m[n - 1] + nu[n] * (r[n] - y[n]) * x[n - m],$$

for $0 \leq n < N$, $0 \leq m < M$,

$$\text{where } nu[n] = \mu / (\mu + E[n])$$

with $E[n] = E[n - 1] + (x[n])^2 - (x[n - M + 1])^2$ an estimate of input signal energy.

On start up, *energyEstimate* should be initialized to the value of $E[-1]$ (zero the first time the filter is invoked). Upon return, *energyEstimate* is updated to the value $E[N - 1]$ (which may be used as the start up value for a subsequent function call if filtering an extension of the input signal).

The operation could result in saturation if the absolute value of $(r[n] - y[n])$ is greater than or equal to one.

Note: Another expression for the energy estimate is:

$$E[n] = (x[n])^2 + (x[n - 1])^2 + \dots + (x[n - M + 2])^2.$$

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

$$-M + 2$$

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

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.s`

FIRLMSNorm (Continued)

Function Profile: System resources usage:

W0..W7	used, not restored
W8..W13	saved, used, restored
ACCA	used, not restored
ACCB	used, not restored
CORCON	saved, used, restored
MODCON	saved, used, restored
XMODSTRT	saved, used, restored
XMODEND	saved, used, restored
YMODSTRT	saved, used, restored

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

Program words (24-bit instructions):
 91

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

FIRStructInit

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

Include: dsp.h

Prototype:

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

Arguments:

<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 delayBase.

Source File: firinit.s

Function Profile: System resources usage:

W0..W5	used, not restored
--------	--------------------

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

Program words (24-bit instructions):
 10

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

16-Bit Language Tools Libraries

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.s

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 IIRCanonicStruct filter structure.

Include: dsp.h

Prototype:

```
extern void IIRCanonicInit (
    IIRCanonicStruct* filter
);
```

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

Initialization Description:
filter pointer to IIRCanonicStruct filter structure

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

Source File: iircan.s

Function Profile: System resources usage:

W0, W1	used, not restored
--------	--------------------

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

Program words (24-bit instructions):
 7

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

16-Bit Language Tools Libraries

IIRLattice

Description:	IIRLattice uses a lattice structure implementation to apply an IIR filter to the sequence of source samples. It then places the results in the sequence of destination samples, and updates the delay values.
Include:	dsp.h
Prototype:	<pre>typedef struct { int order; fractional* kappaVals; fractional* gammaVals; int coeffsPage; fractional* delay; } IIRLatticeStruct; extern fractional* IIRLattice (int numSamps, fractional* dstSamps, fractional* srcSamps, IIRLatticeStruct* filter);</pre>
Arguments:	<p>Filter structure:</p> <p><i>order</i> filter order (also M, $M \leq N$; see FIRLattice for N)</p> <p><i>kappaVals</i> base address for lattice coefficients (also k), either in X-Data or program memory</p> <p><i>gammaVals</i> base address for ladder coefficients (also g), either in X-Data or program memory. If NULL, the function will implement an all-pole filter.</p> <p><i>coeffsPage</i> coefficients buffer page number, or 0xFF00 (defined value COEFFS_IN_DATA) if coefficients in data space</p> <p><i>delay</i> base address for delay (also d), <i>only</i> in Y-Data</p> <p>Filter Description:</p> <p><i>numSamps</i> number of input samples to filter (also N, $N \geq M$; see IIRLatticeStruct for M)</p> <p><i>dstSamps</i> pointer to destination samples (also y)</p> <p><i>srcSamps</i> pointer to source samples (also x)</p> <p><i>filter</i> pointer to IIRLatticeStruct filter structure</p>
Return Value:	Pointer to base address of destination samples.
Remarks:	<p>Lattice coefficients, $k[m]$, defined in $0 \leq m \leq M$.</p> <p>Ladder coefficients, $g[m]$, defined in $0 \leq m \leq M$ (unless if implementing an all-pole filter).</p> <p>Delay, $d[m]$, defined in $0 \leq m \leq M$.</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>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.</p>
Source File:	iirlatt.s

IIRLattice (Continued)

Function Profile: System resources usage:

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

DO and REPEAT instruction usage:

- 2 level DO instructions
- no REPEAT instructions

Program words (24-bit instructions):

76

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

46 + N(16 + 7M), or
49 + N(20 + 8M) if coefficients in program memory.

If implementing an all-pole filter:

46 + N(16 + 6M), or
49 + N(16 + 7M) if coefficients in program memory

IIRLatticeInit

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

Include: dsp.h

Prototype:

```
extern void IIRLatticeInit (
    IIRLatticeStruct* filter
);
```

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

Initialization Description:
filter pointer to IIRLatticeStruct filter structure.

Source File: iirlattd.s

Function Profile: System resources usage:

W0..W2	used, not restored
--------	--------------------

DO and REPEAT instruction usage:

- no DO instructions
- 1 level REPEAT instructions

Program words (24-bit instructions):

6

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

10 + M

16-Bit Language Tools Libraries

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 in Y-Data</i>
<i>delayBase2</i>	pointer to filter state 2, with one word of delay per second order section (also d2), <i>only in Y-Data</i>
<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.

Remarks: There are 5 coefficients per second order (biquadratic) section arranged in the ordered set {b0[s], b1[s], a1[s], b2[s], a2[s]}, $0 \leq s < S$. 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 independent buffers, each buffer containing one word of filter state per section {d2[s], d1[s]}, $0 \leq s < S$. Source samples, x[n], defined in $0 \leq n < N$. Destination samples, y[n], defined in $0 \leq n < N$. The output scale is applied as a shift to the output of the filter structure prior to storing the result in the output sequence. It is used to restore the filter gain to 0 dB. Shift count may be zero; if not zero, it represents the number of bits to shift: negative indicates shift left, positive is shift right.

Source File: iirtrans.s

IIRTransposed (Continued)

Function Profile:	<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 1 level REPEAT instructions</p> <p>Program words (24-bit instructions): 48</p> <p>Cycles (including C-function call and return overheads): 35 + N(11 + 11S), or 38 + N(9 + 17S) if coefficients in P memory. S is number of second order sections.</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												
Example	Please refer to the MPLAB C30 installation folder for a sample project demonstrating the use of this function.												

IIRTransposedInit

Description:	IIRTransposedInit initializes to zero the delay values in an IIRTransposedStruct filter structure.		
Include:	dsp.h		
Prototype:	extern void IIRTransposedInit (IIRTransposedStruct* filter);		
Arguments:	Filter structure: (See description of IIRTransposed function).		
Remarks:	<p>Initialization Description: <i>filter</i> pointer to IIRTransposedStruct filter structure.</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 ≤ s < S.</p>		
Source File:	irtrans.s		
Function Profile:	<p>System resources usage:</p> <table> <tr><td>W0..W2</td><td>used, not restored</td></tr> </table> <p>DO and REPEAT instruction usage:</p> <p>1 level DO instructions no REPEAT instructions</p> <p>Program words (24-bit instructions): 8</p> <p>Cycles (including C-function call and return overheads): 11 + 2S, S is number of second order sections.</p>	W0..W2	used, not restored
W0..W2	used, not restored		
Example	Please refer to the MPLAB C30 installation folder for a sample project demonstrating the use of this function.		

16-Bit Language Tools Libraries

2.6.6 OCTAVE model for analysis of IIRLattice filter

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

IIRLattice OCTAVE model

```
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
```

16-Bit Language Tools Libraries

2.7 TRANSFORM FUNCTIONS

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

2.7.1 Fractional Transform Operations

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

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

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

2.7.2 Fractional Complex Vectors

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

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

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

2.7.3 User Considerations

- a) No boundary checking is performed by these functions. Out of range sizes (including zero length vectors) as well as nonconforming use of source complex vectors and factor sets may produce unexpected results.
- b) It is recommended that the STATUS Register (SR) is examined after completion of each function call. In particular, users can inspect the SA, SB and SAB flags after the function returns to determine if saturation occurred.
- c) The input and output complex vectors involved in the family of transformations *must* be allocated in Y-Data memory. Transforms factors may be allocated either in X-Data or program memory.
- d) Because bit reverse addressing requires the vector set to be modulo aligned, the input and output complex vectors in operations using either explicitly or implicitly the `BitReverseComplex` function must be properly allocated.
- e) Operations which return a destination complex vector can be nested, so that for instance if:
 - a = Op1 (b, c), with b = Op2 (d), and c = Op3 (e, f), then
 - a = Op1 (Op2 (d), Op3 (e, f)).

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

BitReverseComplex

Description:	<code>BitReverseComplex</code> reorganizes the elements of a complex vector in bit reverse order.						
Include:	<code>dsp.h</code>						
Prototype:	<pre>extern fractcomplex* BitReverseComplex (int log2N, fractcomplex* srcCV);</pre>						
Arguments:	<p><code>log2N</code> based 2 logarithm of N (number of complex elements in source vector)</p> <p><code>srcCV</code> pointer to source complex vector</p>						
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 <code>srcCV</code> vector must be allocated at a modulo alignment of N.</p> <p>This function operates in place.</p>						
Source File:	<code>bitrev.s</code>						
Function Profile:	<p>System resources usage:</p> <table> <tr> <td>W0..W7</td> <td>used, not restored</td> </tr> <tr> <td>MODCON</td> <td>saved, used, restored</td> </tr> <tr> <td>XBREV</td> <td>saved, used, restored</td> </tr> </table> <p>DO and REPEAT instruction usage:</p> <ul style="list-style-type: none"> 1 level DO instructions no REPEAT instructions <p>Program words (24-bit instructions):</p> <p>27</p> <p>Cycles (including C-function call and return overheads):</p> <p>See below:</p>	W0..W7	used, not restored	MODCON	saved, used, restored	XBREV	saved, used, restored
W0..W7	used, not restored						
MODCON	saved, used, restored						
XBREV	saved, used, restored						

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

16-Bit Language Tools Libraries

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`

Prototype:

```
extern fractcomplex* CosFactorInit (
    int log2N,
    fractcomplex* cosFactors
);
```

Arguments:

<code>log2N</code>	based 2 logarithm of N (number of complex factors needed by a DCT)
<code>cosFactors</code>	pointer to complex cosine factors

Return Value: Pointer to base address of cosine factors.

Remarks: N *must* be an integer power of 2. Only the first N/2 cosine factors are generated. A complex vector of size N/2 *must* have already been allocated and assigned to `cosFactors` prior to invoking the function. The complex vector *should* reside in X-Data memory. Factors are computed in floating-point arithmetic and converted to 1.15 complex fractionals.

Source File: `initcosf.c`

Function Profile: System resources usage:

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

DO and REPEAT instruction usage:
None

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

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

DCT

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

Include: `dsp.h`

Prototype:

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

16-Bit Language Tools Libraries

DCT (Continued)

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>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 operates out of place. A vector of size 2N elements, <i>must</i> already have been allocated and assigned to <i>dstV</i>.</p> <p>The <i>dstV</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 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, <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 <code>psvoffset()</code>.</p> <p>If the transform factors are stored in X-Data space, <i>factPage</i> must be set to 0xFF00 (defined value <code>COEFFS_IN_DATA</code>). 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 <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.s</code>	
Function Profile:		<p>System resources usage:</p> <ul style="list-style-type: none">W0..W5 used, not restoredplus system resources from <code>VectorZeroPad</code>, and <code>DCTIP</code>. <p>DO and REPEAT instruction usage:</p> <ul style="list-style-type: none">no DO instructionsno REPEAT instructionsplus DO/REPEAT instructions from <code>VectorZeroPad</code>, and <code>DCTIP</code>. <p>Program words (24-bit instructions):</p> <ul style="list-style-type: none">16plus program words from <code>VectorZeroPad</code>, and <code>DCTIP</code>. <p>Cycles (including C-function call and return overheads):</p> <ul style="list-style-type: none">22plus cycles from <code>VectorZeroPad</code>, and <code>DCTIP</code>. <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:	dsp.h										
Prototype:	<pre>extern fractional* DCTIP (int log2N, fractional* srcV, fractcomplex* cosFactors, fractcomplex* twiddleFactors, 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>twiddleFactors</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>twiddleFactors</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>twiddleFactors</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>twiddleFactors</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>twiddleFactors</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 <code>psvoffset()</code>.</p> <p>If the transform factors are stored in X-Data space, <i>factPage</i> must be set to 0xFF00 (defined value <code>COEFFS_IN_DATA</code>). 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 <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>Output is scaled by the factor $1/(\sqrt{2N})$.</p>										
Source File:	dctoop.s										

16-Bit Language Tools Libraries

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: `fftoop.s`

Function Profile: System resources usage:
 W0..W4 used, not restored
 plus system resources from `VectorCopy`, `FFTComplexIP`, and `BitReverseComplex`.

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

Program words (24-bit instructions):
 17
 plus program words from `VectorCopy`, `FFTComplexIP`, and `BitReverseComplex`.

Cycles (including C-function call and return overheads):
 23
 plus cycles from `VectorCopy`, `FFTComplexIP`, and `BitReverseComplex`.

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

16-Bit Language Tools Libraries

FFTComplexIP

Description:	FFTComplexIP computes the Discrete Fourier Transform of a source complex vector in place..								
Include:	dsp.h								
Prototype:	<pre>extern fractcomplex* FFTComplexIP (int log2N, fractcomplex* srcCV, fractcomplex* twiddleFactors, 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 complex vector</td></tr><tr><td><i>twiddleFactors</i></td><td>base address of 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 complex vector	<i>twiddleFactors</i>	base address of 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 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.								
Remarks:	<p><i>N</i> must 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, <i>twiddleFactors</i> points to the actual address where the factors are allocated. If the twiddle factors are stored in program memory, <i>twiddleFactors</i> 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, <i>factPage</i> must be set to 0xFF00 (defined value <code>COEFFS_IN_DATA</code>). 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 <code>psvpage()</code>.</p> <p>The twiddle factors <i>must</i> be initialized with <code>conjFlag</code> set to zero. Output is scaled by the factor 1/N.</p>								
Source File:	fft.s								

FFTComplexIP (Continued)

Function Profile: System resources usage:

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

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

Program words (24-bit instructions):
 59

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

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

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.

IFFTComplex (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]`.
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()`.
If the twiddle factors are stored in X-Data space, `factPage` must be set to `0xFF00` (defined value `COEFFS_IN_DATA`). If they are stored in program memory, `factPage` is the program page number containing the factors. This latter value can be calculated using the inline assembly operator `psvpage()`.
The twiddle factors *must* be initialized with `conjFlag` set to a value other than zero.
Only the first `N/2` twiddle factors are needed.

Source File: `ifftoop.s`

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

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

IFFTComplexIP

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

Include: dsp.h

Prototype:

```
extern fractcomplex* IFFTComplexIP (
    int log2N,
    fractcomplex* srcCV,
    fractcomplex* 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.

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

Source File: `ifft.s`

16-Bit Language Tools Libraries

IFFTComplexIP (Continued)

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

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

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

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

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

SquareMagnitudeCplx

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

Include: dsp.h

Prototype:

```
extern fractional* SquareMagnitudeCplx (  
    int numelems,  
    fractcomplex* srcV,  
    fractional* dstV  
);
```

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

Return Value: Pointer to base address of destination vector.

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

Source File: cplxsqrmag.s

SquareMagnitudeCplx (Continued)

Function Profile: System resources usage:

W0..W2	used, not restored
W4, W5, W10	saved, used, restored
ACCA	used, not restored
CORCON	saved, used, restored

DO and REPEAT instruction usage:

- 1 level DO instructions
- no REPEAT instructions

Program words (24-bit instructions):
19

Cycles (including C-function call and return overheads):
 $20 + 3(\text{numElements})$

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

TwidFactorInit

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

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

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

Include: `dsp.h`

Prototype:

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

Arguments:

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

16-Bit Language Tools Libraries

TwidFactorInit (Continued)

Function Profile: System resources usage:

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

DO and REPEAT instruction usage:
None

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

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

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

2.8 CONTROL FUNCTIONS

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

2.8.1 Proportional Integral Derivative (PID) Control

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

2.8.1.1 PID CONTROLLER BACKGROUND

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

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

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

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

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

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

It should be noted that not all PID controllers will implement the D or, less commonly, the I terms. For example, the speed controller in a Brushless DC motor application described by Microchip Application Note AN901 does not have a D term due to the rel-

16-Bit Language Tools Libraries

atively slow response time of motor speed changes. In this case, the D term could cause excessive changes in PWM duty cycle that could affect the operation of the sensorless algorithm and produce over current trips.

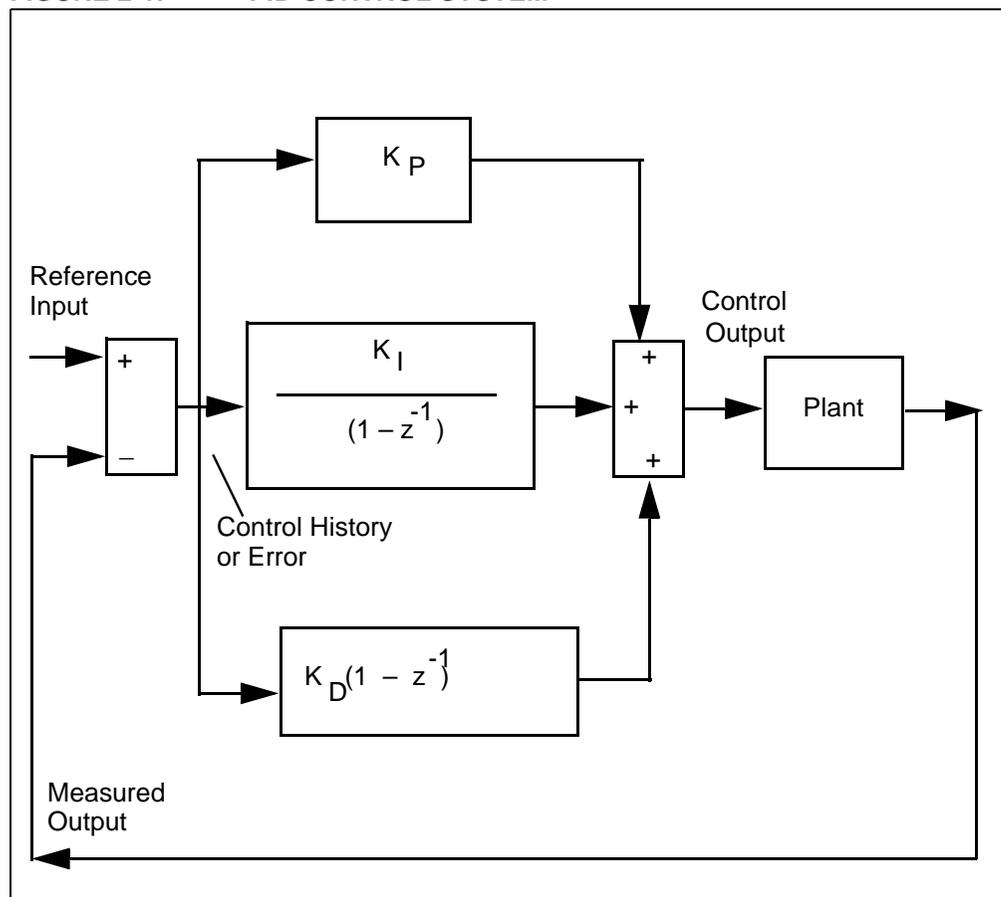
2.8.1.2 ADJUSTING PID GAINS

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

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

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

FIGURE 2-1: PID CONTROL SYSTEM



2.8.1.3 PID LIBRARY FUNCTIONS AND DATA STRUCTURES

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

```
typedef struct {
fractional* abcCoefficients;
fractional* controlHistory;
fractional controlOutput;
fractional measuredOutput;
fractional controlReference;
} tPID;
```

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

1. Calculate Coefficients from PID Gain values

The element `abcCoefficients` in the data structure of type `tPID` is a pointer to A, B & C coefficients located in X-data space. These coefficients are derived from the PID gain values, K_p , K_i and K_d , shown in Figure 2-1, as follows:

$$A = K_p + K_i + K_d$$

$$B = -(K_p + 2 * K_d)$$

$$C = K_d$$

To derive the A, B and C coefficients, the DSP library provides a function, `PIDCoeffCalc`.

2. Clear the PID State Variables

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

2.8.2 Individual Functions

PIDInit

Description:	This routine clears the delay line elements in the 3-element array located in Y-space and pointed to by <code>controlHistory</code> . It also clears the current PID output element, <code>controlOutput</code> .
Include:	<code>dsp.h</code>
Prototype:	<code>void PIDInit (tPID *fooPIDStruct);</code>
Arguments:	<code>fooPIDStruct</code> is a pointer to a PID data structure of type <code>tPID</code>
Return Value:	void.
Remarks:	
Source File:	<code>pid.s</code>

16-Bit Language Tools Libraries

PIDInit (Continued)

Function Profile: System resources usage:

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

DO and REPEAT instruction usage:

0 level DO instructions
0 REPEAT instructions

Program words (24-bit instructions):
11

Cycles (including C-function call and return overheads):
13

PIDCoeffCalc

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

```
abcCoefficients[0] = Kp + Ki + Kd  
abcCoefficients[1] = -(Kp + 2*Kd)  
abcCoefficients[2] = Kd
```

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

Include: dsp.h

Prototype: void PIDCoeffCalc (fractional *fooPIDGainCoeff,
tPID *fooPIDStruct)

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

Return Value: Void.

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

Source File: pid.s

Function Profile: System resources usage:

W0..W2	used, not restored
ACCA, ACCB	used, not restored
CORCON	saved, used, restored

DO and REPEAT instruction usage:

0 level DO instructions
0 REPEAT instructions

Program words (24-bit instructions):
18

Cycles (including C-function call and return overheads):
20

PID

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

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

where,

$$\begin{aligned} \text{abcCoefficient}[0] &= K_p + K_i + K_d \\ \text{abcCoefficient}[1] &= -(K_p + 2 * K_d) \\ \text{abcCoefficient}[2] &= K_d \\ \text{ControlHistory}[n] &= \text{MeasuredOutput}[n] - \text{ReferenceInput}[n] \end{aligned}$$

Include: `dsp.h`

Prototype: `extern void PID (tPID* ooPIDStruct);`

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

Return Value: Pointer to `fooPIDStruct`

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

Source File: `pid.s`

Function Profile: System resources usage:

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

DO and REPEAT instruction usage:

0 level DO instructions
0 REPEAT instructions

Program words (24-bit instructions):
28

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

16-Bit Language Tools Libraries

2.9 MISCELLANEOUS FUNCTIONS

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

2.9.1 Individual Functions

Fract2Float

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

Include: dsp.h

Prototype:
`extern float Fract2Float (
 fractional aVal
);`

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

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

Remarks: None

Source File: flt2frct.c

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

DO and REPEAT instruction usage:
None

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

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

Float2Fract

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

Include: dsp.h

Prototype:
`extern fractional Float2Fract (
 float aVal
);`

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

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

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

Source File: flt2frct.c

Float2Fract (Continued)

Function Profile:

System resources usage:

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

DO and REPEAT instruction usage:

None

Program words (24-bit instructions):

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

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

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

16-Bit Language Tools Libraries

NOTES:

Chapter 3. 16-Bit Peripheral Libraries

3.1 INTRODUCTION

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

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

3.1.1 .Assembly Code Applications

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

3.1.2 C Code Applications

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

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

3.1.3 Chapter Organization

This chapter is organized as follows:

- Using the 16-Bit Peripheral Libraries

Software Functions

- External LCD Functions

Hardware Functions

- CAN Functions
- ADC12 Functions
- ADC10 Functions
- Timer Functions
- Reset/Control Functions
- I/O Port Functions
- Input Capture Functions
- Output Compare Functions
- UART Functions
- DCI Functions
- SPI Functions
- QEI Functions
- PWM Functions
- I2C™ Functions

16-Bit Language Tools Libraries

3.2 USING THE 16-BIT PERIPHERAL LIBRARIES

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

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

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

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

When compiling an application, header file must be referenced (using `#include`) by all source files which call a function in the library or use its symbols or `typedefs`.

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

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

```
makeplib.bat 30f6014
```

or

```
makeplib.bat 30F6014
```

will rebuild the library for the dsPIC30F6014 device.

3.3 EXTERNAL LCD FUNCTIONS

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

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

- dsPIC30F5011
- dsPIC30F5013
- dsPIC30F6010
- dsPIC30F6011
- dsPIC30F6012
- dsPIC30F6013
- dsPIC30F6014

3.3.1 Individual Functions

BusyXLCD

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:	<code>xlcd.h</code>
Prototype:	<code>void OpenXLCD (unsigned char lcdtype);</code>
Arguments:	<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
Return Value:	None
Remarks:	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:

OpenXLCD (Continued)

Control I/O pin definitions

```
RW_PIN      PORTxbits.Rx?
TRIS_RW     TRISxbits.Rx?
RS_PIN      PORTxbits.Rx?
TRIS_RS     TRISxbits.Rx?
E_PIN       PORTxbits.Rx?
TRIS_E      TRISxbits.Rx?
```

where x is the PORT, ? is the pin number

Data pin definitions

```
DATA_PIN_?   PORTxbits.RD?
TRIS_DATA_PIN_? TRISxbits.TRISD?
```

where x is the PORT, ? is the pin number

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

The control pins can be on any port and are not required to be on the same port. The data interface must be defined as either 4-bit or 8-bit.

The 8-bit interface is defined when a

```
#define EIGHT_BIT_INTERFACE is included in the header file
xlcd.h. If no define is included, then the 4-bit interface is included.
```

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

This function also requires three external routines for specific delays:

```
DelayFor18TCY()      18 Tcy delay
DelayPORXLCD()       15ms delay
DelayXLCD()          5ms delay
Delay100XLCD()       100Tcy delay
```

Source File: openXLCD.c

Code Example:

```
OpenXLCD(EIGHT_BIT & TWO_LINE
          & SEG1_50_SEG51_100 & COM1_COM16);
```

putsXLCD putrsXLCD

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

Include: xlcd.h

Prototype:

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

Arguments: *buffer* Pointer to the characters to be written to the LCD controller.

Return Value: None

Remarks: These functions write a string of characters located in *buffer* to the P-tec PCOG1602B LCD controller until a NULL character is encountered in the string.
For continuous display of data written to the P-tec PCOG1602B LCD controller, you could set up the display in a Shift mode.

Source File: PutsXLCD.c
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 `Set??RamAddr()` 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);`

Arguments: None

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

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

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

Source File: `ReadDataXLCD.c`

Code Example:

```
char data;
while (BusyXLCD());
data = ReadDataXLCD();
```

16-Bit Language Tools Libraries

SetCGRamAddr

Description: This function sets the character generator address.

Include: `xlcd.h`

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

Arguments: `CGaddr` Character generator address.

Return Value: None

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

Source File: `SetCGRamAddr.c`

Code Example:

```
char cgaddr = 0x1F;
while (BusyXLCD());
SetCGRamAddr(cgaddr);
```

SetDDRamAddr

Description: This function sets the display data address.

Include: `xlcd.h`

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

Arguments: `DDaddr` Display data address.

Return Value: None

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

Source File: `SetDDRamAddr.c`

Code Example:

```
char ddaddr = 0x10;
while (BusyXLCD());
SetDDRamAddr(ddaddr);
```

WriteDataXLCD

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

Include: `xlcd.h`

Prototype: `void WriteDataXLCD (char data);`

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

Return Value: None

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

Source File: `WriteDataXLCD.c`

Code Example: `WriteDataXLCD(0x30);`

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:	<i>cmd</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> SINLE_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 <u>Display On/Off control</u> DON DOFF CURSOR_ON CURSOR_OFF BLINK_ON BLINK_OFF <u>Cursor or Display Shift defines</u> SHIFT_CUR_LEFT SHIFT_CUR_RIGHT SHIFT_DISP_LEFT SHIFT_DISP_RIGHT
Return Value:	None
Remarks:	This function writes the command byte to the P-tec PCOG1602B LCD controller. The user must first check to see if the LCD controller is busy by calling the <code>BusyXLCD()</code> function.
Source File:	<code>WriteCmdXLCD.c</code>
Code Example:	<pre>while(BusyXLCD()); WriteCmdXLCD(EIGHT_BIT & TWO_LINE); WriteCmdXLCD(DON); WriteCmdXLCD(SHIFT_DISP_LEFT);</pre>

3.3.2 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 dispaly data in the second line */
  SetDDRamAddr(0x40);
  while(BusyXLCD());
  buffer = mesg2;
  PutsXLCD(buffer);
  while(BusyXLCD());
  return 0;
}
```

3.4 CAN FUNCTIONS

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

3.4.1 Individual Functions

CAN1AbortAll **CAN2AbortAll**

Description:	This function initiates abort of all the pending transmissions.
Include:	can.h
Prototype:	void CAN1AbortAll(void); void CAN2AbortAll(void);
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();

16-Bit Language Tools Libraries

CAN1GetTXErrorCount CAN2GetTXErrorCount

Description:	This function returns the transmit error count value.
Include:	can.h
Prototype:	<pre>unsigned char CAN1GetTXErrorCount(void); unsigned char CAN2GetTXErrorCount(void);</pre>
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:	<pre>unsigned char tx_error_count; tx_error_count = CAN1GetTXErrorCount();</pre>

CAN1IsBusOff CAN2IsBusOff

Description:	This function determines whether the CAN node is in BusOff mode.
Include:	can.h
Prototype:	<pre>char CAN1IsBusOff(void); char CAN2IsBusOff(void);</pre>
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:	<pre>while(CAN1IsBusOff());</pre>

CAN1IsRXReady CAN2IsRXReady

Description:	This function returns the receive buffer full status.
Include:	can.h
Prototype:	<pre>char CAN1IsRXReady(char); char CAN2IsRXReady(char);</pre>
Arguments:	<i>buffno</i> The value of buffno indicates the receive buffer whose status is required.
Return Value:	If RXFUL is 1, it indicates that the receive buffer contains a received message. If RXFUL is 0, it indicates that the receive buffer is open to receive a new message.
Remarks:	This function returns the status of the RXFUL bit of Receive Control register.
Source File:	CAN1IsRXReady.c CAN2IsRXReady.c
Code Example:	<pre>char rx_1_status; rx_1_status = CAN1IsRXReady(1);</pre>

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`

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();`

16-Bit Language Tools Libraries

CAN1IsTXReady CAN2IsTXReady

Description:	This function returns the transmitter status indicating if the CAN node is ready for next transmission.
Include:	<code>can.h</code>
Prototype:	<code>char CAN1IsTXReady(char);</code> <code>char CAN2IsTXReady(char);</code>
Arguments:	<i>buffno</i> The value of <i>buffno</i> 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:	<code>CAN1IsTXReady.c</code> <code>CAN2IsTXReady.c</code>
Code Example:	<pre>char tx_2_status; tx_2_status = CAN1IsTXReady(2);</pre>

CAN1ReceiveMessage CAN2ReceiveMessage

Description:	This function read the data from the receive buffer.
Include:	<code>can.h</code>
Prototype:	<code>void CAN1ReceiveMessage(unsigned char * data, unsigned char datalen, char MsgFlag);</code> <code>void CAN2ReceiveMessage(unsigned char * data, unsigned char datalen, char MsgFlag);</code>
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:	<code>CAN1ReceiveMessage.c</code> <code>CAN2ReceiveMessage.c</code>
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:	<table><tr><td><i>sid</i></td><td>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</td></tr><tr><td><i>eid</i></td><td>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</td></tr><tr><td><i>data</i></td><td>The pointer to the location where data to be transmitted is stored.</td></tr><tr><td><i>datalen</i></td><td>The number of bytes of data to be transmitted.</td></tr><tr><td><i>MsgFlag</i></td><td>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.</td></tr></table>	<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.	<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.
<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.										
<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>										

16-Bit Language Tools Libraries

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><tr><td><i>filter_no</i></td><td>The filter (0, 1, 2, 3, 4 or 5) for which new filter 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 filter 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 filter 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:	<table><tr><td><i>mask_no</i></td><td>The mask ('0' or '1') for which mask values have to be configured.</td></tr><tr><td><i>sid</i></td><td>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</td></tr><tr><td><i>eid</i></td><td>The 32-bit value to be written into CiRXMnEIDH and CiRXMnEIDL registers. CAN_MASK_EID(x) x is the required EID value.</td></tr></table>	<i>mask_no</i>	The mask ('0' or '1') for which mask values have to be configured.	<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	<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.
<i>mask_no</i>	The mask ('0' or '1') for which mask values have to be configured.						
<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						
<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.						
Return Value:	None						

CAN1SetMask (Continued) CAN2SetMask

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

Source File: CAN1SetMask.c
CAN2SetMask.c

Code Example:

```
CAN1SetMask(1, CAN_MASK_SID(7) &  
CAN_MATCH_FILTER_TYPE, CAN_MASK_EID(3));
```

CAN1SetOperationMode CAN2SetOperationMode

Description: This function configures the CAN module

Include: can.h

Prototype:

```
void CAN1SetOperationMode(unsigned int config);  
void CAN2SetOperationMode(unsigned int config);
```

Arguments: *config* The 16-bit value to be loaded into CiCTRL register, the combination of the following defines.

- CAN_IDLE_CON CAN On in Idle mode
- CAN_IDLE_STOP CAN Stop in Idle mode
- CAN_MASTERCLOCK_1 FCAN is FCY
- CAN_MASTERCLOCK_0 FCAN is 4 FCY

CAN modes of operation

- CAN_REQ_OPERMODE_NOR
- CAN_REQ_OPERMODE_DIS
- CAN_REQ_OPERMODE_LOOPBK
- CAN_REQ_OPERMODE_LISTENONLY
- CAN_REQ_OPERMODE_CONFIG
- CAN_REQ_OPERMODE_LISTENALL

CAN Capture Enable/Disable

- CAN_CAPTURE_EN
- CAN_CAPTURE_DIS

Return Value: None

Remarks: This function configures the following bits of CiCTRL:-CSIDL, REQOP<2:0> and CANCKS

Source File: CAN1SetOperationMode.c
CAN2SetOperationMode.c

Code Example:

```
CAN1SetOperationMode(CAN_IDLE_STOP &  
CAN_MASTERCLOCK_0 & CAN_REQ_OPERMODE_DIS &  
CAN_CAPTURE_DIS);
```

16-Bit Language Tools Libraries

CAN1SetOperationModeNoWait CAN2SetOperationModeNoWait

Description:	This function aborts the pending transmissions and configures the CAN module
Include:	can.h
Prototype:	<pre>void CAN1SetOperationModeNoWait(unsigned int <i>config</i>); void CAN2SetOperationModeNoWait(unsigned int <i>config</i>);</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_NO_WAIT CAN On in Idle mode CAN_IDLE_STOP_NO_WAIT CAN Stop in Idle mode CAN_MASTERCLOCK_1_NO_WAIT FCAN is FCY CAN_MASTERCLOCK_0_NO_WAIT FCAN is 4 FCY</p> <p><u>CAN modes of operation</u> 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</p> <p><u>CAN Capture Enable/Disable</u> CAN_CAPTURE_EN_NO_WAIT CAN_CAPTURE_DIS_NO_WAIT</p>
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:	<pre>CAN1SetOperationModeNoWait(CAN_IDLE_CON & CAN_MASTERCLOCK_1 & CAN_REQ_OPERMODE_LISTEN & CAN_CAPTURE_DIS_NO_WAIT);</pre>

CAN1SetRXMode CAN2SetRXMode

Description:	This function configures the CAN receiver.
Include:	can.h
Prototype:	<pre>void CAN1SetRXMode(char <i>buffno</i>, unsigned int <i>config</i>); void CAN2SetRXMode(char <i>buffno</i>, unsigned int <i>config</i>);</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>

CAN1SetRXMode (Continued) CAN2SetRXMode

Return Value: None

Remarks: This function configures the following bits of CiRXnCON register: RXRTR, RXFUL (only 0), RXM<1:0> and DBEN

Source File: CAN1SetRXMode.c
CAN2SetRXMode.c

Code Example:

```
CAN1SetRXMode(0, CAN_RXFUL_CLEAR &
CAN_BUF0_DBLBUFFER_EN);
```

CAN1SetTXMode (function) CAN2SetTXMode

Description: This function configures the CAN transmitter module

Include: can.h

Prototype:

```
void CAN1SetTXMode(char buffno, unsigned int
config);
void CAN2SetTXMode(char buffno, unsigned int
config);
```

Arguments: *buffno* buffno indicates the control reg to be configured.
config The value to be written into CiTXnCON reg, the combination of the following defines.

Message send request
CAN_TX_REQ
CAN_TX_STOP_REQ

Message transmission priority
CAN_TX_PRIORITY_HIGH
CAN_TX_PRIORITY_HIGH_INTER
CAN_TX_PRIORITY_LOW_INTER
CAN_TX_PRIORITY_LOW

Return Value: None

Remarks: This function configures the following bits of CiTXnCON register: TXRTR, TXREQ, DLC, TXPRI<1:0>

Source File: CAN1SetTXMode.c
CAN2SetTXMode.c

Code Example:

```
CAN1SetTXMode(1, CAN_TX_STOP_REQ &
CAN_TX_PRIORITY_HIGH);
```

16-Bit Language Tools Libraries

CAN1Initialize CAN2Initialize

Description:	This function configures the CAN module
Include:	can.h
Prototype:	<pre>void CAN1Initialize(unsigned int config1, unsigned int config2); void CAN2Initialize(unsigned int config1, unsigned int config2);</pre>
Arguments:	<p><i>config1</i> The value to be written into CiCFG1 register, the combination of the following defines.</p> <p><u>Sync jump width</u> CAN_SYNC_JUMP_WIDTH1 CAN_SYNC_JUMP_WIDTH2 CAN_SYNC_JUMP_WIDTH3 CAN_SYNC_JUMP_WIDTH4</p> <p><u>Baud Rate prescaler</u> CAN_BAUD_PRE_SCALE(x) $((x - 1) \& 0x3f) 0xC0$</p> <p><i>config2</i> The value to be written into CiCFG2 register, the combination of the following defines.</p> <p><u>CAN bus line filter selection for wake-up</u> CAN_WAKEUP_BY_FILTER_EN CAN_WAKEUP_BY_FILTER_DIS</p> <p><u>CAN propagation segment length</u> CAN_PROPAGATIONTIME_SEG_TQ(x) $((x - 1) \& 0x7) 0xC7F8$</p> <p><u>CAN phase segment 1 length</u> CAN_PHASE_SEG1_TQ(x) $((x - 1) \& 0x7) * 0x8 0xC7C7$</p> <p><u>CAN phase segment 2 length</u> CAN_PHASE_SEG2_TQ(x) $((x - 1) \& 0x7) * 0x100 0xC0FF$</p> <p><u>CAN phase segment 2 mode</u> CAN_SEG2_FREE_PROG CAN_SEG2_TIME_LIMIT_SET</p> <p><u>Sample of the CAN bus line</u> CAN_SAMPLE3TIMES CAN_SAMPLE1TIME</p>
Return Value:	None
Remarks:	This function configures the following bits of CiCFG1 and CiCFG2 registers: SJW<1:0>, BRP<5:0>, CANCAP, WAKEFIL, SEG2PH<2:0>, SEGPHTS, SAM, SEG1PH<2:0>, PRSEG<2:0>
Source File:	CAN1Initialize.c CAN2Initialize.c
Code Example:	<pre>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);</pre>

ConfigIntCAN1 ConfigIntCAN2

Description:	This function configures the CAN Interrupts
Include:	can.h
Prototype:	<pre>void ConfigIntCAN1(unsigned int <i>config1</i>, unsigned int <i>config2</i>); void ConfigIntCAN2(unsigned int <i>config1</i>, unsigned int <i>config2</i>);</pre>
Arguments:	<p><i>config1</i> individual interrupt enable/disable information as defined below: User must enter either enable or disable option for all the individual interrupts.</p> <p><u>Interrupt enable</u> 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</p> <p><u>Interrupt disable</u> 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</p> <p><i>config2</i> CAN interrupt priority and enable/disable information as defined below:</p> <p><u>CAN Interrupt enable/disable</u> CAN_INT_ENABLE CAN_INT_DISABLE</p> <p><u>CAN Interrupt priority</u> 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</p>
Return Value:	None
Remarks:	This function configures the CAN interrupts. It enables/disables the individual CAN interrupts. It also enables/disables the CAN interrupt and sets priority.
Source File:	ConfigIntCAN1.c ConfigIntCAN2.c

3.4.3 Example of Use

```
#define __dsPIC30F6014__
#include<p30fxxxx.h>
#include<can.h>
#define dataarray 0x1820
int main(void)
{
    /* Length of data to be transmitted/read */
    unsigned char datalen;
    unsigned char Txdata[] =
        {'M','I','C','R','O','C','H','I','P','\0'};
    unsigned int TXConfig, RXConfig;
    unsigned long MaskID,MessageID;
    char FilterNo,tx_rx_no;
    unsigned char * datareceived = (unsigned char *)
        dataarray; /* Holds the data received */
    /* Set request for configuration mode */
    CAN1SetOperationMode(CAN_IDLE_CON &
        CAN_MASTERCLOCK_1 &
        CAN_REQ_OPERMODE_CONFIG &
        CAN_CAPTURE_DIS);
    while(C1CTRLbits.OPMODE <=3);
    /* Load configuration register */
    CAN1Initialize(CAN_SYNC_JUMP_WIDTH2 &
        CAN_BAUD_PRE_SCALE(2),
        CAN_WAKEUP_BY_FILTER_DIS &
        CAN_PHASE_SEG2_TQ(5) &
        CAN_PHASE_SEG1_TQ(4) &
        CAN_PROPAGATIONTIME_SEG_TQ(4) &
        CAN_SEG2_FREE_PROG &
        CAN_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);
}
```

16-Bit Language Tools Libraries

```
    /* Set request for Loopback mode */
    CAN1SetOperationMode(CAN_IDLE_CON & CAN_CAPTURE_DIS &
                        CAN_MASTERCLOCK_1 &
                        CAN_REQ_OPERMODE_LOOPBK);
    while(C1CTRLbits.OPMODE !=2);
    /* Wait till message is transmitted completely */
    while(!CAN1IsTXReady(0))
    /* Wait till receive buffer contain valid message */
    while(!CAN1IsRXReady(0));
    /* Read received data from receive buffer and store it into
       user defined dataarray */
    CAN1ReceiveMessage(datareceived, datalen, tx_rx_no);
    while(1);
    return 0;
}
```

3.5 ADC12 FUNCTIONS

This section contains a list of individual functions for the 12-bit ADC and an example of use of the functions. Functions may be implemented as macros.

3.5.1 Individual Functions

BusyADC12

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

CloseADC12

Description:	This function turns off the ADC module and disables the ADC interrupts.
Include:	<code>adc12.h</code>
Prototype:	<code>void CloseADC12(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>CloseADC12.c</code>
Code Example:	<code>CloseADC12();</code>

ConfigIntADC12

Description:	This function configures the ADC interrupt.
Include:	<code>adc12.h</code>
Prototype:	<code>void ConfigIntADC12(unsigned int config);</code>
Arguments:	<i>config</i> ADC interrupt priority and enable/disable information as defined below: <u>ADC Interrupt enable/disable</u> <code>ADC_INT_ENABLE</code> <code>ADC_INT_DISABLE</code>

ConfigIntADC12 (Continued)

ADC Interrupt priority

ADC_INT_PRI_0
ADC_INT_PRI_1
ADC_INT_PRI_2
ADC_INT_PRI_3
ADC_INT_PRI_4
ADC_INT_PRI_5
ADC_INT_PRI_6
ADC_INT_PRI_7

Return Value: None

Remarks: This function clears the Interrupt Flag (ADIF) bit and then sets the interrupt priority and enables/disables the interrupt.

Source File: ConfigIntADC12.c

Code Example: ConfigIntADC12(ADC_INT_PRI_6 &
ADC_INT_ENABLE);

ConvertADC12

Description: This function starts A/D conversion.

Include: adc12.h

Prototype: void ConvertADC12(void);

Arguments: None

Return Value: None

Remarks: This function clears the ADCON1<SAMP> bit and thus stops sampling and starts conversion.
This happens only when trigger source for the A/D conversion is selected as Manual, by clearing the ADCON1 <SSRC> bits.

Source File: ConvertADC12.c

Code Example: ConvertADC12();

OpenADC12

Description: This function configures the ADC.

Include: adc12.h

Prototype: void OpenADC12(unsigned int *config1*,
unsigned int *config2*,
unsigned int *config3*,
unsigned int *configport*,
unsigned int *configscan*)

Arguments: *config1* This contains the parameters to be configured in the ADCON1 register as defined below:

Module On/Off
ADC_MODULE_ON
ADC_MODULE_OFF

Idle mode operation
ADC_IDLE_CONTINUE
ADC_IDLE_STOP

OpenADC12 (Continued)

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_TMR
ADC_CLK_INT0
ADC_CLK_MANUAL

Auto sampling select

ADC_AUTO_SAMPLING_ON
ADC_AUTO_SAMPLING_OFF

Sample enable

ADC_SAMP_ON
ADC_SAMP_OFF

config2

This contains the parameters to be configured in the ADCON2 register as defined below:

Voltage Reference

ADC_VREF_AVDD_AVSS
ADC_VREF_EXT_AVSS
ADC_VREF_AVDD_EXT
ADC_VREF_EXT_EXT

Scan selection

ADC_SCAN_ON
ADC_SCAN_OFF

Number of samples between interrupts

ADC_SAMPLES_PER_INT_1
ADC_SAMPLES_PER_INT_2
.....
ADC_SAMPLES_PER_INT_15
ADC_SAMPLES_PER_INT_16

Buffer mode select

ADC_ALT_BUF_ON
ADC_ALT_BUF_OFF

Alternate Input Sample mode select

ADC_ALT_INPUT_ON
ADC_ALT_INPUT_OFF

config3

This contains the parameters to be configured in the ADCON3 register as defined below:

Auto Sample Time bits

ADC_SAMPLE_TIME_0
ADC_SAMPLE_TIME_1
.....
ADC_SAMPLE_TIME_30
ADC_SAMPLE_TIME_31

Conversion Clock Source select

ADC_CONV_CLK_INTERNAL_RC
ADC_CONV_CLK_SYSTEM

16-Bit Language Tools Libraries

OpenADC12 (Continued)

	<u>Conversion clock select</u> ADC_CONV_CLK_Tcy2 ADC_CONV_CLK_Tcy ADC_CONV_CLK_3Tcy2 ADC_CONV_CLK_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 behavior, Data o/p format, Sample Clk Source, VREF source, No of samples/int, Buffer Fill mode, Alternate i/p sample mod, Auto sample time, Conv clock source, Conv Clock Select bits, Port Config Control bits.
Source File:	OpenADC12.c
Code Example:	<pre>OpenADC12(ADC_MODULE_OFF & ADC_IDLE_CONTINUE & ADC_FORMAT_INTG & ADC_AUTO_SAMPLING_ON, ADC_VREF_AVDD_AVSS & ADC_SCAN_OFF & ADC_BUF_MODE_OFF & ADC_ALT_INPUT_ON & ADC_SAMPLES_PER_INT_15, ADC_SAMPLE_TIME_4 & ADC_CONV_CLK_SYSTEM & ADC_CONV_CLK_Tcy, ENABLE_AN0_ANA, SKIP_SCAN_AN1 & SKIP_SCAN_AN2 & SKIP_SCAN_AN5 & SKIP_SCAN_AN7);</pre>

ReadADC12

Description: This function reads the ADC Buffer register which contains the conversion value.

Include: `adc12.h`

Prototype: `unsigned int ReadADC12(unsigned char bufIndex);`

Arguments: `bufIndex` This is the ADC buffer number which is to be read.

Return Value: None

Remarks: This function returns the contents of the ADC Buffer register. User should provide `bufIndex` value between 0 to 15 to ensure correct read of the ADCBUF0 to ADCBUFF register.

Source File: `ReadADC12.c`

Code Example:

```
unsigned int result;
result = ReadADC12(5);
```

StopSampADC12

Description: This function is identical to `ConvertADC12`.

Source File: `#define to ConvertADC12 in adc12.h`

SetChanADC12

Description: This function sets the positive and negative inputs for sample multiplexers A and B.

Include: `adc12.h`

Prototype: `void SetChanADC12(unsigned int channel);`

Arguments: `channel` This contains the input select parameter to be configured into ADCHS register as defined below:

A/D Channel 0 positive i/p select for SAMPLE A
`ADC_CH0_POS_SAMPLEA_AN0`
`ADC_CH0_POS_SAMPLEA_AN1`
.....
`ADC_CH0_POS_SAMPLEA_AN15`

A/D Channel 0 negative i/p select for SAMPLE A
`ADC_CH0_NEG_SAMPLEA_AN1`
`ADC_CH0_NEG_SAMPLEA_NVREF`

A/D Channel 0 positive i/p select for SAMPLE B
`ADC_CH0_POS_SAMPLEB_AN0`
`ADC_CH0_POS_SAMPLEB_AN1`
.....
`ADC_CH0_POS_SAMPLEB_AN15`

A/D Channel 0 negative i/p select for SAMPLE B
`ADC_CH0_NEG_SAMPLEB_AN1`
`ADC_CH0_NEG_SAMPLEB_NVREF`

Return Value: None

Remarks: This function configures the inputs for the sample multiplexers A and B by writing to the ADCHS register.

Source File: `SetChanADC12.c`

Code Example:

```
SetChanADC12(ADC_CH0_POS_SAMPLEA_AN4 &
ADC_CH0_NEG_SAMPLEA_NVREF);
```

3.5.2 Individual Macros

EnableIntADC

Description: This macro enables the ADC interrupt.
Include: `adc12.h`
Arguments: None
Remarks: This macro sets ADC Interrupt Enable bit of Interrupt Enable Control register.
Code Example: `EnableIntADC;`

DisableIntADC

Description: This macro disables the ADC interrupt.
Include: `adc12.h`
Arguments: None
Remarks: This macro clears ADC Interrupt Enable bit of Interrupt Enable Control register.
Code Example: `DisableIntADC;`

SetPriorityIntADC

Description: This macro sets priority for ADC interrupt.
Include: `adc12.h`
Arguments: *priority*
Remarks: This macro sets ADC Interrupt Priority bits of Interrupt Priority Control register.
Code Example: `SetPriorityIntADC(6);`

3.5.3 Example of Use

```
#define __dsPIC30F6014__
#include <p30fxxxx.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_CHO_POS_SAMPLEA_AN4 &
              ADC_CHO_NEG_SAMPLEA_NVREF &
              ADC_CHO_POS_SAMPLEB_AN2&
              ADC_CHO_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++;
    }
}
```

16-Bit Language Tools Libraries

3.6 ADC10 FUNCTIONS

This section contains a list of individual functions for the 10-bit ADC and an example of use of the functions. Functions may be implemented as macros.

3.6.1 Individual Functions

BusyADC10

Description:	This function returns the ADC conversion status.
Include:	<code>adc10.h</code>
Prototype:	<code>char BusyADC10(void);</code>
Arguments:	None
Return Value:	If the value of DONE is '0', then '1' is returned, indicating that the ADC is busy in conversion. If the value of DONE is '1', then '0' is returned, indicating that the ADC has completed conversion.
Remarks:	This function returns the complement of the ADCON1 <DONE> bit status which indicates whether the ADC is busy in conversion.
Source File:	<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:	<code>config</code> ADC interrupt priority and enable/disable information as defined below: <u>ADC Interrupt enable/disable</u> <code>ADC_INT_ENABLE</code> <code>ADC_INT_DISABLE</code> <u>ADC Interrupt priority</u> <code>ADC_INT_PRI_0</code> <code>ADC_INT_PRI_1</code> <code>ADC_INT_PRI_2</code> <code>ADC_INT_PRI_3</code> <code>ADC_INT_PRI_4</code> <code>ADC_INT_PRI_5</code> <code>ADC_INT_PRI_6</code> <code>ADC_INT_PRI_7</code>

ConfigIntADC10 (Continued)

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:	<pre>ConfigIntADC10(ADC_INT_PRI_3 & ADC_INT_DISABLE);</pre>

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:	<pre>ConvertADC10();</pre>

OpenADC10

Description:	This function configures the ADC.
Include:	adc10.h
Prototype:	<pre>void OpenADC10(unsigned int config1, unsigned int config2, unsigned int config3, unsigned int configport, unsigned int configscan)</pre>
Arguments:	<p><i>config1</i> 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_MPWM ADC_CLK_TMR ADC_CLK_INT0 ADC_CLK_MANUAL</p>

16-Bit Language Tools Libraries

OpenADC10 (Continued)

config2

Auto sampling select
ADC_AUTO_SAMPLING_ON
ADC_AUTO_SAMPLING_OFF

Simultaneous Sampling
ADC_SAMPLE_SIMULTANEOUS
ADC_SAMPLE_INDIVIDUAL

Sample enable
ADC_SAMP_ON
ADC_SAMP_OFF

This contains the parameters to be configured in the ADCON2 register as defined below:

Voltage Reference
ADC_VREF_AVDD_AVSS
ADC_VREF_EXT_AVSS
ADC_VREF_AVDD_EXT
ADC_VREF_EXT_EXT

Scan selection
ADC_SCAN_ON
ADC_SCAN_OFF

A/D channels utilized
ADC_CONVERT_CH0123
ADC_CONVERT_CH01
ADC_CONVERT_CH0

Number of samples between interrupts
ADC_SAMPLES_PER_INT_1
ADC_SAMPLES_PER_INT_2
.....
ADC_SAMPLES_PER_INT_15
ADC_SAMPLES_PER_INT_16

Buffer mode select
ADC_ALT_BUF_ON
ADC_ALT_BUF_OFF

Alternate Input Sample mode select
ADC_ALT_INPUT_ON
ADC_ALT_INPUT_OFF

config3

This contains the parameters to be configured in the ADCON3 register as defined below:

Auto Sample Time bits
ADC_SAMPLE_TIME_0
ADC_SAMPLE_TIME_1
.....
ADC_SAMPLE_TIME_30
ADC_SAMPLE_TIME_31

Conversion Clock Source select
ADC_CONV_CLK_INTERNAL_RC
ADC_CONV_CLK_SYSTEM

Conversion clock select
ADC_CONV_CLK_Tcy2
ADC_CONV_CLK_Tcy
ADC_CONV_CLK_3Tcy2
.....
ADC_CONV_CLK_32Tcy

OpenADC10 (Continued)

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: OpenADC10.c

Code Example:

```
OpenADC10 (ADC_MODULE_OFF &
ADC_IDLE_STOP &
ADC_FORMAT_SIGN_FRACT &
ADC_CLK_INT0 &
ADC_SAMPLE_INDIVIDUAL &
ADC_AUTO_SAMPLING_ON,
ADC_VREF_AVDD_AVSS &
ADC_SCAN_OFF &
ADC_BUF_MODE_OFF &
ADC_ALT_INPUT_ON &
ADC_CONVERT_CH0 &
ADC_SAMPLES_PER_INT_10,
ADC_SAMPLE_TIME_4 &
ADC_CONV_CLK_SYSTEM &
ADC_CONV_CLK_Tcy,
ENABLE_AN1_ANA,
SKIP_SCAN_AN0 &
SKIP_SCAN_AN3 &
SKIP_SCAN_AN4 &
SKIP_SCAN_AN5) ;
```

16-Bit Language Tools Libraries

ReadADC10

Description: This function reads the ADC Buffer register which contains the conversion value.

Include: `adc10.h`

Prototype: `unsigned int ReadADC10(unsigned char bufIndex);`

Arguments: *bufIndex* This is the ADC buffer number which is to be read.

Return Value: None

Remarks: This function returns the contents of the ADC Buffer register. User should provide *bufIndex* value between '0' to '15' to ensure correct read of the ADCBUF0 to ADCBUFF.

Source File: `ReadADC10.c`

Code Example:

```
unsigned int result;
result = ReadADC10(3);
```

StopSampADC10

Description: This function is identical to `ConvertADC10`.

Source File: `#define to ConvertADC10 in adc10.h`

SetChanADC10

Description: This function sets the positive and negative inputs for the sample multiplexers A and B.

Include: `adc10.h`

Prototype: `void SetChanADC10(unsigned int channel);`

Arguments: *channel* This contains the input select parameter to be configured into the ADCHS register as defined below:

A/D Channel 1, 2, 3 Negative input for Sample A
`ADC_CHX_NEG_SAMPLEA_AN9AN10AN11`
`ADC_CHX_NEG_SAMPLEA_AN6AN7AN8`
`ADC_CHX_NEG_SAMPLEA_NVREF`

A/D Channel 1, 2, 3 Negative input for Sample B
`ADC_CHX_NEG_SAMPLEB_AN9AN10AN11`
`ADC_CHX_NEG_SAMPLEB_AN6AN7AN8`
`ADC_CHX_NEG_SAMPLEB_NVREF`

A/D Channel 1, 2, 3 Positive input for Sample A
`ADC_CHX_POS_SAMPLEA_AN3AN4AN5`
`ADC_CHX_POS_SAMPLEA_AN0AN1AN2`

A/D Channel 1, 2, 3 Positive input for Sample B
`ADC_CHX_POS_SAMPLEA_AN3AN4AN5`
`ADC_CHX_POS_SAMPLEB_AN0AN1AN2`

A/D Channel 0 positive i/p select for Sample A
`ADC_CH0_POS_SAMPLEA_AN0`
`ADC_CH0_POS_SAMPLEA_AN1`
`.....`
`ADC_CH0_POS_SAMPLEA_AN15`

A/D Channel 0 negative i/p select for Sample A
`ADC_CH0_NEG_SAMPLEA_AN1`
`ADC_CH0_NEG_SAMPLEA_NVREF`

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);
```

3.6.2 Individual Macros

EnableIntADC

Description: This macro enables the ADC interrupt.

Include: adc10.h

Arguments: None

Remarks: This macro sets ADC Interrupt Enable bit of Interrupt Enable Control register.

Code Example:

```
EnableIntADC;
```

DisableIntADC

Description: This macro disables the ADC interrupt.

Include: adc10.h

Arguments: None

Remarks: This macro clears ADC Interrupt Enable bit of Interrupt Enable Control register.

Code Example:

```
DisableIntADC;
```

SetPriorityIntADC

Description: This macro sets priority for ADC interrupt.

Include: adc10.h

Arguments: *priority*

Remarks: This macro sets ADC Interrupt Priority bits of Interrupt Priority Control register.

Code Example:

```
SetPriorityIntADC(2);
```

16-Bit Language Tools Libraries

3.6.3 Example of Use

```
#define __dsPIC30F6010__
#include <p30fxxxx.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_CHO_POS_SAMPLEA_AN4 &
              ADC_CHO_NEG_SAMPLEA_NVREF &
              ADC_CHO_POS_SAMPLEB_AN2 &
              ADC_CHO_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++;
    }
}
```

3.7 TIMER FUNCTIONS

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

3.7.1 Individual Functions

CloseTimer1 CloseTimer2 CloseTimer3 CloseTimer4 CloseTimer5

Description: This function turns off the 16-bit timer module.

Include: timer.h

Prototype:
void CloseTimer1(void);
void CloseTimer2(void);
void CloseTimer3(void);
void CloseTimer4(void);
void CloseTimer5(void);

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: CloseTimer1.c
CloseTimer2.c
CloseTimer3.c
CloseTimer4.c
CloseTimer5.c

Code Example: CloseTimer1();

CloseTimer23 CloseTimer45

Description: This function turns off the 32-bit timer module.

Include: timer.h

Prototype:
void CloseTimer23(void);
void CloseTimer45(void);

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: CloseTimer23.c
CloseTimer45.c

Code Example: CloseTimer23();

16-Bit Language Tools Libraries

ConfigIntTimer1 ConfigIntTimer2 ConfigIntTimer3 ConfigIntTimer4 ConfigIntTimer5

Description:	This function configures the 16-bit timer interrupt.
Include:	timer.h
Prototype:	<pre>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);</pre>
Arguments:	<p><i>config</i> Timer interrupt priority and enable/disable information as defined below:</p> <pre>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</pre>
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:	<pre>ConfigIntTimer1.c ConfigIntTimer2.c ConfigIntTimer3.c ConfigIntTimer4.c ConfigIntTimer5.c</pre>
Code Example:	<pre>ConfigIntTimer1(T1_INT_PRIOR_3 & T1_INT_ON);</pre>

ConfigIntTimer23 ConfigIntTimer45

Description:	This function configures the 32-bit timer interrupt.
Include:	timer.h
Prototype:	<pre>void ConfigIntTimer23(unsigned int config); void ConfigIntTimer45(unsigned int config);</pre>
Arguments:	<p><i>config</i> Timer interrupt priority and enable/disable information as defined below:</p> <pre>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</pre>
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:	<pre>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)</pre>
Arguments:	<p><i>config</i> This contains the parameters to be configured in the TxCON register as defined below:</p> <p><u>Timer Module On/Off</u> Tx_ON Tx_OFF</p> <p><u>Timer Module Idle mode On/Off</u> Tx_IDLE_CON Tx_IDLE_STOP</p>

16-Bit Language Tools Libraries

OpenTimer1 (Continued)

OpenTimer2

OpenTimer3

OpenTimer4

OpenTimer5

Timer Gate time accumulation enable

Tx_GATE_ON

Tx_GATE_OFF

Timer prescaler

Tx_PS_1_1

Tx_PS_1_8

Tx_PS_1_64

Tx_PS_1_128

Timer Synchronous clock enable

Tx_SYNC_EXT_ON

Tx_SYNC_EXT_OFF

Timer clock source

Tx_SOURCE_EXT

Tx_SOURCE_INT

period This contains the period match value to be stored into the PR register

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

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();
```

16-Bit Language Tools Libraries

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

Source File: `WriteTimer23.c`
`WriteTimer45.c`

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

3.7.2 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);

3.7.3 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 = 0xFFF;
    OpenTimer1(T1_ON & T1_GATE_OFF & T1_IDLE_STOP &
               T1_PS_1_1 & T1_SYNC_EXT_OFF &
               T1_SOURCE_INT, match_value);
    /* Wait till the timer matches with the period value */
    while(1)
    {
        timer_value = ReadTimer1();
        if(timer_value >= 0x7FF)
        {
            PORTDbits.RD1 = 0; /* turn on LED on RD1 */
        }
    }
    CloseTimer1();
}
```

3.8 RESET/CONTROL FUNCTIONS

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

3.8.1 Individual Functions

isBOR

Description:	This function checks if Reset is due to Brown-out Reset.
Include:	reset.h
Prototype:	char isBOR(void);
Arguments:	None
Return Value:	This function returns the RCON<BOR> bit status. 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:	<pre>char reset_state; reset_state = isBOR();</pre>

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:	<pre>char reset_state; reset_state = isPOR();</pre>

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:	<pre>char lvd; lvd = isLVD();</pre>

16-Bit Language Tools Libraries

isMCLR

Description: This function checks if Reset condition is due to $\overline{\text{MCLR}}$ 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 $\overline{\text{MCLR}}$ pin going low.
If return value is '0', then Reset is not due to $\overline{\text{MCLR}}$ 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

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 Wake-up 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 Wake-up from Sleep is due to MCLR, POR, BOR or any interrupt.
Include:	<code>reset.h</code>
Prototype:	<code>char isWU(void);</code>
Arguments:	None
Return Value:	This function checks if Wake-up from Sleep has occurred. If yes, it checks for the cause for wake-up. if '1', wake-up is due to the occurrence of interrupt. if '2', wake-up is due to MCLR. if '3', wake-up is due to POR. if '4', wake-up is due to BOR. If Wake-up from Sleep has not occurred, then a value of '0' is returned.
Remarks:	None
Source File:	<code>isWU.c</code>
Code Example:	<pre>char reset_state; reset_state = isWU();</pre>

3.8.2 Individual Macros

DisableInterrupts

Description:	This macro disables all the peripheral interrupts for specified number of instruction cycles.
Include:	<code>reset.h</code>
Arguments:	<i>cycles</i>
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>

16-Bit Language Tools Libraries

WDTSWEnable

Description:	This macro turns on the Watchdog Timer
Include:	reset.h
Arguments:	None
Remarks:	This macro sets Software WDT Enable (SWDTEN) bit of RCON register
Code Example:	WDTSWEnable;

WDTSWDisable

Description:	This macro clears Software WDT Enable (SWDTEN) bit of RCON register
Include:	reset.h
Arguments:	None
Remarks:	This macro disables WDT if FWDTEN Fuse bit is '0'.
Code Example:	WDTSWDisable;

3.9 I/O PORT FUNCTIONS

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

3.9.1 Individual Functions

CloseINT0 CloseINT1 CloseINT2 CloseINT3 CloseINT4

Description:	This function disables the external interrupt on INT pin.
Include:	ports.h
Prototype:	void CloseINT0(void); void CloseINT1(void); void CloseINT2(void); void CloseINT3(void); void CloseINT4(void);
Arguments:	None
Return Value:	None
Remarks:	This function disables the interrupt on INT pin and clears the corresponding Interrupt flag.
Source File:	CloseInt0.c CloseInt1.c CloseInt2.c CloseInt3.c CloseInt4.c
Code Example:	CloseINT0();

ConfigINT0 ConfigINT1 ConfigINT2 ConfigINT3 ConfigINT4

Description:	This function configures the interrupt on INT pin.
Include:	ports.h
Prototype:	<pre>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);</pre>
Arguments:	<p><i>config</i> Interrupt edge, priority and enable/disable information as defined below:</p> <p><u>Interrupt edge selection</u> RISING_EDGE_INT FALLING_EDGE_INT</p> <p><u>Interrupt enable</u> INT_ENABLE INT_DISABLE</p> <p><u>Interrupt priority</u> INT_PRI_0 INT_PRI_1 INT_PRI_2 INT_PRI_3 INT_PRI_4 INT_PRI_5 INT_PRI_6 INT_PRI_7</p>
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:	<pre>ConfigINT0(RISING_EDGE_INT & EXT_INT_PRI_5 & EXT_INT_ENABLE);</pre>

16-Bit Language Tools Libraries

ConfigCNPullups

Description:	This function configures the pull-up resistors for CN pins.
Include:	<code>ports.h</code>
Prototype:	<code>void ConfigCNPullups(long int <i>config</i>);</code>
Arguments:	<i>config</i> This is the 32-bit value for configuring pull-ups. The lower word is stored into CNPU1 register and next upper word is stored into CNPU2 register. The upper 8 bits of CNPU2 register are unimplemented.
Return Value:	None
Remarks:	None
Source File:	<code>ConfigCNPullups.c</code>
Code Example:	<code>ConfigCNPullups(0xFFFF);</code>

ConfigIntCN

Description:	This function configures the CN interrupts.
Include:	<code>ports.h</code>
Prototype:	<code>void ConfigIntCN(long int <i>config</i>);</code>
Arguments:	<i>config</i> This is the 32-bit value for configuring the CN interrupts. The lower 24 bits contain the individual enable/disable information for the CN interrupts. Setting bit <i>x</i> (<i>x</i> = 0, 1, ..., 23) would enable the CN <i>x</i> interrupt. The upper most byte of <i>config</i> 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:	<code>ConfigIntCN.c</code>
Code Example:	<pre>// This would enable CN0, CN1, CN2 and CN7 only. ConfigIntCN(CHANGE_INT_OFF & CHANGE_INT_PRI_4 & 0xFF000087);</pre>

3.9.2 Individual Macros

EnableCN0
EnableCN1
EnableCN2

.....
EnableCN23

Description: This macro enables the individual change notification interrupt.
Include: ports.h
Arguments: None
Remarks: None
Code Example: EnableCN6;

DisableCN0
DisableCN1
DisableCN2

.....
DisableCN23

Description: This macro disables individual change notification interrupt.
Include: ports.h
Arguments: None
Remarks: None
Code Example: DisableCN14;

EnableINT0
EnableINT1
EnableINT2
EnableINT3
EnableINT4

Description: This macro enables the individual external interrupt.
Include: ports.h
Arguments: None
Remarks: None
Code Example: EnableINT2;

16-Bit Language Tools Libraries

DisableINT0
DisableINT1
DisableINT2
DisableINT3
DisableINT4

Description: This macro disables the individual external interrupt.
Include: `ports.h`
Arguments: None
Remarks: None
Code Example: `DisableINT2;`

SetPriorityInt0
SetPriorityInt1
SetPriorityInt2
SetPriorityInt3
SetPriorityInt4

Description: This macro sets priority for external interrupts.
Include: `ports.h`
Arguments: `priority`
Remarks: This macro sets External Interrupt Priority bits of Interrupt Priority Control register.
Code Example: `SetPriorityInt4(6);`

3.10 INPUT CAPTURE FUNCTIONS

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

3.10.1 Individual Functions

CloseCapture1

CloseCapture2

CloseCapture3

CloseCapture4

CloseCapture5

CloseCapture6

CloseCapture7

CloseCapture8

Description: This function turns off the Input Capture module.

Include: InCap.h

Prototype:

```
void CloseCapture1(void);  
void CloseCapture2(void);  
void CloseCapture3(void);  
void CloseCapture4(void);  
void CloseCapture5(void);  
void CloseCapture6(void);  
void CloseCapture7(void);  
void CloseCapture8(void);
```

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:

```
CloseCapture1.c  
CloseCapture2.c  
CloseCapture3.c  
CloseCapture4.c  
CloseCapture5.c  
CloseCapture6.c  
CloseCapture7.c  
CloseCapture8.c
```

Code Example: CloseCapture1();

16-Bit Language Tools Libraries

ConfigIntCapture1
ConfigIntCapture2
ConfigIntCapture3
ConfigIntCapture4
ConfigIntCapture5
ConfigIntCapture6
ConfigIntCapture7
ConfigIntCapture8

Description: This function configures the Input Capture interrupt.

Include: InCap.h

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

Code Example:

```
OpenCapture1(IC_IDLE_CON & IC_TIMER2_SRC &  
IC_INT_1CAPTURE & IC_EVERY_RISE_EDGE);
```

16-Bit Language Tools Libraries

ReadCapture1
ReadCapture2
ReadCapture3
ReadCapture4
ReadCapture5
ReadCapture6
ReadCapture7
ReadCapture8

Description: This function reads all the pending Input Capture buffers.

Include: InCap.h

Prototype:

```
void ReadCapture1(unsigned int *buffer);  
void ReadCapture2(unsigned int *buffer);  
void ReadCapture3(unsigned int *buffer);  
void ReadCapture4(unsigned int *buffer);  
void ReadCapture5(unsigned int *buffer);  
void ReadCapture6(unsigned int *buffer);  
void ReadCapture7(unsigned int *buffer);  
void ReadCapture8(unsigned int *buffer);
```

Arguments: *buffer* This is the pointer to the locations where the data read from the Input Capture buffers have to be stored.

Return Value: None

Remarks: This function reads all the pending Input Capture buffers until the buffers are empty indicated by the ICxCON<ICBNE> bit getting cleared.

Source File:

```
ReadCapture1.c  
ReadCapture2.c  
ReadCapture3.c  
ReadCapture4.c  
ReadCapture5.c  
ReadCapture6.c  
ReadCapture7.c  
ReadCapture8.c
```

Code Example:

```
unsigned int *buffer = 0x1900;  
ReadCapture1(buffer);
```

3.10.2 Individual Macros

EnableIntIC1
EnableIntIC2
EnableIntIC3
EnableIntIC4
EnableIntIC5
EnableIntIC6
EnableIntIC7
EnableIntIC8

Description: This macro enables the interrupt on capture event.
Include: InCap.h
Arguments: None
Remarks: This macro sets Input Capture Interrupt Enable bit of Interrupt Enable Control register.
Code Example: EnableIntIC7;

DisableIntIC1
DisableIntIC2
DisableIntIC3
DisableIntIC4
DisableIntIC5
DisableIntIC6
DisableIntIC7
DisableIntIC8

Description: This macro disables the interrupt on capture event.
Include: InCap.h
Arguments: None
Remarks: This macro clears Input Capture Interrupt Enable bit of Interrupt Enable Control register.
Code Example: DisableIntIC7;

SetPriorityIntIC1
SetPriorityIntIC2
SetPriorityIntIC3
SetPriorityIntIC4
SetPriorityIntIC5
SetPriorityIntIC6
SetPriorityIntIC7
SetPriorityIntIC8

Description: This macro sets priority for input capture interrupt.
Include: InCap.h
Arguments: priority
Remarks: This macro sets Input Capture Interrupt Priority bits of Interrupt Priority Control register.
Code Example: SetPriorityIntIC4(1);

16-Bit Language Tools Libraries

3.10.3 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();
}
```

3.11 OUTPUT COMPARE FUNCTIONS

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

3.11.1 Individual Functions

CloseOC1
CloseOC2
CloseOC3
CloseOC4
CloseOC5
CloseOC6
CloseOC7
CloseOC8

Description: This function turns off the Output Compare module.

Include: outcompare.h

Prototype:

```
void CloseOC1(void);  
void CloseOC2(void);  
void CloseOC3(void);  
void CloseOC4(void);  
void CloseOC5(void);  
void CloseOC6(void);  
void CloseOC7(void);  
void CloseOC8(void);
```

Arguments: None

Return Value: None

Remarks: This function disables the Output Compare interrupt and then turns off the module. The Interrupt Flag bit is also cleared.

Source File:

```
CloseOC1.c  
CloseOC2.c  
CloseOC3.c  
CloseOC4.c  
CloseOC5.c  
CloseOC6.c  
CloseOC7.c  
CloseOC8.c
```

Code Example: CloseOC1();

16-Bit Language Tools Libraries

ConfigIntOC1
ConfigIntOC2
ConfigIntOC3
ConfigIntOC4
ConfigIntOC5
ConfigIntOC6
ConfigIntOC7
ConfigIntOC8

Description: This function configures the Output Compare interrupt.

Include: `outcompare.h`

Prototype:
`void ConfigIntOC1(unsigned int config);`
`void ConfigIntOC2(unsigned int config);`
`void ConfigIntOC3(unsigned int config);`
`void ConfigIntOC4(unsigned int config);`
`void ConfigIntOC5(unsigned int config);`
`void ConfigIntOC6(unsigned int config);`
`void ConfigIntOC7(unsigned int config);`
`void ConfigIntOC8(unsigned int config);`

Arguments: *config* Output Compare interrupt priority and enable/disable information as defined below:
Interrupt enable/disable
`OC_INT_ON`
`OC_INT_OFF`
Interrupt Priority
`OC_INT_PRIOR_0`
`OC_INT_PRIOR_1`
`OC_INT_PRIOR_2`
`OC_INT_PRIOR_3`
`OC_INT_PRIOR_4`
`OC_INT_PRIOR_5`
`OC_INT_PRIOR_6`
`OC_INT_PRIOR_7`

Return Value: None

Remarks: This function clears the Interrupt Flag bit and then sets the interrupt priority and enables/disables the interrupt.

Source File: `ConfigIntOC1.c`
`ConfigIntOC2.c`
`ConfigIntOC3.c`
`ConfigIntOC4.c`
`ConfigIntOC5.c`
`ConfigIntOC6.c`
`ConfigIntOC7.c`
`ConfigIntOC8.c`

Code Example: `ConfigIntOC1(OC_INT_ON & OC_INT_PRIOR_2);`

OpenOC1
OpenOC2
OpenOC3
OpenOC4
OpenOC5
OpenOC6
OpenOC7
OpenOC8

Description: This function configures the Output Compare module.

Include: outcompare.h

Prototype:

```
void OpenOC1(unsigned int config,
             unsigned int value1, unsigned int value2);
void OpenOC2(unsigned int config,
             unsigned int value1, unsigned int value2);
void OpenOC3(unsigned int config,
             unsigned int value1, unsigned int value2);
void OpenOC4(unsigned int config,
             unsigned int value1, unsigned int value2);
void OpenOC5(unsigned int config,
             unsigned int value1, unsigned int value2);
void OpenOC6(unsigned int config,
             unsigned int value1, unsigned int value2);
void OpenOC7(unsigned int config,
             unsigned int value1, unsigned int value2);
void OpenOC8(unsigned int config,
             unsigned int value1, unsigned int value2);
```

Arguments:

config This contains the parameters to be configured in the OCxCON register as defined below:

- Idle mode operation
 - OC_IDLE_STOP
 - OC_IDLE_CON
- Clock select
 - OC_TIMER2_SRC
 - OC_TIMER3_SRC
- Output Compare modes of operation
 - OC_PWM_FAULT_PIN_ENABLE
 - OC_PWM_FAULT_PIN_DISABLE
 - OC_CONTINUE_PULSE
 - OC_SINGLE_PULSE
 - OC_TOGGLE_PULSE
 - OC_HIGH_LOW
 - OC_LOW_HIGH
 - OC_OFF

value1 This contains the value to be stored into OCxRS Secondary Register.

value2 This contains the value to be stored into OCxR Main Register.

Return Value: None

16-Bit Language Tools Libraries

OpenOC1 (Continued)

OpenOC2

OpenOC3

OpenOC4

OpenOC5

OpenOC6

OpenOC7

OpenOC8

Remarks: This function configures the Output Compare Module Control register (OCxCON) with the following parameters:
Clock select, mode of operation, operation in Idle mode.
It also configures the OCxRS and OCxR registers.

Source File: OpenOC1.c
OpenOC2.c
OpenOC3.c
OpenOC4.c
OpenOC5.c
OpenOC6.c
OpenOC7.c
OpenOC8.c

Code Example: `OpenOC1(OC_IDLE_CON & OC_TIMER2_SRC & OC_PWM_FAULT_PIN_ENABLE, 0x80, 0x60);`

ReadDCOC1PWM
ReadDCOC2PWM
ReadDCOC3PWM
ReadDCOC4PWM
ReadDCOC5PWM
ReadDCOC6PWM
ReadDCOC7PWM
ReadDCOC8PWM

Description: This function reads the duty cycle from the Output Compare Secondary register.

Include: `outcompare.h`

Prototype:

```
unsigned int ReadDCOC1PWM(void);  
unsigned int ReadDCOC2PWM(void);  
unsigned int ReadDCOC3PWM(void);  
unsigned int ReadDCOC4PWM(void);  
unsigned int ReadDCOC5PWM(void);  
unsigned int ReadDCOC6PWM(void);  
unsigned int ReadDCOC7PWM(void);  
unsigned int ReadDCOC8PWM(void);
```

Arguments: None

Return Value: This function returns the content of OCxRS register when Output Compare module is in PWM mode. Else '-1' is returned

Remarks: This function reads the duty cycle from the Output Compare Secondary register (OCxRS) when Output Compare module is in PWM mode. If not in PWM mode, the functions returns a value of '-1'.

Source File:

```
ReadDCOC1PWM.c  
ReadDCOC2PWM.c  
ReadDCOC3PWM.c  
ReadDCOC4PWM.c  
ReadDCOC5PWM.c  
ReadDCOC6PWM.c  
ReadDCOC7PWM.c  
ReadDCOC8PWM.c
```

Code Example:

```
unsigned int compare_reg;  
compare_reg = ReadDCOC1PWM();
```

16-Bit Language Tools Libraries

ReadRegOC1
ReadRegOC2
ReadRegOC3
ReadRegOC4
ReadRegOC5
ReadRegOC6
ReadRegOC7
ReadRegOC8

Description: This function reads the duty cycle registers when Output Compare module is not in PWM mode.

Include: `outcompare.h`

Prototype:
`unsigned int ReadRegOC1(char reg);`
`unsigned int ReadRegOC2(char reg);`
`unsigned int ReadRegOC3(char reg);`
`unsigned int ReadRegOC4(char reg);`
`unsigned int ReadRegOC5(char reg);`
`unsigned int ReadRegOC6(char reg);`
`unsigned int ReadRegOC7(char reg);`
`unsigned int ReadRegOC8(char reg);`

Arguments: *reg* This indicates if the read should happen from the main or secondary duty cycle registers of Output Compare module.
If *reg* is '1', then the contents of Main Duty Cycle register (OCxR) is read.
If *reg* is '0', then the contents of Secondary Duty Cycle register (OCxRS) is read.

Return Value: If *reg* is '1', then the contents of Main Duty Cycle register (OCxR) is read.
If *reg* is '0', then the contents of Secondary Duty Cycle register (OCxRS) is read.
If Output Compare module is in PWM mode, '-1' is returned.

Remarks: The read of Duty Cycle register happens only when Output Compare module is not in PWM mode. Else, a value of '-1' is returned.

Source File: `ReadRegOC1.c`
`ReadRegOC2.c`
`ReadRegOC3.c`
`ReadRegOC4.c`
`ReadRegOC5.c`
`ReadRegOC6.c`
`ReadRegOC7.c`
`ReadRegOC8.c`

Code Example:
`unsigned int dutycycle_reg;`
`dutycycle_reg = ReadRegOC1(1);`

SetDCOC1PWM
SetDCOC2PWM
SetDCOC3PWM
SetDCOC4PWM
SetDCOC5PWM
SetDCOC6PWM
SetDCOC7PWM
SetDCOC8PWM

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.

Source File:

```
SetDCOC1PWM.c  
SetDCOC2PWM.c  
SetDCOC3PWM.c  
SetDCOC4PWM.c  
SetDCOC5PWM.c  
SetDCOC6PWM.c  
SetDCOC7PWM.c  
SetDCOC8PWM.c
```

Code Example: SetDCOC1PWM(*dutycycle*);

16-Bit Language Tools Libraries

SetPulseOC1
SetPulseOC2
SetPulseOC3
SetPulseOC4
SetPulseOC5
SetPulseOC6
SetPulseOC7
SetPulseOC8

Description: This function configures the Output Compare main and secondary registers (OCxR and OCxRS) when the module is not in PWM mode.

Include: outcompare.h

Prototype:

```
void SetPulseOC1(unsigned int pulse_start,  
                unsigned int pulse_stop);  
void SetPulseOC2(unsigned int pulse_start,  
                unsigned int pulse_stop);  
void SetPulseOC3(unsigned int pulse_start,  
                unsigned int pulse_stop);  
void SetPulseOC4(unsigned int pulse_start,  
                unsigned int pulse_stop);  
void SetPulseOC5(unsigned int pulse_start,  
                unsigned int pulse_stop);  
void SetPulseOC6(unsigned int pulse_start,  
                unsigned int pulse_stop);  
void SetPulseOC7(unsigned int pulse_start,  
                unsigned int pulse_stop);  
void SetPulseOC8(unsigned int pulse_start,  
                unsigned int pulse_stop);
```

Arguments:

pulse_start This is the value to be stored into Output Compare Main register (OCxR).

pulse_stop This is the value to be stored into Output Compare Secondary register (OCxRS).

Return Value: None

Remarks: The Output Compare duty cycle registers (OCxR and OCxRS) will be configured with new values only if the module is not in PWM mode.

Source File:

```
SetPulseOC1.c  
SetPulseOC2.c  
SetPulseOC3.c  
SetPulseOC4.c  
SetPulseOC5.c  
SetPulseOC6.c  
SetPulseOC7.c  
SetPulseOC8.c
```

Code Example:

```
pulse_start = 0x40;  
pulse_stop = 0x60;  
SetPulseOC1(pulse_start, pulse_stop);
```

3.11.2 Individual Macros

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`
Arguments: None
Remarks: This macro clears OC Interrupt Enable bit of Interrupt Enable Control register.
Code Example: `DisableIntOC7;`

SetPriorityIntIC1
SetPriorityIntIC2
SetPriorityIntIC3
SetPriorityIntIC4
SetPriorityIntIC5
SetPriorityIntIC6
SetPriorityIntIC7
SetPriorityIntIC8

Description: This macro sets priority for output compare interrupt.
Include: `outcompare.h`
Arguments: *priority*
Remarks: This macro sets OC Interrupt Priority bits of Interrupt Priority Control register.
Code Example: `SetPriorityIntOC4(0);`

3.11.3 Example of Use

```
#define __dsPIC30F6014__
#include<p30fxxxx.h>
#include<outcompare.h>
/* This is ISR corresponding to OC1 interrupt */
void __attribute__((__interrupt__)) _OC1Interrupt(void)
{
    IFS0bits.OC1IF = 0;
}
int main(void)
{
    /* Holds the value at which OCx Pin to be driven high */
    unsigned int pulse_start ;
    /* Holds the value at which OCx Pin to be driven low */
    unsigned int pulse_stop;
    /* Turn off OC1 module */
    CloseOC1();
    /* Configure output compare1 interrupt */
    ConfigIntOC1(OC_INT_OFF & OC_INT_PRIOR_5);
    /* Configure OC1 module for required pulse width */
    pulse_start = 0x40;
    pulse_stop = 0x60;
    PR3 = 0x80 ;
    PR1 = 0xffff;
    TMR1 = 0x0000;
    T3CON = 0x8000;
    T1CON = 0x8000;
    /* Configure Output Compare module to 'initialise OCx pin
    low and generate continuous pulse'mode */
    OpenOC1(OC_IDLE_CON & OC_TIMER3_SRC &
            OC_CONTINUE_PULSE,
            pulse_stop, pulse_start);
    /* Generate continuous pulse till TMR1 reaches 0xff00 */
    while(TMR1<= 0xff00);
    asm("nop");
    CloseOC1();
    return 0;
}
```

3.12 UART FUNCTIONS

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

3.12.1 Individual Functions

BusyUART1

BusyUART2

Description:	This function returns the UART transmission status.
Include:	uart.h
Prototype:	char BusyUART1(void); char BusyUART2(void);
Arguments:	None
Return Value:	If '1' is returned, it indicates that UART is busy in transmission and UxSTA<TRMT> bit is '0'. If '0' is returned, it indicates that UART is not busy and UxSTA<TRMT> bit is '1'.
Remarks:	This function returns the status of the UART. This indicates if the UART is busy in transmission as indicated by the UxSTA<TRMT> bit.
Source File:	BusyUART1.c BusyUART2.c
Code Example:	while (BusyUART1 ());

CloseUART1

CloseUART2

Description:	This function turns off the UART module
Include:	uart.h
Prototype:	void CloseUART1(void); void CloseUART2(void);
Arguments:	None
Return Value:	None
Remarks:	This function first turns off the UART module and then disables the UART transmit and receive interrupts. The Interrupt Flag bits are also cleared.
Source File:	CloseUART1.c CloseUART2.c
Code Example:	CloseUART1 ();

16-Bit Language Tools Libraries

ConfigIntUART1 ConfigIntUART2

Description:	This function configures the UART Interrupts.
Include:	uart.h
Prototype:	<pre>void ConfigIntUART1(unsigned int <i>config</i>); void ConfigIntUART2(unsigned int <i>config</i>);</pre>
Arguments:	<p><i>config</i> Individual interrupt enable/disable information as defined below:</p> <p><u>Receive Interrupt enable</u> UART_RX_INT_EN UART_RX_INT_DIS</p> <p><u>Receive Interrupt Priority</u> 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</p> <p><u>Transmit Interrupt enable</u> UART_TX_INT_EN UART_TX_INT_DIS</p> <p><u>Transmit Interrupt Priority</u> 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</p>
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:	<pre>ConfigIntUART1(UART_RX_INT_EN & UART_RX_INT_PR5 & UART_TX_INT_EN & UART_TX_INT_PR3);</pre>

DataRdyUART1 DataRdyUART2

Description:	This function returns the UART receive buffer status.
Include:	uart.h
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 <i>config1</i> , unsigned int <i>config2</i> , unsigned int <i>ubrg</i>); void OpenUART2(unsigned int <i>config1</i> , unsigned int <i>config2</i> , unsigned int <i>ubrg</i>);
Arguments:	<i>config1</i> This contains the parameters to be configured in the UxMODE register as defined below: <u>UART enable/disable</u> UART_EN UART_DIS <u>UART Idle mode operation</u> UART_IDLE_CON UART_IDLE_STOP <u>UART communication with ALT pins</u> 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. <u>UART Wake-up on Start</u> UART_EN_WAKE UART_DIS_WAKE <u>UART Loopback mode enable/disable</u> UART_EN_LOOPBACK UART_DIS_LOOPBACK <u>Input to Capture module</u> UART_EN_ABAUD UART_DIS_ABAUD

16-Bit Language Tools Libraries

OpenUART1 (Continued) OpenUART2

Parity and data bits select

UART_NO_PAR_9BIT
UART_ODD_PAR_8BIT
UART_EVEN_PAR_8BIT
UART_NO_PAR_8BIT

Number of Stop bits

UART_2STOPBITS
UART_1STOPBIT

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

UART Transmission mode interrupt select

UART_INT_TX_BUF_EMPTY
UART_INT_TX

UART Transmit Break bit

UART_TX_PIN_NORMAL
UART_TX_PIN_LOW

UART transmit enable/disable

UART_TX_ENABLE
UART_TX_DISABLE

UART Receive Interrupt mode select

UART_INT_RX_BUF_FUL
UART_INT_RX_3_4_FUL
UART_INT_RX_CHAR

UART address detect enable/disable

UART_ADR_DETECT_EN
UART_ADR_DETECT_DIS

UART OVERRUN bit clear

UART_RX_OVERRUN_CLEAR

ubrg This is the value to be written into UxBRG register to set the baud rate.

Return Value: None

Remarks: This functions configures the UART transmit and receive sections and sets the communication baud rate.

Source File: OpenUART1.c
OpenUART2.c

Code Example:

```
baud = 5;
UMODEvalue = UART_EN & UART_IDLE_CON &
              UART_DIS_WAKE & UART_EN_LOOPBACK &
              UART_EN_ABAUD & UART_NO_PAR_8BIT &
              UART_1STOPBIT;
U1STAvalue = UART_INT_TX_BUF_EMPTY &
              UART_TX_PIN_NORMAL &
              UART_TX_ENABLE &
              UART_INT_RX_3_4_FUL &
              UART_ADR_DETECT_DIS &
              UART_RX_OVERRUN_CLEAR;
OpenUART1(U1MODEvalue, U1STAvalue, baud);
```

ReadUART1 ReadUART2

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

16-Bit Language Tools Libraries

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:	<p><i>length</i> This is the length of the string to be received.</p> <p><i>buffer</i> This is the pointer to the location where the data received have to be stored.</p> <p><i>uart_data_wait</i> This is the time-out count for which the module has to wait before return. If the time-out count is 'N', the actual time out would be about (19 * N – 1) instruction cycles.</p>
Return Value:	This function returns the number of bytes yet to be received. If the return value is '0', it indicates that the complete string has been received. If the return value is non-zero, it indicates that the complete string has not been received.
Remarks:	None
Source File:	getsUART1.c getsUART2.c
Code Example:	<pre>Datarem = getsUART1(6, Rxdata_loc, 40);</pre>

putsUART1 putsUART2

Description:	This function writes a string of data to be transmitted into the UART transmit buffer.
Include:	uart.h
Prototype:	<pre>void putsUART1(unsigned int *buffer); void putsUART2(unsigned int *buffer);</pre>
Arguments:	<i>buffer</i> This is the pointer to the string of data to be transmitted.
Return Value:	None
Remarks:	This function writes the data to be transmitted into the transmit buffer until NULL character is encountered. Once the transmit buffer is full, it waits until data gets transmitted and then writes the next data into the Transmit register.
Source File:	putsUART1.c putsUART2.c
Code Example:	<pre>putsUART1(Txdata_loc);</pre>

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

3.12.2 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;

16-Bit Language Tools Libraries

DisableIntU1TX DisableIntU2TX

Description: This macro disables the UART transmit interrupt.
Include: `uart.h`
Arguments: None
Remarks: This macro clears UART Transmit Interrupt Enable bit of Interrupt Enable Control register.
Code Example: `DisableIntU1TX;`

SetPriorityIntU1RX SetPriorityIntU2RX

Description: This macro sets priority for UART receive interrupt.
Include: `uart.h`
Arguments: *priority*
Remarks: This macro sets UART Receive Interrupt Priority bits of Interrupt Priority Control register.
Code Example: `SetPriorityIntU1RX(6);`

SetPriorityIntU1TX SetPriorityIntU2TX

Description: This macro sets priority for UART transmit interrupt.
Include: `uart.h`
Arguments: *priority*
Remarks: This macro sets UART Transmit Interrupt Priority bits of Interrupt Priority Control register.
Code Example: `SetPriorityIntU1TX(5);`

3.12.3 Example of Use

```
#define __dsPIC30F6014__
#include<p30fxxxx.h>
#include<uart.h>
/* Received data is stored in array Buf */
char Buf[80];
char * Receiveddata = Buf;
/* This is UART1 transmit ISR */
void __attribute__((__interrupt__)) _U1TXInterrupt(void)
{
    IFS0bits.U1TXIF = 0;
}
/* This is UART1 receive ISR */
void __attribute__((__interrupt__)) _U1RXInterrupt(void)
{
    IFS0bits.U1RXIF = 0;
    /* Read the receive buffer till atleast one or more character can be
    read */
    while( DataRdyUART1())
    {
        ( *( Receiveddata)++) = ReadUART1();
    }
}
int main(void)
{
    /* Data to be transmitted using UART communication module */
    char Txdata[] = {'M','i','c','r','o','c','h','i','p',' ',
                    ' ','I','C','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;
    U1STAvalue = UART_INT_TX_BUF_EMPTY &
                UART_TX_PIN_NORMAL &
                UART_TX_ENABLE & UART_INT_RX_3_4_FUL &
                UART_ADR_DETECT_DIS &
                UART_RX_OVERRUN_CLEAR;
    OpenUART1(U1MODEvalue, U1STAvalue, baudvalue);
}
```

16-Bit Language Tools Libraries

```
/* Load transmit buffer and transmit the same till null character is
encountered */
    putsUART1 ((unsigned int *)Txdata);
/* Wait for transmission to complete */
    while(BusyUART1());
/* Read all the data remaining in receive buffer which are unread */
    while(DataRdyUART1())
    {
        (*( Receiveddata)++) = ReadUART1() ;
    }
/* Turn off UART1 module */
    CloseUART1();
    return 0;
}
```

3.13 DCI FUNCTIONS

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

3.13.1 Individual Functions

CloseDCI

Description:	This function turns off the DCI module
Include:	dci.h
Prototype:	void CloseDCI(void);
Arguments:	None
Return Value:	None
Remarks:	This function first turns off the DCI module and then disables the DCI interrupt. The Interrupt Flag bit is also cleared.
Source File:	CloseDCI.c
Code Example:	CloseDCI();

BufferEmptyDCI

Description:	This function returns the DCI Transmit Buffer Full status.
Include:	dci.h
Prototype:	char BufferEmptyDCI(void);
Arguments:	None
Return Value:	If the value of TMPTY is '1', then '1' is returned, indicating that the transmit buffer is empty. If the value of TMPTY is '0', then '0' is returned, indicating that the transmit buffer is not empty.
Remarks:	This function returns the status of the DCISTAT<TMPTY> bit. This bit indicates whether the transmit buffer is empty.
Source File:	BufferEmptyDCI.c
Code Example:	while(!BufferEmptyDCI());

ConfigIntDCI

Description: This function configures the DCI interrupt.

Include: `dci.h`

Prototype: `void ConfigIntDCI(unsigned int config);`

Arguments: `config` DCI interrupt priority and enable/disable information as defined below:

DCI Interrupt enable/disable
DCI_INT_ON
DCI_INT_OFF

DCI Interrupt priority
DCI_INT_PRI_0
DCI_INT_PRI_1
DCI_INT_PRI_2
DCI_INT_PRI_3
DCI_INT_PRI_4
DCI_INT_PRI_5
DCI_INT_PRI_6
DCI_INT_PRI_7

Return Value: None

Remarks: This function clears the Interrupt Flag (DCIIF) bit and then sets the interrupt priority and enables/disables the interrupt.

Source File: `ConfigIntDCI.c`

Code Example: `ConfigIntDCI(DCI_INT_PRI_6 & DCI_INT_ENABLE);`

DataRdyDCI

Description: This function returns the status of DCI receive buffers.

Include: `dci.h`

Prototype: `char DataRdyDCI(void);`

Arguments: None

Return Value: If the value of RFUL is '1', then '1' is returned, indicating that the data is ready to be read from the receive buffers.
If the value of RFUL is '0', then '0' is returned, indicating that the receive buffers are empty.

Remarks: This function returns the status of the DCI<STAT><RFUL> bit. This bit indicates whether the data is available in the receive buffers.

Source File: `DataRdyDCI.c`

Code Example: `while(!DataRdyDCI());`

OpenDCI

Description: This function configures the DCI.

Include: `dci.h`

Prototype: `void OpenDCI(unsigned int config1,
 unsigned int config2,
 unsigned int config3,
 unsigned int trans_mask,
 unsigned int recv_mask)`

Arguments: `config1` This contains the parameters to be configured in the DCION1 register as defined below:

16-Bit Language Tools Libraries

OpenDCI (Continued)

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

CSDK 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

DCI Frame sync generator control

DCI_FRAME_LEN_16
DCI_FRAME_LEN_15
DCI_FRAME_LEN_14
.....
DCI_FRAME_LEN_1

DCI data word size

DCI_DATA_WORD_16
DCI_DATA_WORD_15
DCI_DATA_WORD_14
.....
DCI_DATA_WORD_5
DCI_DATA_WORD_4

OpenDCI (Continued)

config3 This contains the bit clock generator value to be configured in the DCICON3 register.

*trans_mask /
recv_mask* This contains the transmit/receive slot Enable bits to be configured into the TSCON/RSCON register as defined below:

DCI_DIS_SLOT_15
DCI_DIS_SLOT_14
.....
DCI_DIS_SLOT_1
DCI_DIS_SLOT_0
DCI_EN_SLOT_ALL
DCI_DIS_SLOT_ALL

Return Value: None

Remarks: This routine configures the following parameters:

1. DCICON1 register:
 - Enable bit,
 - Frame Sync mode,
 - Data Justification,
 - Sample Clock Direction,
 - Sample Clock,
 - Edge Control,
 - Output Frame Synchronization Directions Control,
 - Continuous Transmit/Receive mode,
 - Underflow mode.
2. DCICON2 register:
 - Frame Sync Generator Control,
 - Data Word Size bits,
 - Buffer Length Control bits.
3. DCICON3 register: Clock Generator Control bits
4. TSCON register: Transmit Time Slot Enable Control bits.
5. RSCON register: Receive Time Slot Enable Control bits.

Source File: OpenDCI.c

Code Example:

```
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);
```

16-Bit Language Tools Libraries

ReadDCI

Description: This function reads the contents of DCI receive buffer.

Include: `dci.h`

Prototype: `unsigned int ReadDCI(unsigned char buffer);`

Arguments: *buffer* This is the DCI buffer number to be read.

Return Value: None

Remarks: This function returns the contents of DCI receive buffer pointed by the *buffer*.

Source File: `ReadDCI.c`

Code Example:

```
unsigned int DCI_buf0;
DCI_buf0 = ReadDCI(0);
```

WriteDCI

Description: This function writes the data to be transmitted to the DCI transmit buffer.

Include: `dci.h`

Prototype: `void WriteDCI(unsigned int data_out,
 unsigned char buffer);`

Arguments: *data_out* This is the data to be transmitted.
buffer This is the DCI buffer number to be written.

Return Value: None

Remarks: This function loads the transmit buffer specified by the *buffer* with *data_out*.

Source File: `WriteDCI.c`

Code Example:

```
unsigned int DCI_tx0 = 0x60;
WriteDCI(DCI_tx0, 0);
```

3.13.2 Individual Macros

EnableIntDCI

Description: This macro enables the DCI interrupt.
Include: `dci.h`
Arguments: None
Remarks: This macro sets DCI Interrupt Enable bit of Interrupt Enable Control register.
Code Example: `EnableIntDCI;`

DisableIntDCI

Description: This macro disables the DCI interrupt.
Include: `dci.h`
Arguments: None
Remarks: This macro clears DCI Interrupt Enable bit of Interrupt Enable Control register.
Code Example: `DisableIntDCI;`

SetPriorityIntDCI

Description: This macro sets priority for DCI interrupt.
Include: `dci.h`
Arguments: *priority*
Remarks: This macro sets DCI Interrupt Priority bits of Interrupt Priority Control register.
Code Example: `SetPriorityIntDCI(4);`

16-Bit Language Tools Libraries

3.13.3 Example of Use

```
#define __dsPIC30F6014__
#include<p30fxxxx.h>
#include<dci.h>
/* Received data is stored from 0x1820 onwards. */
unsigned int * Receiveddata = ( unsigned int *)0x1820;
void __attribute__((__interrupt__)) _DCIInterrupt(void)
{
    IFS2bits.DCIIF = 0;
}
int main(void)
{
    /* Data to be transmitted using DCI module */
    unsigned int data16[] = {0xabcd, 0x1234, 0x1578,
        0xfff0, 0xf679};
    /* Holds configuration information */
    unsigned int DCICON1value;
    /* Holds the value of 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;
}
```

16-Bit Language Tools Libraries

3.14 SPI FUNCTIONS

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

3.14.1 Individual Functions

ConfigIntSPI1

ConfigIntSPI2

Description:	This function configures the SPI Interrupt.
Include:	<code>spi.h</code>
Prototype:	<code>void ConfigIntSPI1(unsigned int <i>config</i>);</code> <code>void ConfigIntSPI2(unsigned int <i>config</i>);</code>
Arguments:	<i>config</i> SPI interrupt priority and enable/disable information as defined below: <u>Interrupt enable/disable</u> SPI_INT_EN SPI_INT_DIS <u>Interrupt Priority</u> SPI_INT_PRI_0 SPI_INT_PRI_1 SPI_INT_PRI_2 SPI_INT_PRI_3 SPI_INT_PRI_4 SPI_INT_PRI_5 SPI_INT_PRI_6 SPI_INT_PRI_7
Return Value:	None
Remarks:	This function clears the Interrupt Flag bit, sets the interrupt priority and enables/disables the interrupt.
Source File:	<code>ConfigIntSPI1.c</code> <code>ConfigIntSPI2.c</code>
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:	<code>void CloseSPI1(void);</code> <code>void CloseSPI2(void);</code>
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:	<code>CloseSPI1.c</code> <code>CloseSPI2.c</code>
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 lower byte of SPIxBUF is returned. The SPIxBUF is read only if it contains any data as indicated by the SPISTAT<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>

16-Bit Language Tools Libraries

WriteSPI1 WriteSPI2

Description:	This function writes the data to be transmitted into the Transmit Buffer (SPIxBUF) register.
Include:	<code>spi.h</code>
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> FRAME_ENABLE_ON FRAME_ENABLE_OFF <u>Frame Sync Pulse direction control</u> FRAME_SYNC_INPUT FRAME_SYNC_OUTPUT <u>SDO Pin Control bit</u> DISABLE_SDO_PIN ENABLE_SDO_PIN <u>Word/Byte Communication mode</u> SPI_MODE16_ON SPI_MODE16_OFF <u>SPI Data Input Sample phase</u> SPI_SMP_ON SPI_SMP_OFF <u>SPI Clock Edge Select</u> SPI_CKE_ON SPI_CKE_OFF <u>SPI slave select enable</u> SLAVE_SELECT_ENABLE_ON SLAVE_SELECT_ENABLE_OFF

OpenSPI1 (Continued) OpenSPI2

SPI Clock polarity select

CLK_POL_ACTIVE_LOW
CLK_POL_ACTIVE_HIGH

SPI Mode Select bit

MASTER_ENABLE_ON
MASTER_ENABLE_OFF

Secondary Prescale select

SEC_PRESCAL_1_1
SEC_PRESCAL_2_1
SEC_PRESCAL_3_1
SEC_PRESCAL_4_1
SEC_PRESCAL_5_1
SEC_PRESCAL_6_1
SEC_PRESCAL_7_1
SEC_PRESCAL_8_1

Primary Prescale select

PRI_PRESCAL_1_1
PRI_PRESCAL_4_1
PRI_PRESCAL_16_1
PRI_PRESCAL_64_1

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

SPI Enable/Disable

SPI_ENABLE
SPI_DISABLE

SPI Idle mode operation

SPI_IDLE_CON
SPI_IDLE_STOP

Clear Receive Overflow Flag bit

SPI_RX_OVFLOW_CLR

Return Value: None

Remarks: This function initializes the SPI module and sets the Idle mode operation.

Source File: OpenSPI1.c
OpenSPI2.c

Code Example:

```
config1 = FRAME_ENABLE_OFF &  
          FRAME_SYNC_OUTPUT &  
          ENABLE_SDO_PIN &  
          SPI_MODE16_ON &  
          SPI_SMP_ON &  
          SPI_CKE_OFF &  
          SLAVE_SELECT_ENABLE_OFF &  
          CLK_POL_ACTIVE_HIGH &  
          MASTER_ENABLE_ON &  
          SEC_PRESCAL_7_1 &  
          PRI_PRESCAL_64_1;  
config2 = SPI_ENABLE &  
          SPI_IDLE_CON &  
          SPI_RX_OVFLOW_CLR OpenSPI1(config1,  
          config2);
```

16-Bit Language Tools Libraries

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><code>length</code> This is the number of data words/bytes to be transmitted.</p> <p><code>wrptr</code> 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 until 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><code>length</code> This is the length of the string to be received.</p> <p><code>rdptr</code> This is the pointer to the location where the data received have to be stored.</p> <p><code>spi_data_wait</code> This is the time-out count for which the module has to wait before return. If the time-out count is 'N', the actual time out would be about $(19 * N - 1)$ instruction cycles.</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 non-zero, it indicates that the complete string has not been received.</p>
Remarks:	None
Source File:	<code>getsSPI1.c</code> <code>getsSPI2.c</code>
Code Example:	<code>Datarem = getsSPI1(6, Rxdata_loc, 40);</code>

getcSPI1 getcSPI2

Description: This function is identical to ReadSPI1 and ReadSPI2.
Source File: #define to ReadSPI1 and ReadSPI2 in spi.h

putcSPI1 putcSPI2

Description: This function is identical to WriteSPI1 and WriteSPI2.
Source File: #define to WriteSPI1 and WriteSPI2 in spi.h

3.14.2 Individual Macros

EnableIntSPI1 EnableIntSPI2

Description: This macro enables the SPI interrupt.
Include: spi.h
Arguments: None
Remarks: This macro sets SPI Interrupt Enable bit of Interrupt Enable Control register.
Code Example: EnableIntSPI1;

DisableIntSPI1 DisableIntSPI2

Description: This macro disables the SPI interrupt.
Include: spi.h
Arguments: None
Remarks: This macro clears SPI Interrupt Enable bit of Interrupt Enable Control register.
Code Example: DisableIntSPI2;

SetPriorityIntSPI1 SetPriorityIntSPI2

Description: This macro sets priority for SPI interrupt.
Include: spi.h
Arguments: *priority*
Remarks: This macro sets SPI Interrupt Priority bits of Interrupt Priority Control register.
Code Example: SetPriorityIntSPI2(2);

16-Bit Language Tools Libraries

3.14.3 Example of Use

```
#define __dsPIC30F6014__
#include<p30fxxxx.h>
#include<spi.h>
/* Data received at SPI2 */
unsigned int datard ;
void __attribute__((__interrupt__)) _SPI1Interrupt(void)
{
    IFS0bits.SPI1IF = 0;
}
void __attribute__((__interrupt__)) _SPI2Interrupt(void)
{
    IFS1bits.SPI2IF = 0;
    SPI1STATbits.SPIROV = 0; /* Clear SPI1 receive overflow
                             flag if set */
}
int main(void)
{
    /* Holds the information about SPI configuartion */
    unsigned int SPICONValue;
    /* Holds the information about SPI Enable/Disable */
    unsigned int SPISTATValue;
    /*Timeout value during which timer1 is ON */
    int timeout;
    /* Turn off SPI modules */
    CloseSPI1();
    CloseSPI2();
    TMR1 = 0 ;
    timeout = 0;
    TRISDbits.TRISD0 = 0;
    /* Configure SPI2 interrupt */
    ConfigIntSPI2(SPI_INT_EN & SPI_INT_PRI_6);
    /* Configure SPI1 module to transmit 16 bit timer1 value
    in master mode */
    SPICONValue = FRAME_ENABLE_OFF & FRAME_SYNC_OUTPUT &
        ENABLE_SDO_PIN & SPI_MODE16_ON &
        SPI_SMP_ON & SPI_CKE_OFF &
        SLAVE_SELECT_ENABLE_OFF &
        CLK_POL_ACTIVE_HIGH &
        MASTER_ENABLE_ON &
        SEC_PRESCAL_7_1 &
        PRI_PRESCAL_64_1;
    SPISTATValue = SPI_ENABLE & SPI_IDLE_CON &
        SPI_RX_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_OVFLOW_CLR;
OpensPI2(SPICONValue, SPISTATValue );
T1CON = 0X8000;
while(timeout < 100 )
{
    timeout = timeout+2 ;
}
T1CON = 0;
WriteSPI1(TMR1);
while(SPI1STATbits.SPITBF);
while(!DataRdySPI2());
datard = ReadSPI2();
if(datard <= 600)
{
    PORTDbits.RD0 = 1;
}

/* Turn off SPI module and clear IF bit */
CloseSPI1();
CloseSPI2();
return 0;
}
```

16-Bit Language Tools Libraries

3.15 QEI FUNCTIONS

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

3.15.1 Individual Functions

CloseQEI

Description:	This function turns off the QEI module
Include:	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> QEI_INT_ENABLE QEI_INT_DISABLE <u>QEI Interrupt priority</u> 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:	<code>qei.h</code>
Prototype:	<code>void OpenQEI(unsigned int <i>config1</i>, unsigned int <i>config2</i>);</code>
Arguments:	<p><i>config1</i> This contains the parameters to be configured in the QEIxCON register as defined below:</p> <ul style="list-style-type: none"><u>Position Counter Direction Selection Control</u> QEI_DIR_SEL_QEB QEI_DIR_SEL_CNTRL<u>Timer Clock Source Select bit</u> QEI_EXT_CLK QEI_INT_CLK<u>Position Counter Reset Enable</u> QEI_INDEX_RESET_ENABLE QEI_INDEX_RESET_DISABLE<u>Timer Input Clock Prescale Select bits</u> QEI_CLK_PRESCALE_1 QEI_CLK_PRESCALE_8 QEI_CLK_PRESCALE_64 QEI_CLK_PRESCALE_256<u>Timer Gated Time Accumulation Enable</u> QEI_GATED_ACC_ENABLE QEI_GATED_ACC_DISABLE<u>Position Counter Direction State Output Enable</u> QEI_LOGIC_CONTROL_IO QEI_NORMAL_IO<u>Phase A and Phase B Input Swap Select bit</u> QEI_INPUTS_SWAP QEI_INPUTS_NOSWAP<u>QEI Mode of operation select</u> QEI_MODE_x4_MATCH QEI_MODE_x4_PULSE QEI_MODE_x2_MATCH QEI_MODE_x2_PULSE QEI_MODE_TIMER QEI_MODE_OFF<u>Position Counter Direction Status</u> QEI_UP_COUNT QEI_DOWN_COUNT<u>Idle Mode Operation</u> QEI_IDLE_STOP QEI_IDLE_CON <p><i>config2</i> This contains the parameters to be configured in the DFLTxCON register.</p> <p>In 4x Quadrature Count Mode:</p> <ul style="list-style-type: none"><u>Required State of Phase A input signal for match on index pulse</u> MATCH_INDEX_PHASEA_HIGH MATCH_INDEX_PHASEA_LOW<u>Required State of Phase B input signal for match on index pulse</u> MATCH_INDEX_PHASEB_HIGH MATCH_INDEX_PHASEB_LOW

OpenQEI (Continued)

In 2x Quadrature Count Mode:
Phase input signal for index state match
MATCH_INDEX_INPUT_PHASEA
MATCH_INDEX_INPUT_PHASEB
Phase input signal state for match on index pulse
MATCH_INDEX_INPUT_HIGH
MATCH_INDEX_INPUT_LOW
Enable/Disable interrupt due to position count event
POS_CNT_ERR_INT_ENABLE
POS_CNT_ERR_INT_DISABLE
QEA/QEB Digital Filter Clock Divide Select bits
QEI_QE_CLK_DIVIDE_1_1
QEI_QE_CLK_DIVIDE_1_2
QEI_QE_CLK_DIVIDE_1_4
QEI_QE_CLK_DIVIDE_1_16
QEI_QE_CLK_DIVIDE_1_32
QEI_QE_CLK_DIVIDE_1_64
QEI_QE_CLK_DIVIDE_1_128
QEI_QE_CLK_DIVIDE_1_256
QEA/QEB Digital Filter Output Enable
QEI_QE_OUT_ENABLE
QEI_QE_OUT_DISABLE

Return Value None
Remarks: This function configures the QEICON and DFLTCON registers of QEI module.
This function also clears the QEICON<CNTERR> bit.
Source File: OpenQEI.c
Code Example:

```
OpenQEI(QEI_DIR_SEL_QEB & QEI_INT_CLK &
QEI_INDEX_RESET_ENABLE &
QEI_CLK_PRESCALE_1 & QEI_NORMAL_IO &
QEI_MODE_TIMER & QEI_UP_COUNT, 0);
```

ReadQEI

Description: This function read the position count value from the POSCNT register.
Include: qei.h
Prototype: unsigned int ReadQEI(void);
Arguments: None
Remarks: None
Return Value This functions returns the contents of the POSCNT register.
Source File: ReadQEI.c
Code Example:

```
unsigned int pos_count;
pos_count = ReadQEI();
```

WriteQEI

Description:	This function sets the maximum count value for QEI.
Include:	<code>qei.h</code>
Prototype:	<code>void WriteQEI(unsigned int <i>position</i>);</code>
Arguments:	<i>position</i> This is the value to be stored into the MAXCNT register.
Return Value	None
Remarks:	None
Source File:	<code>WriteQEI.c</code>
Code Example:	<pre>unsigned int position = 0x3FFF; WriteQEI(position);</pre>

3.15.2 Individual Macros

EnableIntQEI

Description:	This macro enables the QEI interrupt.
Include:	<code>qei.h</code>
Arguments:	None
Remarks:	This macro sets QEI Interrupt Enable bit of Interrupt Enable Control register.
Code Example:	<code>EnableIntQEI;</code>

DisableIntQEI

Description:	This macro disables the QEI interrupt.
Include:	<code>qei.h</code>
Arguments:	None
Remarks:	This macro clears QEI Interrupt Enable bit of Interrupt Enable Control register.
Code Example:	<code>DisableIntQEI;</code>

SetPriorityIntQEI

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

16-Bit Language Tools Libraries

3.15.3 Example of Use

```
#define __dsPIC30F6010__
#include <p30fxxxx.h>
#include <gei.h>
unsigned int pos_value;

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

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

    /* Enable QEI Interrupt and Priority to "1" */
    ConfigIntQEI(QEI_INT_PRI_1 & QEI_INT_ENABLE);

    POSCNT = 0;
    MAXCNT = 0xFFFF;
    OpenQEI(QEI_INT_CLK & QEI_INDEX_RESET_ENABLE &
            QEI_CLK_PRESCALE_256 &
            QEI_GATED_ACC_DISABLE & QEI_INPUTS_NOSWAP &
            QEI_MODE_TIMER & QEI_DIR_SEL_CNTRL &
            QEI_IDLE_CON, 0);
    QEICONbits.UPDN = 1;
    while(1)
    {
        pos_value = ReadQEI();
        if(pos_value >= 0x7FFF)
        {
            PORTDbits.RD1 = 0; /* turn on LED on RD1 */
        }
    }
    CloseQEI();
}
```

3.16 PWM FUNCTIONS

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

3.16.1 Individual Functions

CloseMCPWM

Description:	This function turns off the Motor Control PWM module.
Include:	<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:	<i>config</i> PWM interrupt priority and enable/disable information as defined below: <u>PWM Interrupt enable/disable</u> <code>PWM_INT_EN</code> <code>PWM_INT_DIS</code> <u>PWM Interrupt priority</u> <code>PWM_INT_PR0</code> <code>PWM_INT_PR1</code> <code>PWM_INT_PR2</code> <code>PWM_INT_PR3</code> <code>PWM_INT_PR4</code> <code>PWM_INT_PR5</code> <code>PWM_INT_PR6</code> <code>PWM_INT_PR7</code> <u>Fault A Interrupt enable/disable</u> <code>PWM_FLTA_EN_INT</code> <code>PWM_FLTA_DIS_INT</code> <u>Fault A Interrupt priority</u> <code>PWM_FLTA_INT_PR0</code> <code>PWM_FLTA_INT_PR1</code> <code>PWM_FLTA_INT_PR2</code> <code>PWM_FLTA_INT_PR3</code> <code>PWM_FLTA_INT_PR4</code> <code>PWM_FLTA_INT_PR5</code> <code>PWM_FLTA_INT_PR6</code> <code>PWM_FLTA_INT_PR7</code> <u>Fault B Interrupt enable/disable</u> <code>PWM_FLTB_EN_INT</code> <code>PWM_FLTB_DIS_INT</code>

16-Bit Language Tools Libraries

ConfigIntMCPWM (Continued)

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:	<pre>ConfigIntMCPWM(PWM_INT_EN & PWM_INT_PR5 & PWM_FLTA_EN_INT & PWM_FLTA_INT_PR6 & PWM_FLTB_EN_INT & PWM_FLTB_INT_PR7);</pre>

OpenMCPWM

Description:	This function configure the motor control PWM module.
Include:	pwm.h
Prototype:	<pre>void OpenMCPWM(unsigned int period, unsigned int sptime, unsigned int config1, unsigned int config2, unsigned int config3);</pre>
Arguments:	<p><i>period</i> This contains the PWM timebase period value to be stored in PTPER register.</p> <p><i>sptime</i> This contains the special event compare value to be stored in SEVTCMP register.</p> <p><i>config1</i> This contains the parameters to be configured in the PTCON register as defined below:</p> <p><u>PWM module enable/disable</u> PWM_EN PWM_DIS</p> <p><u>Idle mode enable/disable</u> PWM_IDLE_STOP PWM_IDLE_CON</p> <p><u>Output post scaler select</u> PWM_OP_SCALE1 PWM_OP_SCALE2 PWM_OP_SCALE15 PWM_OP_SCALE16</p> <p><u>Input prescaler select</u> PWM_IPCLK_SCALE1 PWM_IPCLK_SCALE4 PWM_IPCLK_SCALE16 PWM_IPCLK_SCALE64</p>

OpenMCPWM (Continued)

PWM mode of operation

PWM_MOD_FREE
PWM_MOD_SING
PWM_MOD_UPDN
PWM_MOD_DBL

config2 This contains the parameters to be configured in the PWMCON1 register as defined below:

PWM I/O pin pair

PWM_MOD4_COMP
PWM_MOD3_COMP
PWM_MOD2_COMP
PWM_MOD1_COMP
PWM_MOD4_IND
PWM_MOD3_IND
PWM_MOD2_IND
PWM_MOD1_IND

PWM H/L I/O enable/disable select

PWM_PEN4H
PWM_PDIS4H
PWM_PEN3H
PWM_PDIS3H
PWM_PEN2H
PWM_PDIS2H
PWM_PEN1H
PWM_PDIS1H
PWM_PEN4L
PWM_PDIS4L
PWM_PEN3L
PWM_PDIS3L
PWM_PEN2L
PWM_PDIS2L
PWM_PEN1L
PWM_PDIS1L

Bit defines related to PWM4 is available only for certain devices and the suitable data sheet should be referred to.

config3 This contains the parameters to be configured in the PWMCON2 register as defined below:

Special event post scaler

PWM_SEVOPS1
PWM_SEVOPS2
.....
PWM_SEVOPS15
PWM_SEVOPS16

Output Override synchronization select

PWM_OSYNC_PWM
PWM_OSYNC_Tcy

PWM update enable/disable

PWM_UDIS
PWM_UEN

Return Value None

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

16-Bit Language Tools Libraries

OpenMCPWM (Continued)

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 Duty Cycle register.

updatedisable This is the value to be loaded into the 'Update Disable' bit of the PWMCON2 register.

Return Value None

Remarks: None

Source File: SetDCMCPWM.c

Code Example:

```
dutycyclereg = 1;  
dutycycle = 0xFFFF;  
updatedisable = 0;  
SetDCMCPWM(dutycyclereg, dutycycle, updatedisable);
```

16-Bit Language Tools Libraries

SetMCPWMDeadTimeAssignment

Description:	This function configures the assignment of dead-time units to PWM output pairs.
Include:	<code>pwm.h</code>
Prototype:	<code>void SetMCPWMDeadTimeAssignment (unsigned int <i>config</i>);</code>
Arguments:	<i>config</i> This contains the parameters to be configured in the DTCON2 register as defined below: <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. <u>Dead-Time Select bits for PWM3 signal</u> PWM_DTS3A_UA PWM_DTS3A_UB PWM_DTS3I_UA PWM_DTS3I_UB <u>Dead-Time Select bits for PWM2 signal</u> PWM_DTS2A_UA PWM_DTS2A_UB PWM_DTS2I_UA PWM_DTS2I_UB <u>Dead-Time Select bits for PWM1 signal</u> PWM_DTS1A_UA PWM_DTS1A_UB PWM_DTS1I_UA PWM_DTS1I_UB
Return Value	None
Remarks:	None
Source File:	<code>SetMCPWMDeadTimeAssignment.c</code>
Code Example:	<code>SetMCPWMDeadTimeAssignment (PWM_DTS3A_UA & PWM_DTS2I_UA & PWM_DTS1I_UA);</code>

SetMCPWMDeadTimeGeneration

Description:	This function configures dead-time values and clock prescalers.
Include:	pwm.h
Prototype:	<pre>void SetMCPWMDeadTimeGeneration(unsigned int <i>config</i>);</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> <p><u>Dead-Time Unit A Prescale Select constants</u> PWM_DTA0 PWM_DTA1 PWM_DTA2 PWM_DTA62 PWM_DTA63</p> <p><u>Dead-Time Unit B Prescale Select constants</u> PWM_DTB0 PWM_DTB1 PWM_DTB2 PWM_DTB62 PWM_DTB63</p> <p><u>Dead-Time Unit A Prescale Select bits</u> PWM_DTAPS8 PWM_DTAPS4 PWM_DTAPS2 PWM_DTAPS1</p>
Return Value	None
Remarks:	None
Source File:	SetMCPWMDeadTimeGeneration.c
Code Example:	<pre>SetMCPWMDeadTimeGeneration(PWM_DTBPS16 & PWM_DT54 & PWM_DTAPS8);</pre>

16-Bit Language Tools Libraries

SetMCPWMFaultA

Description: This function configures Fault A Override bits, Fault A Mode bit and Fault Input A Enable bits of PWM.

Include: pwm.h

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

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

Fault Input A PWM Override Value bits

PWM_OVA4H_ACTIVE

PWM_OVA3H_ACTIVE

PWM_OVA2H_ACTIVE

PWM_OVA1H_ACTIVE

PWM_OVA4L_ACTIVE

PWM_OVA3L_ACTIVE

PWM_OVA2L_ACTIVE

PWM_OVA1L_ACTIVE

PWM_OVA4H_INACTIVE

PWM_OVA3H_INACTIVE

PWM_OVA2H_INACTIVE

PWM_OVA1H_INACTIVE

PWM_OVA4L_INACTIVE

PWM_OVA3L_INACTIVE

PWM_OVA2L_INACTIVE

PWM_OVA1L_INACTIVE

Bit defines related to PWM4 is available only for certain devices and the suitable data sheet should be referred to.

Fault A Mode bit

PWM_FLTA_MODE_CYCLE

PWM_FLTA_MODE_LATCH

Fault Input A Enable bits.

PWM_FLTA4_EN

PWM_FLTA4_DIS

PWM_FLTA3_EN

PWM_FLTA3_DIS

PWM_FLTA2_EN

PWM_FLTA2_DIS

PWM_FLTA1_EN

PWM_FLTA1_DIS

Bit defines related to PWM4 is available only for certain devices and the suitable data sheet should be referred to.

Return Value None

Remarks: None

Source File: SetMCPWMFaultA.c

Code Example: SetMCPWMFaultA(PWM_OVA3L_INACTIVE &
PWM_FLTA_MODE_LATCH &
PWM_FLTA1_DIS);

SetMCPWMFaultB

Description: This function configures Fault B Override bits, Fault B Mode bit and Fault Input B Enable bits of PWM.

Include: `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

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);`

3.16.2 Individual Macros

EnableIntMCPWM

Description: This macro enables the PWM interrupt.
Include: `pwm.h`
Arguments: None
Remarks: This macro sets PWM Interrupt Enable bit of Interrupt Enable Control register.
Code Example: `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`
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*
Remarks: This macro sets FLTB Interrupt Priority bits of Interrupt Priority Control register.
Code Example: `SetPriorityIntFLTB(1);`

16-Bit Language Tools Libraries

3.16.3 Example of Use

```
#define __dsPIC30F6010__
#include <p30fxxxx.h>
#include<pwm.h>
void __attribute__((__interrupt__)) _PWMInterrupt(void)
{
    IFS2bits.PWMIF = 0;
}
int main()
{
    /* Holds the PWM interrupt configuration value*/
    unsigned int config;
    /* Holds the value to be loaded into dutycycle register */
    unsigned int period;
    /* Holds the value to be loaded into special event compare register */
    unsigned int sptime;
    /* Holds PWM configuration value */
    unsigned int config1;
    /* Holds the value be loaded into PWMCON1 register */
    unsigned int config2;
    /* Holds the value to configure the special event trigger
    postscale and dutycycle */
    unsigned int config3;
    /* The value of 'dutycyclereg' determines the duty cycle
    register(PDCx) to be written */
    unsigned int dutycyclereg;
    unsigned int dutycycle;
    unsigned char updatedisable;

    /* Configure pwm interrupt enable/disable and set interrupt
    priorities */
    config = (PWM_INT_EN & PWM_FLTA_DIS_INT & PWM_INT_PR1
        & PWM_FLTA_INT_PRO
        & PWM_FLTB_DIS_INT & PWM_FLTB_INT_PRO);
    ConfigIntMCPWM( config );
    /* Configure PWM to generate square wave of 50% duty cycle */
    dutycyclereg = 1;
    dutycycle = 0x3FFF;
    updatedisable = 0;

    SetDCMCPWM(dutycyclereg,dutycycle,updatedisable);
    period = 0x7fff;
    sptime = 0x0;
    config1 = (PWM_EN & PWM_PTSIDL_DIS & PWM_OP_SCALE16
        & PWM_IPCLK_SCALE16 &
        PWM_MOD_UPDN);
    config2 = (PWM_MOD1_COMP & PWM_PDIS4H & PWM_PDIS3H &
        PWM_PDIS2H & PWM_PEN1H & PWM_PDIS4L &
        PWM_PDIS3L & PWM_PDIS2L & PWM_PEN1L);
    config3 = (PWM_SEVOPS1 & PWM_OSYNC_PWM & PWM_UEN);
    OpenMCPWM(period,sptime,config1,config2,config3);
    while(1);
}
```

3.17 I²C™ FUNCTIONS

This section contains a list of individual functions for I²C module and an example of use of the functions. Functions may be implemented as macros.

3.17.1 Individual Functions

CloseI2C

Description:	This function turns off the I ² C module
Include:	<code>i2c.h</code>
Prototype:	<code>void CloseI2C(void);</code>
Arguments:	None
Return Value	None
Remarks:	This function disables the I ² C 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 configures the I ² C Interrupt.
Include:	<code>i2c.h</code>
Prototype:	<code>void ConfigIntI2C(unsigned int config);</code>
Arguments:	<code>config</code> I ² C 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>
Return Value	None

16-Bit Language Tools Libraries

ConfigIntI2C (Continued)

Remarks: This function clears the Interrupt Flag bits, sets the interrupt priorities of master and slave and enables/disables the interrupt.

Source File: ConfigIntI2C.c

Code Example: ConfigIntI2C(MI2C_INT_ON & MI2C_INT_PRI_3
& SI2C_INT_ON & SI2C_INT_PRI_5);

AckI2C

Description: Generates 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 I ² C bus.						
Include:	<code>i2c.h</code>						
Prototype:	<code>unsigned int MastergetsI2C(unsigned int length, unsigned char *rdptr, unsigned int i2c_data_wait);</code>						
Arguments:	<table><tr><td><code>length</code></td><td>Number of bytes to read from I²C device.</td></tr><tr><td><code>rdptr</code></td><td>Character type pointer to RAM for storage of data read from I²C device</td></tr><tr><td><code>i2c_data_wait</code></td><td>This is the time-out count for which the module has to wait before return. If the time-out count is 'N', the actual time out would be about (20 * N - 1) instruction cycles.</td></tr></table>	<code>length</code>	Number of bytes to read from I ² C device.	<code>rdptr</code>	Character type pointer to RAM for storage of data read from I ² C device	<code>i2c_data_wait</code>	This is the time-out count for which the module has to wait before return. If the time-out count is 'N', the actual time out would be about (20 * N - 1) instruction cycles.
<code>length</code>	Number of bytes to read from I ² C device.						
<code>rdptr</code>	Character type pointer to RAM for storage of data read from I ² C device						
<code>i2c_data_wait</code>	This is the time-out count for which the module has to wait before return. If the time-out count is 'N', the actual time out would be about (20 * N - 1) instruction cycles.						
Return Value	This function returns '0' if all bytes have been sent or number of bytes read from I ² C bus if its not able to read the data with in the specified <code>i2c_data_wait</code> time out value						
Remarks:	This routine reads a predefined data string from the I ² C 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 I ² C bus.		
Include:	<code>i2c.h</code>		
Prototype:	<code>unsigned int MasterputsI2C(unsigned char *wrptr);</code>		
Arguments:	<table><tr><td><code>wrptr</code></td><td>Character type pointer to data objects in RAM. The data objects are written to the I²C device.</td></tr></table>	<code>wrptr</code>	Character type pointer to data objects in RAM. The data objects are written to the I ² C device.
<code>wrptr</code>	Character type pointer to data objects in RAM. The data objects are written to the I ² C device.		
Return Value	This function returns -3 if a write collision occurred. This function returns '0' if the null character was reached in data string.		
Remarks:	This function writes a string to the I ² C 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>		

16-Bit Language Tools Libraries

MasterReadI2C

Description: This function is used to read a single byte from I²C bus

Include: `i2c.h`

Prototype: `unsigned char MasterReadI2C(void);`

Arguments: None

Return Value The return value is the data byte read from the I²C bus.

Remarks: This function reads in a single byte from the I²C bus.
This function performs the same function as `MastergetcI2C`.

Source File: `MasterReadI2C.c`

Code Example:

```
unsigned char value;
value = MasterReadI2C();
```

MasterWriteI2C

Description: This function is used to write out a single data byte to the I²C device.

Include: `i2c.h`

Prototype: `unsigned char MasterWriteI2C(unsigned char data_out);`

Arguments: `data_out` A single data byte to be written to the I²C bus device.

Return Value This function returns -1 if there was a write collision else it returns a 0.

Remarks: This function writes out a single data byte to the I²C bus device. This function performs the same function as `MasterputcI2C`.

Source File: `MasterWriteI2C.c`

Code Example: `MasterWriteI2C('a');`

NotAckI2C

Description: Generates I²C bus Not Acknowledge condition.

Include: `i2c.h`

Prototype: `void NotAckI2C(void);`

Arguments: None

Return Value None

Remarks: This function generates an I²C bus *Not Acknowledge* condition.

Source File: `NotAckI2C.c`

Code Example: `NotAckI2C();`

OpenI2C

Description:	Configures the I ² C module.
Include:	<code>i2c.h</code>
Prototype:	<code>void OpenI2C(unsigned int <i>config1</i>, unsigned int <i>config2</i>);</code>
Arguments:	<code><i>config1</i></code> This contains the parameter to configure the I2CCON register <u>I²C Enable bit</u> I2C_ON I2C_OFF <u>I²C Stop in Idle Mode bit</u> I2C_IDLE_STOP I2C_IDLE_CON <u>SCL Release Control bit</u> I2C_CLK_REL I2C_CLK_HLD <u>Intelligent Peripheral Management Interface Enable bit</u> I2C_IPMI_EN I2C_IPMI_DIS <u>10-bit Slave Address bit</u> I2C_10BIT_ADD I2C_7BIT_ADD <u>Disable Slew Rate Control bit</u> I2C_SLW_DIS I2C_SLW_EN <u>SMBus Input Level bits</u> I2C_SM_EN I2C_SM_DIS <u>General Call Enable bit</u> I2C_GCALL_EN I2C_GCALL_DIS <u>SCL Clock Stretch Enable bit</u> I2C_STR_EN I2C_STR_DIS <u>Acknowledge Data bit</u> I2C_ACK I2C_NACK <u>Acknowledge Sequence Enable bit</u> I2C_ACK_EN I2C_ACK_DIS <u>Receive Enable bit</u> I2C_RCV_EN I2C_RCV_DIS <u>Stop Condition Enable bit</u> I2C_STOP_EN I2C_STOP_DIS <u>Repeated Start Condition Enable bit</u> I2C_RESTART_EN I2C_RESTART_DIS <u>Start Condition Enable bit</u> I2C_START_EN I2C_START_DIS <code><i>config2</i></code> computed value for the baud rate generator

16-Bit Language Tools Libraries

OpenI2C (Continued)

Return Value	None
Remarks:	This function configures the I ² C Control register and I ² C Baud Rate Generator register.
Source File:	OpenI2C.c
Code Example:	OpenI2C();

RestartI2C

Description:	Generates I ² C Bus Restart condition.
Include:	i2c.h
Prototype:	void RestartI2C(void);
Arguments:	None
Return Value	None
Remarks:	This function generates an I ² C Bus Restart condition.
Source File:	RestartI2C.c
Code Example:	RestartI2C();

SlavegetsI2C

Description:	This function reads pre-determined data string length from the I ² C bus.
Include:	i2c.h
Prototype:	unsigned int SlavegetsI2C(unsigned char *rdptr, unsigned int i2c_data_wait);
Arguments:	<i>rdptr</i> Character type pointer to RAM for storage of data read from I ² C device. <i>i2c_data_wait</i> This is the time-out count for which the module has to wait before return. If the time-out count is 'N', the actual time out would be about (20*N - 1) instruction cycles.
Return Value	Returns the number of bytes received from the I ² C bus.
Remarks:	This routine reads a predefined data string from the I ² C bus.
Source File:	SlavegetsI2C.c
Code Example:	unsigned char string[12]; unsigned char *rdptr; rdptr = string; i2c_data_out = 0x11; SlavegetsI2C(rdptr, i2c_data_wait);

SlaveputsI2C

Description:	This function is used to write out a data string to the I ² C bus.
Include:	<code>i2c.h</code>
Prototype:	<code>unsigned int SlaveputsI2C(unsigned char *wrptr);</code>
Arguments:	<code>wrptr</code> Character type pointer to data objects in RAM. The data objects are written to the I ² C 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 I ² C bus until a null character is reached.
Source File:	<code>SlaveputsI2C.c</code>
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 I ² C bus.
Include:	<code>i2c.h</code>
Prototype:	<code>unsigned char SlaveReadI2C(void);</code>
Arguments:	None
Return Value	The return value is the data byte read from the I ² C bus.
Remarks:	This function reads in a single byte from the I ² C 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 I ² C 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 I ² C bus device.
Return Value	None
Remarks:	This function writes out a single data byte to the I ² C bus device. This function performs the same function as <code>SlaveputcI2C</code> .
Source File:	<code>SlaveWriteI2C.c</code>
Code Example:	<pre>SlaveWriteI2C('a');</pre>

16-Bit Language Tools Libraries

StartI2C

Description: Generates I²C Bus Start condition.
Include: `i2c.h`
Prototype: `void StartI2C(void);`
Arguments: None
Return Value None
Remarks: This function generates a I²C Bus Start condition.
Source File: `StartI2C.c`
Code Example: `StartI2C();`

StopI2C

Description: Generates I²C Bus Stop condition.
Include: `i2c.h`
Prototype: `void StopI2C(void);`
Arguments: None
Return Value None
Remarks: This function generates a I²C Bus Stop condition.
Source File: `StopI2C.c`
Code Example: `StopI2C();`

3.17.2 Individual Macros

EnableIntMI2C

Description: This macro enables the master I²C interrupt.
Include: `i2c.h`
Arguments: None
Remarks: This macro sets Master I²C Enable bit of Interrupt Enable Control register.
Code Example: `EnableIntMI2C;`

DisableIntMI2C

Description: This macro disables the master I²C interrupt.
Include: `i2c.h`
Arguments: None
Remarks: This macro clears Master I²C Interrupt Enable bit of Interrupt Enable Control register.
Code Example: `DisableIntMI2C;`

SetPriorityIntMI2C

Description: This macro sets priority for master I²C interrupt.
Include: `i2c.h`
Arguments: `priority`
Remarks: This macro sets Master I²C Interrupt Priority bits of Interrupt Priority Control register.
Code Example: `SetPriorityIntMI2C(1);`

EnableIntSI2C

Description: This macro enables the slave I²C interrupt.
Include: `i2c.h`
Arguments: None
Remarks: This macro sets Slave I²C Enable bit of Interrupt Enable Control register.
Code Example: `EnableIntSI2C;`

DisableIntSI2C

Description: This macro disables the slave I²C interrupt.
Include: `i2c.h`
Arguments: None
Remarks: This macro clears Slave I²C Interrupt Enable bit of Interrupt Enable Control register.
Code Example: `DisableIntSI2C;`

SetPriorityIntSI2C

Description: This macro sets priority for master I²C interrupt.
Include: `i2c.h`
Arguments: `priority`
Remarks: This macro sets Master I²C Interrupt Priority bits of Interrupt Priority Control register.
Code Example: `SetPriorityIntSI2C(4);`

16-Bit Language Tools Libraries

3.17.3 Example of Use

```
#define __dsPIC30F6014__
#include <p30fxxxx.h>
#include<i2c.h>

void main(void )
{
    unsigned int config2, config1;
    unsigned char *wrptr;
    unsigned char tx_data[] =
        {'M','I','C','R','O','C','H','I','P','\0'};
    wrptr = tx_data;
    /* Baud rate is set for 100 Khz */
    config2 = 0x11;
    /* Configure I2C for 7 bit address mode */
    config1 = (I2C_ON & I2C_IDLE_CON & I2C_CLK_HLD
        & I2C_IPMI_DIS & I2C_7BIT_ADD
        & I2C_SLW_DIS & I2C_SM_DIS &
        I2C_GCALL_DIS & I2C_STR_DIS &
        I2C_NACK & I2C_ACK_DIS & I2C_RCV_DIS &
        I2C_STOP_DIS & I2C_RESTART_DIS
        & I2C_START_DIS);
    OpenI2C(config1,config2);
    IdleI2C();
    StartI2C();
    /* Wait till Start sequence is completed */
    while(I2CCONbits.SEN );
    /* Write Slave address and set master for transmission */
    MasterWriteI2C(0xE);
    /* Wait till address is transmitted */
    while(I2CSTATbits.TBF);
    while(I2CSTATbits.ACKSTAT);
    /* Transmit string of data */
    MasterputsI2C(wrptr);
    StopI2C();
    /* Wait till stop sequence is completed */
    while(I2CCONbits.PEN);
    CloseI2C();
}
```

Chapter 4. Standard C Libraries with Math Functions

4.1 INTRODUCTION

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

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

4.1.1 Assembly Code Applications

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

4.1.2 C Code Applications

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

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

In addition, there is a file, `ResourceGraphs.pdf`, which contains diagrams of resources used by each function, located in `lib`.

4.1.3 Chapter Organization

This chapter is organized as follows:

- Using the Standard C Libraries

libc-omf.a

- `<assert.h>` diagnostics
- `<ctype.h>` character handling
- `<errno.h>` errors
- `<float.h>` floating-point characteristics
- `<limits.h>` implementation-defined limits
- `<locale.h>` localization
- `<setjmp.h>` non-local jumps
- `<signal.h>` signal handling
- `<stdarg.h>` variable argument lists
- `<stddef.h>` common definitions
- `<stdio.h>` input and output
- `<stdlib.h>` utility functions
- `<string.h>` string functions
- `<time.h>` date and time functions

libm-omf.a

- `<math.h>` mathematical functions

16-Bit Language Tools Libraries

libpic30-omf.a

- pic30-libs

4.2 USING THE STANDARD C LIBRARIES

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

4.2.1 Header Files

All standard C library entities are declared or defined in one or more standard headers (See list in **Section 4.1.3 “Chapter Organization”**.) To make use of a library entity in a program, write an include directive that names the relevant standard header.

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

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

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

A standard header never includes another standard header.

4.2.2 Library Files

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

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

A typical C application will require three library files: `libc-omf.a`, `libm-omf.a`, and `libpic30-omf.a`. (See **Section 1.2 “OMF-Specific Libraries/Start-up Modules”** for more on OMF-specific libraries.) These libraries will be included automatically if linking is performed using the MPLAB C30 compiler.

Note: Some standard library functions require a heap. These include the standard I/O functions that open files and the memory allocation functions. See the “*MPLAB ASM30, MPLAB LINK30 and Utilities User’s Guide*” (DS51317) and “*MPLAB C30 C Compiler User’s Guide*” (DS51284) for more information on the heap.

4.3 <ASSERT.H> DIAGNOSTICS

The header file `assert.h` consists of a single macro that is useful for debugging logic errors in programs. By using the `assert` statement in critical locations where certain conditions should be true, the logic of the program may be tested.

Assertion testing may be turned off without removing the code by defining `NDEBUG` before including `<assert.h>`. If the macro `NDEBUG` is defined, `assert()` is ignored and no code is generated.

assert

Description: If the expression is false, an assertion message is printed to `stderr` and the program is aborted.

Include: `<assert.h>`

Prototype: `void assert(int expression);`

Argument: `expression` The expression to test.

Remarks: The expression evaluates to zero or non-zero. If zero, the assertion fails, and a message is printed to `stderr`. The message includes the source file name (`__FILE__`), the source line number (`__LINE__`), the expression being evaluated and the message. The macro then calls the function `abort()`. If the macro `_VERBOSE_DEBUGGING` is defined, a message will be printed to `stderr` each time `assert()` is called.

Example:

```
#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
```

16-Bit Language Tools Libraries

4.4 <CTYPE.H> CHARACTER HANDLING

The header file `ctype.h` consists of functions that are useful for classifying and mapping characters. Characters are interpreted according to the Standard C locale.

isalnum

Description: Test for an alphanumeric character.
Include: `<ctype.h>`
Prototype: `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.
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.

Standard C Libraries with Math Functions

isalpha (Continued)

Example:

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

int main(void)
{
    int ch;

    ch = 'B';
    if (isalpha(ch))
        printf("B is alphabetic\n");
    else
        printf("B is NOT alphabetic\n");

    ch = '#';
    if (isalpha(ch))
        printf("# is alphabetic\n");
    else
        printf("# is NOT alphabetic\n");
}
```

Output:

```
B is alphabetic
# is NOT alphabetic
```

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:

```
#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");
}
```

Output:

```
B is NOT a control character
A tab is a control character
```

16-Bit Language Tools Libraries

isdigit

Description: Test for a decimal digit.

Include: <ctype.h>

Prototype: int isdigit(int c);

Argument: c character to test.

Return Value: Returns a non-zero integer value if the character is a digit; otherwise, returns zero.

Remarks: A character is considered to be a digit character if it is in the range of '0'-'9'.

Example: #include <ctype.h> /* for isdigit */
#include <stdio.h> /* for printf */

```
int main(void)
{
    int ch;

    ch = '3';
    if (isdigit(ch))
        printf("3 is a digit\n");
    else
        printf("3 is NOT a digit\n");

    ch = '#';
    if (isdigit(ch))
        printf("# is a digit\n");
    else
        printf("# is NOT a digit\n");
}
```

Output:

```
3 is a digit
# is NOT a digit
```

isgraph

Description: Test for a graphical character.

Include: <ctype.h>

Prototype: int isgraph (int c);

Argument: c character to test

Return Value: Returns a non-zero integer value if the character is a graphical character; otherwise, returns zero.

Remarks: A character is considered to be a graphical character if it is any printable character except a space.

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

```
int main(void)
{
    int ch;
```

Standard C Libraries with Math Functions

isgraph (Continued)

```
ch = '3';
if (isgraph(ch))
    printf("3 is a graphical character\n");
else
    printf("3 is NOT a graphical character\n");

ch = '#';
if (isgraph(ch))
    printf("# is a graphical character\n");
else
    printf("# is NOT a graphical character\n");

ch = ' ';
if (isgraph(ch))
    printf("a space is a graphical character\n");
else
    printf("a space is NOT a graphical character\n");
}
```

Output:

```
3 is a graphical character
# is a graphical character
a space is NOT a graphical character
```

islower

Description: Test for a lower case alphabetic character.

Include: <ctype.h>

Prototype: int islower (int c);

Argument: c character to test

Return Value: Returns a non-zero integer value if the character is a lower case alphabetic character; otherwise, returns zero.

Remarks: A character is considered to be a lower case alphabetic character if it is in the range of 'a'-'z'.

Example:

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

int main(void)
{
    int ch;

    ch = 'B';
    if (islower(ch))
        printf("B is lower case\n");
    else
        printf("B is NOT lower case\n");

    ch = 'b';
    if (islower(ch))
        printf("b is lower case\n");
    else
        printf("b is NOT lower case\n");
}
```

Output:

```
B is NOT lower case
b is lower case
```

16-Bit Language Tools Libraries

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:	<pre>#include <ctype.h> /* for isprint */ #include <stdio.h> /* for printf */ int main(void) { int ch; ch = '&'; if (isprint(ch)) printf("& is a printable character\n"); else printf("& is NOT a printable character\n"); ch = '\t'; if (isprint(ch)) printf("a tab is a printable character\n"); else printf("a tab is NOT a printable character\n"); } Output: & is a printable character a tab is NOT a printable character</pre>

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: !"#\$%&'();<=>?@[\\]*+,-./:^_{ }~

Standard C Libraries with Math Functions

ispunct (Continued)

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').

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
```

16-Bit Language Tools Libraries

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:	<pre>#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"); }</pre> <p>Output: B is upper case b is NOT upper case</p>

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.

Standard C Libraries with Math Functions

isxdigit (Continued)

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.

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
```

16-Bit Language Tools Libraries

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));
}
```

Output:

```
b changes to upper case B
B remains upper case B
@ has no upper case, so @ is returned
```

4.5 <ERRNO.H> ERRORS

The header file `errno.h` consists of macros that provide error codes that are reported by certain library functions (see individual functions). The variable `errno` may return any value greater than zero. To test if a library function encounters an error, the program should store the value zero in `errno` immediately before calling the library function. The value should be checked before another function call could change the value. At program start-up, `errno` is zero. Library functions will never set `errno` to zero.

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 start-up, `errno` is set to zero. `Errno` should be reset to zero prior to calling a function that sets it.

16-Bit Language Tools Libraries

4.6 <FLOAT.H> FLOATING-POINT CHARACTERISTICS

The header file `float.h` consists of macros that specify various properties of floating-point types. These properties include number of significant figures, size limits, and what rounding mode is used.

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

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.

Standard C Libraries with Math Functions

DBL_MAX_10_EXP

Description: Maximum integer value for a double precision floating-point exponent in base 10

Include: `<float.h>`

Value: 38 by default, 308 if the switch `-fno-short-double` is used

Remarks: By default, a double type is the same size as a float type (32-bit representation). The `-fno-short-double` switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.

DBL_MAX_EXP

Description: Maximum integer value for a double precision floating-point exponent in base `FLT_RADIX`

Include: `<float.h>`

Value: 128 by default, 1024 if the switch `-fno-short-double` is used

Remarks: By default, a double type is the same size as a float type (32-bit representation). The `-fno-short-double` switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.

DBL_MIN

Description: Minimum double precision floating-point value

Include: `<float.h>`

Value: 1.175494e-38 by default, 2.225074e-308 if the switch `-fno-short-double` is used

Remarks: By default, a double type is the same size as a float type (32-bit representation). The `-fno-short-double` switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.

DBL_MIN_10_EXP

Description: Minimum negative integer value for a double precision floating-point exponent in base 10

Include: `<float.h>`

Value: -37 by default, -307 if the switch `-fno-short-double` is used

Remarks: By default, a double type is the same size as a float type (32-bit representation). The `-fno-short-double` switch allows the IEEE 64-bit representation to be used for a double precision floating-point value.

16-Bit Language Tools Libraries

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

Standard C Libraries with Math Functions

FLT_MAX_EXP

Description: Maximum integer value for a single precision floating-point exponent in base FLT_RADIX

Include: <float.h>

Value: 128

FLT_MIN

Description: Minimum single precision floating-point value

Include: <float.h>

Value: 1.175494e-38

FLT_MIN_10_EXP

Description: Minimum negative integer value for a single precision floating-point exponent in base 10

Include: <float.h>

Value: -37

FLT_MIN_EXP

Description: Minimum negative integer value for a single precision floating-point exponent in base FLT_RADIX

Include: <float.h>

Value: -125

FLT_RADIX

Description: Radix of exponent representation

Include: <float.h>

Value: 2

Remarks: The base representation of the exponent is base-2 or binary.

FLT_ROUNDS

Description: Represents the rounding mode for floating-point operations

Include: <float.h>

Value: 1

Remarks: Rounds to the nearest representable value

LDBL_DIG

Description: Number of decimal digits of precision in a long double precision floating-point value

Include: <float.h>

Value: 15

16-Bit Language Tools Libraries

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

Standard C Libraries with Math Functions

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

4.7 <LIMITS.H> IMPLEMENTATION-DEFINED LIMITS

The header file `limits.h` consists of macros that define the minimum and maximum values of integer types. Each of these macros can be used in `#if` preprocessing directives.

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>`

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 long `int`

Include: `<limits.h>`

Value: 9223372036854775807

16-Bit Language Tools Libraries

LLONG_MIN

Description: Minimum value of a long 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

Standard C Libraries with Math Functions

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 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

4.8 <LOCALE.H> LOCALIZATION

This compiler defaults to the C locale and does not support any other locales; therefore it does not support the header file `locale.h`. The following would normally be found in this file:

- struct lconv
- NULL
- LC_ALL
- LC_COLLATE
- LC_CTYPE
- LC_MONETARY
- LC_NUMERIC
- LC_TIME
- localeconv
- setlocale

16-Bit Language Tools Libraries

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 `setjmp` and `longjmp` to save and restore the program environment.

Include: `<setjmp.h>`

Prototype: `typedef int jmp_buf[_NSETJMP];`

Remarks: `_NSETJMP` is defined as `16 + 2` 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 `longjmp`.

Include: `<setjmp.h>`

Prototype: `#define setjmp(jmp_buf env)`

Argument: `env` variable where environment is stored

Return Value: If the return is from a direct call, `setjmp` returns zero. If the return is from a call to `longjmp`, `setjmp` returns a non-zero value.
Note: If the argument `val` from `longjmp` is 0, `setjmp` returns 1.

Example: See `longjmp`.

longjmp

Description: A function that restores the environment saved by `setjmp`.

Include: `<setjmp.h>`

Prototype: `void longjmp(jmp_buf env, int val);`

Arguments: `env` variable where environment is stored
`val` value to be returned to `setjmp` call.

Remarks: The value parameter `val` should be non-zero. If `longjmp` 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 <code>signal</code> to specify that the signal should be ignored.
Include:	<code><signal.h></code>

16-Bit Language Tools Libraries

SIGABRT

Description: Name for the abnormal termination signal.

Include: `<signal.h>`

Prototype: `#define SIGABRT`

Remarks: SIGABRT represents an abnormal termination signal and is used in conjunction with `raise` or `signal`. The default `raise` behavior (action identified by `SIG_DFL`) is to output to the standard error stream:

```
abort - terminating
```

See the example accompanying `signal` to see general usage of signal names and signal handling.

Example:

```
#include <signal.h> /* for raise, SIGABRT */
#include <stdio.h> /* for printf */

int main(void)
{
    raise(SIGABRT);
    printf("Program never reaches here.");
}
```

Output:
ABRT

Explanation:
ABRT stands for "abort".

SIGFPE

Description: Signals floating-point error such as for division by zero or result out of range.

Include: `<signal.h>`

Prototype: `#define SIGFPE`

Remarks: SIGFPE is used as an argument for `raise` and/or `signal`. When used, the default behavior is to print an arithmetic error message and terminate the calling program. This may be overridden by a user function that defines the signal handler actions. See `signal` for an example of a user defined function.

Example:

```
#include <signal.h> /* for raise, SIGFPE */
#include <stdio.h> /* for printf */

int main(void)
{
    raise(SIGFPE);
    printf("Program never reaches here");
}
```

Output:
FPE

Explanation:
FPE stands for "floating-point error".

Standard C Libraries with Math Functions

SIGILL

Description:	Signals illegal instruction.
Include:	<signal.h>
Prototype:	#define SIGILL
Remarks:	SIGILL is used as an argument for <code>raise</code> and/or <code>signal</code> . 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 <code>signal</code> for an example of a user defined function.
Example:	<pre>#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".</pre>

SIGINT

Description:	Interrupt signal.
Include:	<signal.h>
Prototype:	#define SIGINT
Remarks:	SIGINT is used as an argument for <code>raise</code> and/or <code>signal</code> . 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 <code>signal</code> for an example of a user defined function.
Example:	<pre>#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".</pre>

16-Bit Language Tools Libraries

SIGSEGV

Description:	Signals invalid access to storage.
Include:	<code><signal.h></code>
Prototype:	<code>#define SIGSEGV</code>
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."); } Output: SEGV Explanation: SEGV stands for "invalid storage access".</pre>

SIGTERM

Description:	Signals a termination request
Include:	<code><signal.h></code>
Prototype:	<code>#define SIGTERM</code>
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."); } Output: TERM Explanation: TERM stands for "termination request".</pre>

Standard C Libraries with Math Functions

raise

Description: Reports a synchronous signal.
Include: <signal.h>
Prototype: int raise(int sig);
Argument: sig signal name
Return Value: Returns a 0 if successful; otherwise, returns a non-zero value.
Remarks: raise sends the signal identified by sig to the executing program.
Example:

```
#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.

16-Bit Language Tools Libraries

signal

Description: Controls interrupt signal handling.

Include: `<signal.h>`

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

Arguments: `sig` signal name
`func` function to be executed

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

Example:

```
#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 `va_list` declares a variable that will refer to each argument in a variable-length argument list.

Include: `<stdarg.h>`

Example: See `va_arg`.

`va_arg`

Description: Gets the current argument

Include: `<stdarg.h>`

Prototype: `#define va_arg(va_list ap, Ty)`

Argument: `ap` pointer to list of arguments

`Ty` type of argument to be retrieved

Return Value: Returns the current argument

Remarks: `va_start` must be called before `va_arg`.

Example:

```
#include <stdio.h> /* for printf */
#include <stdarg.h> /* for va_arg, va_start,
                    va_list, va_end */
```

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

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

16-Bit Language Tools Libraries

va_arg (Continued)

```
        case '%':
            fmt++;
            if (*fmt == 'd')
            {
                int d = va_arg(ap, int);
                printf("<%d> is an integer\n",d);
            }
            else if (*fmt == 's')
            {
                char *s = va_arg(ap, char*);
                printf("<%s> is a string\n", s);
            }
            else
            {
                printf("%%c is an unknown format\n",
                    *fmt);
            }
            fmt++;
            break;
        default:
            printf("%c is unknown\n", *fmt);
            fmt++;
            break;
    }
}
va_end(ap);
}
```

```
int main(void)
{
    tprint("%d%s.%c", 83, "This is text.", 'a');
}
```

Output:

```
<83> is an integer
<This is text.> is a string
. is unknown
%c is an unknown format
```

Standard C Libraries with Math Functions

va_end

Description: Ends the use of *ap*.

Include: `<stdarg.h>`

Prototype: `#define va_end(va_list ap)`

Argument: *ap* pointer to list of arguments

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>`

16-Bit Language Tools Libraries

offsetof

Description:	Gives the offset of a structure member from the beginning of the structure.
Include:	<code><stddef.h></code>
Prototype:	<code>#define offsetof(T, mbr)</code>
Arguments:	<i>T</i> name of structure <i>mbr</i> name of member in structure T
Return Value:	Returns the offset in bytes of the specified member (<i>mbr</i>) from the beginning of the structure.
Remarks:	The macro <code>offsetof</code> is undefined for bitfields. An error message will occur if bitfields are used.
Example:	<pre>#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)); }</pre>

Output:

```
Offset of item1 = 0
Offset of item2 = 6
Offset of item3 = 8
Offset of item4 = 10
```

Explanation:

This program shows the offset in bytes of each structure member from the start of the structure. Although `item1` is only 5 bytes (`char item1[5]`), padding is added so the address of `item2` falls on an even boundary. The same occurs with `item3`; it is 1 byte (`char item3`) with 1 byte of padding.

4.13 <STDIO.H> INPUT AND OUTPUT

The header file `stdio.h` consists of types, macros and functions that provide support to perform input and output operations on files and streams. When a file is opened it is associated with a stream. A stream is a pipeline for the flow of data into and out of files. Because different systems use different properties, the stream provides more uniform properties to allow reading and writing of the files.

Streams can be text streams or binary streams. Text streams consist of a sequence of characters divided into lines. Each line is terminated with a newline (`'\n'`) character. The characters may be altered in their internal representation, particularly in regards to line endings. Binary streams consist of sequences of bytes of information. The bytes transmitted to the binary stream are not altered. There is no concept of lines, the file is just a series of bytes.

At start-up three streams are automatically opened: `stdin`, `stdout`, and `stderr`. `stdin` provides a stream for standard input, `stdout` is standard output and `stderr` is the standard error. Additional streams may be created with the `fopen` function. See `fopen` for the different types of file access that are permitted. These access types are used by `fopen` and `freopen`.

The type `FILE` is used to store information about each opened file stream. It includes such things as error indicators, end-of-file indicators, file position indicators, and other internal status information needed to control a stream. Many functions in the `stdio` use `FILE` as an argument.

There are three types of buffering: unbuffered, line buffered and fully buffered. Unbuffered means a character or byte is transferred one at a time. Line buffered collects and transfers an entire line at a time (i.e., the newline character indicates the end of a line). Fully buffered allows blocks of an arbitrary size to be transmitted. The functions `setbuf` and `setvbuf` control file buffering.

The `stdio.h` file also contains functions that use input and output formats. The input formats, or scan formats, are used for reading data. Their descriptions can be found under `scanf`, but they are also used by `fscanf` and `sscanf`. The output formats, or print formats, are used for writing data. Their descriptions can be found under `printf`. These print formats are also used by `fprintf`, `sprintf`, `vfprintf`, `vprintf` and `vsprintf`.

Certain compiler options may affect how standard I/O performs. In an effort to provide a more tailored version of the formatted I/O routines, the tool chain may convert a call to a `printf` or `scanf` style function to a different call. The options are summarized below:

- The `-msmart-io` option, when enabled, will attempt to convert `printf`, `scanf` and other functions that use the input output formats to an integer only variant. The functionality is the same as that of the C standard forms, minus the support for floating-point output. `-msmart-io=0` disables this feature and no conversion will take place. `-msmart-io=1` or `-msmart-io` (the default) will convert a function call if it can be proven that an I/O function will never be presented with a floating-point conversion. `-msmart-io=2` is more optimistic than the default and will assume that non-constant format strings or otherwise unknown format strings will not contain a floating-point format. In the event that `-msmart-io=2` is used with a floating-point format, the format letter will appear as literal text and its corresponding argument will not be consumed.
- `-fno-short-double` will cause the compiler to generate calls to formatted I/O routines that support `double` as if it were a `long double` type.

Mixing modules compiled with these options may result in a larger executable size, or incorrect execution if large and small double-sized data is shared across modules.

16-Bit Language Tools Libraries

FILE

Description: Stores information for a file stream.

Include: <stdio.h>

fpos_t

Description: Type of a variable used to store a file position.

Include: <stdio.h>

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

Standard C Libraries with Math Functions

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.

16-Bit Language Tools Libraries

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

Standard C Libraries with Math Functions

clearerr

Description: Resets the error indicator for the stream

Include: <stdio.h>

Prototype: void clearerr(FILE *stream);

Argument: *stream* stream to reset error indicators

Remarks: The function clears the end-of-file and error indicators for the given stream (i.e., feof and ferror will return false after the function clearerr is called).

Example:

```
/* This program tries to write to a file that is */
/* readonly. This causes the error indicator to */
/* be set. The function ferror is used to check */
/* the error indicator. The function clearerr is */
/* used to reset the error indicator so the next */
/* time ferror is called it will not report an */
/* error. */
#include <stdio.h> /* for ferror, clearerr, */
                  /* printf, fprintf, fopen, */
                  /* fclose, FILE, NULL */

int main(void)
{
    FILE *myfile;

    if ((myfile = fopen("sampclearerr.c", "r")) ==
        NULL)
        printf("Cannot open file\n");
    else
    {
        fprintf(myfile, "Write this line to the "
                "file.\n");
        if (ferror(myfile))
            printf("Error\n");
        else
            printf("No error\n");
        clearerr(myfile);
        if (ferror(myfile))
            printf("Still has Error\n");
        else
            printf("Error indicator reset\n");

        fclose(myfile);
    }
}
```

Output:

```
Error
Error indicator reset
```

16-Bit Language Tools Libraries

fclose

Description: Close a stream.

Include: <stdio.h>

Prototype: int fclose(FILE *stream);

Argument: *stream* pointer to the stream to close

Return Value: Returns 0 if successful; otherwise, returns EOF if any errors were detected.

Remarks: fclose writes any buffered output to the file.

Example:

```
#include <stdio.h> /* for fopen, fclose, */
                  /* printf, FILE, NULL, EOF */

int main(void)
{
    FILE *myfile1, *myfile2;
    int y;

    if ((myfile1 = fopen("afile1", "w+")) == NULL)
        printf("Cannot open afile1\n");
    else
    {
        printf("afile1 was opened\n");

        y = fclose(myfile1);
        if (y == EOF)
            printf("afile1 was not closed\n");
        else
            printf("afile1 was closed\n");
    }
}
```

Output:
afile1 was opened
afile1 was closed

Standard C Libraries with Math Functions

feof

Description: Tests for end-of-file
Include: <stdio.h>
Prototype: int feof(FILE *stream);
Argument: *stream* stream to check for end-of-file
Return Value: Returns non-zero if stream is at the end-of-file; otherwise, returns zero.
Example:

```
#include <stdio.h> /* for feof, fgetc, fputc, */
                  /* fopen, fclose, FILE, */
                  /* NULL */

int main(void)
{
    FILE *myfile;
    int y = 0;

    if( (myfile = fopen( "afile.txt", "rb" )) == NULL )
        printf( "Cannot open file\n" );
    else
    {
        for (;;)
        {
            y = fgetc(myfile);
            if (feof(myfile))
                break;
            fputc(y, stdout);
        }
        fclose( myfile );
    }
}
```

Input:

Contents of afile.txt (used as input):
This is a sentence.

Output:

This is a sentence.

16-Bit Language Tools Libraries

fclose

Description: Tests if error indicator is set.

Include: <stdio.h>

Prototype: int fclose(FILE *stream);

Argument: stream pointer to FILE structure

Return Value: Returns a non-zero value if error indicator is set; otherwise, returns a zero.

Example:

```
/* This program tries to write to a file that is */
/* readonly. This causes the error indicator to */
/* be set. The function fclose 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 fclose is called */
/* it will not report an error. */
```

```
#include <stdio.h> /* for fclose, clearerr, */
                  /* printf, fprintf, */
                  /* fopen, fclose, */
                  /* FILE, NULL */

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 (fclose(myfile))
            printf("Error\n");
        else
            printf("No error\n");
        clearerr(myfile);
        if (fclose(myfile))
            printf("Still has Error\n");
        else
            printf("Error indicator reset\n");

        fclose(myfile);
    }
}
```

Output:

```
Error
Error indicator reset
```

Standard C Libraries with Math Functions

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|
|
```

16-Bit Language Tools Libraries

fgetpos

Description: Gets the stream's file position.

Include: <stdio.h>

Prototype: int fgetpos(FILE *stream, fpos_t *pos);

Arguments: stream target stream
pos position-indicator storage

Return Value: Returns 0 if successful; otherwise, returns a non-zero value.

Remarks: The function stores the file-position indicator for the given stream in *pos if successful, otherwise, fgetpos sets errno.

Example:

```
/* This program opens a file and reads bytes at */
/* several different locations. The fgetpos */
/* function notes the 8th byte. 21 bytes are */
/* read then 18 bytes are read. Next the */
/* fsetpos function is set based on the */
/* fgetpos position and the previous 21 bytes */
/* are reread. */
```

```
#include <stdio.h> /* for fgetpos, fread, */
/* printf, fopen, fclose, */
/* FILE, NULL, perror, */
/* fpos_t, sizeof */
```

```
int main(void)
{
    FILE *myfile;
    fpos_t pos;
    char buf[25];

    if ((myfile = fopen("sampfgetpos.c", "rb")) ==
        NULL)
        printf("Cannot open file\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
```

Standard C Libraries with Math Functions

fgets

Description:	Get a string from a stream
Include:	<stdio.h>
Prototype:	<code>char *fgets(char *s, int n, FILE *stream);</code>
Arguments:	<code>s</code> pointer to the storage string <code>n</code> maximum number of characters to read <code>stream</code> pointer to the open stream.
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 input stream and stores them into the string pointed to by <code>s</code> until it has read <code>n-1</code> 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 <code>fgets</code> 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 <code>afile.txt</code> (used as input): Short Longer string</p> <p>Output: Short Longer string </p>

16-Bit Language Tools Libraries

fopen

Description: Opens a file.

Include: `<stdio.h>`

Prototype: `FILE *fopen(const char *filename, const char *mode);`

Arguments: *filename* name of the file
mode type of access permitted

Return Value: Returns a pointer to the open stream. If the function fails a null pointer is returned.

Remarks: Following are the types of file access:

- `r -` opens an existing text file for reading
- `w -` opens an empty text file for writing. (An existing file will be overwritten.)
- `a -` opens a text file for appending. (A file is created if it doesn't exist.)
- `rb -` opens an existing binary file for reading.
- `wb -` opens an empty binary file for writing. (An existing file will be overwritten.)
- `ab -` opens a binary file for appending. (A file is created if it doesn't exist.)
- `r+ -` opens an existing text file for reading and writing.
- `w+ -` opens an empty text file for reading and writing. (An existing file will be overwritten.)
- `a+ -` opens a text file for reading and appending. (A file is created if it doesn't exist.)
- `r+b or rb+ -` opens an existing binary file for reading and writing.
- `w+b or wb+ -` opens an empty binary file for reading and writing. (An existing file will be overwritten.)
- `a+b or ab+ -` opens a binary file for reading and appending. (A file is created if it doesn't exist.)

Example:

```
#include <stdio.h> /* for fopen, fclose, */
                  /* printf, FILE, */
                  /* NULL, EOF */

int main(void)
{
    FILE *myfile1, *myfile2;
    int y;
```

fopen (Continued)

```
if ((myfile1 = fopen("afile1", "r")) == NULL)
    printf("Cannot open afile1\n");
else
{
    printf("afile1 was opened\n");
    y = fclose(myfile1);
    if (y == EOF)
        printf("afile1 was not closed\n");
    else
        printf("afile1 was closed\n");
}

if ((myfile1 = fopen("afile1", "w+")) == NULL)
    printf("Second try, cannot open afile1\n");
else
{
    printf("Second try, afile1 was opened\n");
    y = fclose(myfile1);
    if (y == EOF)
        printf("afile1 was not closed\n");
    else
        printf("afile1 was closed\n");
}

if ((myfile2 = fopen("afile2", "w+")) == NULL)
    printf("Cannot open afile2\n");
else
{
    printf("afile2 was opened\n");
    y = fclose(myfile2);
    if (y == EOF)
        printf("afile2 was not closed\n");
    else
        printf("afile2 was closed\n");
}
}
```

Output:

```
Cannot open afile1
Second try, afile1 was opened
afile1 was closed
afile2 was opened
afile2 was closed
```

Explanation:

afile1 must exist before it can be opened for reading (r) or the fopen function will fail. If the fopen function opens a file for writing (w+) it does not have to already exist. If it doesn't exist, it will be created and then opened.

16-Bit Language Tools Libraries

fprintf

Description: Prints formatted data to a stream.

Include: <stdio.h>

Prototype: int fprintf(FILE *stream, const char *format, ...);

Arguments: stream pointer to the stream in which to output data
format format control string
... optional arguments

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

Remarks: The format argument has the same syntax and use that it has in print.

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

Standard C Libraries with Math Functions

fputc

Description: Puts a character to the stream.

Include: <stdio.h>

Prototype: `int fputc(int c, FILE *stream);`

Arguments: *c* character to be written
stream pointer to the open stream

Return Value: Returns the character written or EOF if a write error occurs.

Remarks: The function writes the character to the output stream, advances the file-position indicator and returns the character as an unsigned char converted to an int.

Example: `#include <stdio.h> /* for fputc, EOF, stdout */`

```
int main(void)
{
    char *y;
    char buf[] = "This is text\n";
    int x;

    x = 0;

    for (y = buf; (x != EOF) && (*y != '\0'); y++)
    {
        x = fputc(*y, stdout);
        fputc('|', stdout);
    }
}
```

Output:

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

fputs

Description: Puts a string to the stream.

Include: <stdio.h>

Prototype: `int fputs(const char *s, FILE *stream);`

Arguments: *s* string to be written
stream pointer to the open stream

Return Value: Returns a non-negative value if successful; otherwise, returns EOF.

Remarks: The function writes characters to the output stream up to but not including the null character.

Example: `#include <stdio.h> /* for fputs, stdout */`

```
int main(void)
{
    char buf[] = "This is text\n";

    fputs(buf, stdout);
    fputs("|", stdout);
}
```

Output:

```
This is text
|
```

16-Bit Language Tools Libraries

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: `ptr` pointer to the storage buffer
`size` size of item
`nelem` maximum number of items to be read
`stream` pointer to the stream

Return Value: Returns the number of complete elements read up to `nelem` whose size is specified by `size`.

Remarks: The function reads characters from a given stream into the buffer pointed to by `ptr` until the function stores `size * nelem` characters or sets the end-of-file or error indicator. `fread` returns `n/size` where `n` is the number of characters it read. If `n` is not a multiple of `size`, the value of the last element is indeterminate. If the function sets the error indicator, the file-position indicator is indeterminate.

Example:

```
#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");
}
```

Standard C Libraries with Math Functions

fread (Continued)

```
if ((buf = fopen("afile.out", "r+")) != NULL)
{
    numread = fread(readnums, sizeof(double),
                    10, buf);
    printf("Read %d numbers\n", numread);
    for (x = 0; x < 10; x++)
    {
        printf("%d * %f = %f\n", x+1, readnums[x],
              (x + 1) * readnums[x]);
    }
    fclose(buf);
}
else
    printf("Cannot open afile.out\n");
}
```

Output:

```
10.0/1 = 10.000000
10.0/2 = 5.000000
10.0/3 = 3.333333
10.0/4 = 2.500000
10.0/5 = 2.000000
10.0/6 = 1.666667
10.0/7 = 1.428571
10.0/8 = 1.250000
10.0/9 = 1.111111
10.0/10 = 1.000000
Wrote 10 numbers
```

```
Read 10 numbers
1 * 10.000000 = 10.000000
2 * 5.000000 = 10.000000
3 * 3.333333 = 10.000000
4 * 2.500000 = 10.000000
5 * 2.000000 = 10.000000
6 * 1.666667 = 10.000000
7 * 1.428571 = 10.000000
8 * 1.250000 = 10.000000
9 * 1.111111 = 10.000000
10 * 1.000000 = 10.000000
```

Explanation:

This program uses `fwrite` to save 10 numbers to a file in binary form. This allows the numbers to be saved in the same pattern of bits as the program is using which provides more accuracy and consistency. Using `fprintf` would save the numbers as text strings which could cause the numbers to be truncated. Each number is divided into 10 to produce a variety of numbers. Retrieving the numbers with `fread` to a new array and multiplying them by the original number shows the numbers were not truncated in the save process.

16-Bit Language Tools Libraries

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: filename name of the new file
mode type of access permitted
stream pointer to the currently open stream

Return Value: Returns a pointer to the new open file. If the function fails a null pointer is returned.

Remarks: The function closes the file associated with the stream as though fclose was called. Then it opens the new file as though fopen was called. freopen will fail if the specified stream is not open. See fopen for the possible types of file access.

Example:

```
#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>

Standard C Libraries with Math Functions

fscanf (Continued)

Prototype: `int fscanf(FILE *stream, const char *format, ...);`

Arguments: `stream` pointer to the open stream from which to read data
`format` format control string
`...` optional arguments

Return Value: Returns the number of items successfully converted and assigned. If no items are assigned, a 0 is returned. EOF is returned if end-of-file is encountered before the first conversion or if an error occurs.

Remarks: The format argument has the same syntax and use that it has in `scanf`.

Example:

```
#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

16-Bit Language Tools Libraries

fseek

Description: Moves file pointer to a specific location.

Include: <stdio.h>

Prototype: int fseek(FILE *stream, long offset, int mode);

Arguments: stream stream in which to move the file pointer.
offset value to add to the current position
mode type of seek to perform

Return Value: Returns 0 if successful; otherwise, returns a non-zero value and 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);
        }
    }
}
```

fseek (Continued)

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

Output:

```
"This is the beginning"
```

```
"this is the middle and this is the end."
```

```
"this is the end."
```

Explanation:

The file `afile.out` is created with the text, "This is the beginning, this is the middle and this is the end".

The function `fseek` uses an offset of zero and `SEEK_SET` to set the file pointer to the beginning of the file. `fgets` then reads 22 characters which are "This is the beginning", and adds a null character to the string.

Next, `fseek` uses an offset of two and `SEEK_CURRENT` to set the file pointer to the current position plus two (skipping the comma and space). `fgets` then reads up to the next 70 characters. The first 39 characters are "this is the middle and this is the end". It stops when it reads EOF and adds a null character to the string.

Finally, `fseek` uses an offset of negative 16 characters and `SEEK_END` to set the file pointer to 16 characters from the end of the file. `fgets` then reads up to 70 characters. It stops at the EOF after reading 16 characters "this is the end". and adds a null character to the string.

fsetpos

Description:	Sets the stream's file position.
Include:	<code><stdio.h></code>
Prototype:	<code>int fsetpos(FILE *stream, const fpos_t *pos);</code>
Arguments:	<code>stream</code> target stream <code>pos</code> 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 <code>*pos</code> if successful; otherwise, <code>fsetpos</code> sets <code>errno</code> .

16-Bit Language Tools Libraries

fsetpos (Continued)

```
Example:      /* This program opens a file and reads bytes at */
                /* several different locations. The fgetpos */
                /* function notes the 8th byte. 21 bytes are */
                /* read then 18 bytes are read. Next the */
                /* fsetpos function is set based on the */
                /* fgetpos position and the previous 21 bytes */
                /* are reread. */

#include <stdio.h> /* for fgetpos, fread, */
                  /* printf, fopen, fclose, */
                  /* FILE, NULL, perror, */
                  /* fpos_t, sizeof */

int main(void)
{
    FILE *myfile;
    fpos_t pos;
    char buf[25];

    if ((myfile = fopen("sampfgetpos.c", "rb")) ==
        NULL)
        printf("Cannot open file\n");
    else
    {
        fread(buf, sizeof(char), 8, myfile);
        if (fgetpos(myfile, &pos) != 0)
            perror("fgetpos error");
        else
        {
            fread(buf, sizeof(char), 21, myfile);
            printf("Bytes read: %.21s\n", buf);
            fread(buf, sizeof(char), 18, myfile);
            printf("Bytes read: %.18s\n", buf);
        }

        if (fsetpos(myfile, &pos) != 0)
            perror("fsetpos error");

        fread(buf, sizeof(char), 21, myfile);
        printf("Bytes read: %.21s\n", buf);
        fclose(myfile);
    }
}
```

Output:

```
Bytes read: program opens a file
Bytes read: and reads bytes at
Bytes read: program opens a file
```

Standard C Libraries with Math Functions

ftell

Description: Gets the current position of a file pointer.
Include: <stdio.h>
Prototype: long ftell(FILE *stream);
Argument: stream stream in which to get the current file position
Return Value: Returns the position of the file pointer if successful; otherwise, returns -1.

Example:

```
#include <stdio.h> /* for ftell, fread,      */
                  /* fprintf, printf,     */
                  /* fopen, fclose, sizeof, */
                  /* FILE, NULL */

int main(void)
{
    FILE *myfile;
    char s[75];
    long y;

    myfile = fopen("afile.out", "w+");
    if (myfile == NULL)
        printf("Cannot open afile.out\n");
    else
    {
        fprintf(myfile, "This is a very long sentence "
                  "for input into the file named "
                  "afile.out for testing.");

        fclose(myfile);

        if ((myfile = fopen("afile.out", "rb")) != NULL)
        {
            printf("Read some characters:\n");
            fread(s, sizeof(char), 29, myfile);
            printf("\t\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
```

16-Bit Language Tools Libraries

fwrite

Description:	Writes data to the stream.
Include:	<stdio.h>
Prototype:	<pre>size_t fwrite(const void *ptr, size_t size, size_t nelem, FILE *stream);</pre>
Arguments:	<pre>ptr pointer to the storage buffer size size of item nelem maximum number of items to be read stream pointer to the open stream</pre>
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.
Example:	<pre>#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"); }</pre>

Standard C Libraries with Math Functions

fwrite (Continued)

```
if ((buf = fopen("afile.out", "r+")) != NULL)
{
    numread = fread(readnums, sizeof(double),
                    10, buf);
    printf("Read %d numbers\n", numread);
    for (x = 0; x < 10; x++)
    {
        printf("%d * %f = %f\n", x+1, readnums[x],
              (x + 1) * readnums[x]);
    }
    fclose(buf);
}
else
    printf("Cannot open afile.out\n");
}
```

Output:

```
10.0/1 = 10.000000
10.0/2 = 5.000000
10.0/3 = 3.333333
10.0/4 = 2.500000
10.0/5 = 2.000000
10.0/6 = 1.666667
10.0/7 = 1.428571
10.0/8 = 1.250000
10.0/9 = 1.111111
10.0/10 = 1.000000
Wrote 10 numbers
```

```
Read 10 numbers
1 * 10.000000 = 10.000000
2 * 5.000000 = 10.000000
3 * 3.333333 = 10.000000
4 * 2.500000 = 10.000000
5 * 2.000000 = 10.000000
6 * 1.666667 = 10.000000
7 * 1.428571 = 10.000000
8 * 1.250000 = 10.000000
9 * 1.111111 = 10.000000
10 * 1.000000 = 10.000000
```

Explanation:

This program uses `fwrite` to save 10 numbers to a file in binary form. This allows the numbers to be saved in the same pattern of bits as the program is using which provides more accuracy and consistency. Using `fprintf` would save the numbers as text strings, which could cause the numbers to be truncated. Each number is divided into 10 to produce a variety of numbers. Retrieving the numbers with `fread` to a new array and multiplying them by the original number shows the numbers were not truncated in the save process.

16-Bit Language Tools Libraries

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;

    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|
|
```

Standard C Libraries with Math Functions

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);
}
```

Input:
Contents of `UartIn.txt` (used as `stdin` input for simulator):
Short
Longer string

Output:
S|h|o|r|t|

gets

Description: Get a string from `stdin`.

Include: `<stdio.h>`

Prototype: `char *gets(char *s);`

Argument: `s` pointer to the storage string

Return Value: Returns a pointer to the string `s` if successful; otherwise, returns a null pointer

Remarks: The function reads characters from the stream `stdin` and stores them into the string pointed to by `s` until it reads a newline character (which is not stored) or sets the end-of-file or error indicators. If any characters were read, a null character is stored immediately after the last read character in the next element of the array. If `gets` sets the error indicator, the array contents are indeterminate.

16-Bit Language Tools Libraries

gets (Continued)

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

```
int main(void)
{
    char y[50];

    gets(y) ;
    printf("Text: %s\n", y);
}
```

Input:

Contents of UartIn.txt (used as stdin input for simulator):

Short

Longer string

Output:

Text: Short

perror

Description: Prints an error message to stderr.

Include: `<stdio.h>`

Prototype: `void perror(const char *s);`

Argument: `s` string to print

Return Value: None.

Remarks: The string `s` is printed followed by a colon and a space. Then an error message based on `errno` is printed followed by a newline

Example: `#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

Standard C Libraries with Math Functions

printf

Description:	Prints formatted text to <code>stdout</code> .
Include:	<code><stdio.h></code>
Prototype:	<code>int printf(const char *format, ...);</code>
Arguments:	<code>format</code> format control string ... optional arguments
Return Value:	Returns number of characters generated or a negative number if an error occurs.
Remarks:	<p>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:</p> <pre>%[flags][width][.precision][size]type</pre> <p>flags</p> <ul style="list-style-type: none">- left justify the value within a given field width0 Use 0 for the pad character instead of space (which is the default)+ generate a plus sign for positive signed valuesspace 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 <p>width</p> <p>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 <code>int</code>) 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.</p> <p>precision</p> <p>The field width can be followed with dot (.) and a decimal integer representing the precision that specifies one of the following:</p> <ul style="list-style-type: none">- 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 <p>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 <code>int</code>) will be used for the precision.</p>

16-Bit Language Tools Libraries

printf (Continued)

size

- h modifier – used with type d, i, o, u, x, X; converts the value to a short int or unsigned short int
- h modifier – used with n; specifies that the pointer points to a short int
- l modifier – used with type d, i, o, u, x, X; converts the value to a long int or unsigned long int
- l modifier – used with n; specifies that the pointer points to a long int
- l modifier – used with c; specifies a wide character
- l modifier – used with type e, E, f, F, g, G; converts the value to a double
- ll modifier – used with type d, i, o, u, x, X; converts the value to a long long int or unsigned long long int
- ll modifier – used with n; specifies that the pointer points to a long long int
- L modifier – used with e, E, f, g, 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');

    /* 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);
}
```

Standard C Libraries with Math Functions

printf (Continued)

```
/* 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 *c*, FILE **stream*);

Arguments: *c* character to be written
stream pointer to FILE structure

Return Value: Returns the character or EOF if an error occurs or end-of-file is reached.

Remarks: putc is the same as the function fputc.

Example: #include <stdio.h> /* for putc, EOF, stdout */

```
int main(void)
{
    char *y;
    char buf[] = "This is text\n";
    int x;

    x = 0;

    for (y = buf; (x != EOF) && (*y != '\0'); y++)
    {
        x = putc(*y, stdout);
        putc('|', stdout);
    }
}
```

Output:

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

16-Bit Language Tools Libraries

putchar

Description: Put a character to `stdout`.

Include: `<stdio.h>`

Prototype: `int putchar(int c);`

Argument: `c` character to be written

Return Value: Returns the character or EOF if an error occurs or end-of-file is reached.

Remarks: Same effect as `fputc` with `stdout` as an argument.

Example:

```
#include <stdio.h> /* for putchar, printf, */
                    /* EOF, stdout          */
```

```
int main(void)
{
    char *y;
    char buf[] = "This is text\n";
    int x;

    x = 0;

    for (y = buf; (x != EOF) && (*y != '\0'); y++)
        x = putchar(*y);
}
```

Output:
This is text

puts

Description: Put a string to `stdout`.

Include: `<stdio.h>`

Prototype: `int puts(const char *s);`

Argument: `s` string to be written

Return Value: Returns a non-negative value if successful; otherwise, returns EOF.

Remarks: The function writes characters to the stream `stdout`. A newline character is appended. The terminating null character is not written to the stream.

Example:

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

```
int main(void)
{
    char buf[] = "This is text\n";

    puts(buf);
    puts("|");
}
```

Output:
This is text

|

Standard C Libraries with Math Functions

remove

Description: Deletes the specified file.

Include: `<stdio.h>`

Prototype: `int remove(const char *filename);`

Argument: *filename* name of file to be deleted.

Return Value: Returns 0 if successful, -1 if not.

Remarks: If filename does not exist or is open, remove will fail.

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

```
int main(void)
{
    if (remove("myfile.txt") != 0)
        printf("Cannot remove file");
    else
        printf("File removed");
}
```

Output:
File removed

rename

Description: Renames the specified file.

Include: `<stdio.h>`

Prototype: `int rename(const char *old, const char *new);`

Arguments: *old* pointer to the old name
new pointer to the new name.

Return Value: Return 0 if successful, non-zero if not.

Remarks: The new name must not already exist in the current working directory, the old name must exist in the current working directory.

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

```
int main(void)
{
    if (rename("myfile.txt", "newfile.txt") != 0)
        printf("Cannot rename file");
    else
        printf("File renamed");
}
```

Output:
File renamed

16-Bit Language Tools Libraries

rewind

Description: Resets the file pointer to the beginning of the file.

Include: <stdio.h>

Prototype: void rewind(FILE *stream);

Argument: *stream* stream to reset the file pointer

Remarks: The function calls `fseek(stream, 0L, SEEK_SET)` and then clears the error indicator for the given stream.

Example:

```
#include <stdio.h> /* for rewind, fopen, */
                    /* fscanf, fclose, */
                    /* fprintf, printf, */
                    /* FILE, NULL */

int main(void)
{
    FILE *myfile;
    char s[] = "cookies";
    int x = 10;

    if ((myfile = fopen("afile", "w+")) == NULL)
        printf("Cannot open afile\n");
    else
    {
        fprintf(myfile, "%d %s", x, s);
        printf("I have %d %s.\n", x, s);

        /* set pointer to beginning of file */
        rewind(myfile);
        fscanf(myfile, "%d %s", &x, &s);
        printf("I ate %d %s.\n", x, s);

        fclose(myfile);
    }
}
```

Output:

```
I have 10 cookies.
I ate 10 cookies.
```

Standard C Libraries with Math Functions

scanf

Description:	Scans formatted text from <code>stdin</code> .
Include:	<code><stdio.h></code>
Prototype:	<code>int scanf(const char *format, ...);</code>
Argument:	<code>format</code> 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:	Each format specifier begins with a percent sign followed by optional fields and a required type as shown here: %[*] [width] [modifier] type * indicates assignment suppression. This will cause the input field to be skipped and no assignment made. width specify the maximum number of input characters to match for the conversion not including white space that can be skipped. modifier h modifier – used with type d, i, o, u, x, X; converts the value to a short int or unsigned short int. 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

16-Bit Language Tools Libraries

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
```

Standard C Libraries with Math Functions

setbuf

Description: Defines how a stream is buffered.

Include: `<stdio.h>`

Prototype: `void setbuf(FILE *stream, char *buf);`

Arguments: *stream* pointer to the open stream
buf user allocated buffer

Remarks: `setbuf` must be called after `fopen` but before any other function calls that operate on the stream. If *buf* is a null pointer, `setbuf` calls the function `setvbuf(stream, 0, _IONBF, BUFSIZ)` for no buffering; otherwise `setbuf` calls `setvbuf(stream, buf, _IOFBF, BUFSIZ)` for full buffering with a buffer of size `BUFSIZ`. See `setvbuf`.

Example:

```
#include <stdio.h> /* for setbuf, printf, */
                  /* fopen, fclose,      */
                  /* FILE, NULL, BUFSIZ */

int main(void)
{
    FILE *myfile1, *myfile2;
    char buf[BUFSIZ];

    if ((myfile1 = fopen("afile1", "w+")) != NULL)
    {
        setbuf(myfile1, NULL);
        printf("myfile1 has no buffering\n");
        fclose(myfile1);
    }

    if ((myfile2 = fopen("afile2", "w+")) != NULL)
    {
        setbuf(myfile2, buf);
        printf("myfile2 has full buffering");
        fclose(myfile2);
    }
}
```

Output:

```
myfile1 has no buffering
myfile2 has full buffering
```

16-Bit Language Tools Libraries

setvbuf

Description: Defines the stream to be buffered and the buffer size.

Include: <stdio.h>

Prototype: `int setvbuf(FILE *stream, char *buf, int mode, size_t size);`

Arguments:

- `stream` pointer to the open stream
- `buf` user allocated buffer
- `mode` type of buffering
- `size` size of buffer

Return Value: Returns 0 if successful

Remarks: `setvbuf` must be called after `fopen` but before any other function calls that operate on the stream. For mode use one of the following:
`_IOFBF` – for full buffering
`_IOLBF` – for line buffering
`_IONBF` – for no buffering

Example:

```
#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
```

Standard C Libraries with Math Functions

sprintf

Description:	Prints formatted text to a string
Include:	<stdio.h>
Prototype:	int sprintf(char * <i>s</i> , const char * <i>format</i> , ...);
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 <i>s</i> excluding the terminating null character.
Remarks:	The format argument has the same syntax and use that it has in printf.
Example:	<pre>#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); }</pre>

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 * <i>s</i> , const char * <i>format</i> , ...);
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.

16-Bit Language Tools Libraries

sscanf (Continued)

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);
}
```

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: `#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");
}
```

Output:

```
Temporary file was created
```

Standard C Libraries with Math Functions

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.

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.

16-Bit Language Tools Libraries

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

Standard C Libraries with Math Functions

fprintf

Description: Prints formatted data to a stream using a variable length argument list.

Include: <stdio.h>
<stdarg.h>

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

Arguments: stream pointer to the open stream
format format control string
ap pointer to a list of arguments

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

Remarks: The format argument has the same syntax and use that it has in printf.

To access the variable length argument list, the *ap* variable must be initialized by the macro *va_start* and may be reinitialized by additional calls to *va_arg*. This must be done before the *fprintf* function is called. Invoke *va_end* after the function returns. For more details see *stdarg.h*.

Example:

```
#include <stdio.h> /* for fprintf, 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);
    fprintf(myfile, fmt, ap);
    va_end(ap);
}

int main(void)
{
    int num = 3;

    if ((myfile = fopen("afile.txt", "w")) == NULL)
        printf("Cannot open afile.txt\n");
    else
    {
        errmsg("Error: The letter '%c' is not %s\n", 'a',
              "an integer value.");
        errmsg("Error: Requires %d%s%c", num,
              " or more characters.", '\n');
    }
    fclose(myfile);
}
```

Output:

Contents of afile.txt

Error: The letter 'a' is not an integer value.
Error: Requires 3 or more characters.

16-Bit Language Tools Libraries

vprintf

Description: Prints formatted text to `stdout` using a variable length argument list

Include: `<stdio.h>`
`<stdarg.h>`

Prototype: `int vprintf(const char *format, va_list ap);`

Arguments: `format` format control string
`ap` pointer to a list of arguments

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

Remarks: The format argument has the same syntax and use that it has in `printf`.

To access the variable length argument list, the `ap` variable must be initialized by the macro `va_start` and may be reinitialized by additional calls to `va_arg`. This must be done before the `vprintf` function is called. Invoke `va_end` after the function returns. For more details see `stdarg.h`

Example:

```
#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.
```

Standard C Libraries with Math Functions

vsprintf

Description: Prints formatted text to a string using a variable length argument list

Include: `<stdio.h>`
`<stdarg.h>`

Prototype: `int vsprintf(char *s, const char *format, va_list ap);`

Arguments: `s` storage string for output
`format` format control string
`ap` pointer to a list of arguments

Return Value: Returns number of characters stored in `s` excluding the terminating null character.

Remarks: The format argument has the same syntax and use that it has in `printf`.

To access the variable length argument list, the `ap` variable must be initialized by the macro `va_start` and may be reinitialized by additional calls to `va_arg`. This must be done before the `vsprintf` function is called. Invoke `va_end` after the function returns. For more details see `stdarg.h`

Example:

```
#include <stdio.h> /* for vsprintf, printf */
#include <stdarg.h> /* for va_start,          */
                          /* va_list, va_end */
```

```
void errmsg(const char *fmt, ...)
{
    va_list ap;
    char buf[100];

    va_start(ap, fmt);
    vsprintf(buf, fmt, ap);
    va_end(ap);
    printf("Error: %s", buf);
}

int main(void)
{
    int num = 3;

    errmsg("The letter '%c' is not %s\n", 'a',
           "an integer value.");
    errmsg("Requires %d%s\n", num,
           " or more characters.\n");
}
```

Output:
Error: The letter 'a' is not an integer value.
Error: Requires 3 or more characters.

16-Bit Language Tools Libraries

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>`

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.

Standard C Libraries with Math Functions

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

16-Bit Language Tools Libraries

abs

Description: Calculates the absolute value.

Include: <stdlib.h>

Prototype: int abs(int *i*);

Argument: *i* integer value

Return Value: Returns the absolute value of *i*.

Remarks: A negative number is returned as positive; a positive number is unchanged.

Example:

```
#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);
```

Standard C Libraries with Math Functions

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
```

16-Bit Language Tools Libraries

atof

Description: Converts a string to a double precision floating-point value.

Include: <stdlib.h>

Prototype: double atof(const char *s);

Argument: *s* pointer to the string to be converted

Return Value: Returns the converted value if successful; otherwise, returns 0.

Remarks: The number may consist of the following:
[whitespace] [sign] digits [.digits]
[{ e | E } [sign] digits]
optional *whitespace*, followed by an optional *sign* then a sequence of one or more *digits* with an optional decimal point, followed by one or more optional *digits* and an optional *e* or *E* followed by an optional signed exponent. The conversion stops when the first unrecognized character is reached. The conversion is the same as `strtod(s, 0, 0)` except it does no error checking so `errno` will not be set.

Example:

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

int main(void)
{
    char a[] = " 1.28";
    char b[] = "27.835e2";
    char c[] = "Number1";
    double x;

    x = atof(a);
    printf("String = \"%s\" float = %f\n", a, x);

    x = atof(b);
    printf("String = \"%s\" float = %f\n", b, x);

    x = atof(c);
    printf("String = \"%s\" float = %f\n", c, x);
}

Output:
String = "1.28" float = 1.280000
String = "27.835:e2" float = 2783.500000
String = "Number1" float = 0.000000
```

Standard C Libraries with Math Functions

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.

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\" \tint = %d\n", a, x);

    x = atoi(b);
    printf("String = \"%s\" \tint = %d\n", b, x);
}
```

Output:

```
String = " -127"          int = -127
String = "Number1"       int = 0
```

atol

Description: Converts a string to a long integer.

Include: <stdlib.h>

Prototype: 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.

16-Bit Language Tools Libraries

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:

- key* object to search for
- base* pointer to the start of the search data
- nelem* number of elements
- size* size of elements
- cmp* pointer to the comparison function
- ck* pointer to the key for the search
- ce* pointer to the element being compared with the key.

Return Value: Returns a pointer to the object being searched for if found; otherwise, returns NULL.

Remarks: The value returned by the compare function is <0 if *ck* is less than *ce*, 0 if *ck* is equal to *ce*, or >0 if *ck* is greater than *ce*. In the following example, *qsort* is used to sort the list before *bsearch* is called. *bsearch* requires the list to be sorted according to the comparison function. This *comp* uses ascending order.

Standard C Libraries with Math Functions

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
```

16-Bit Language Tools Libraries

calloc

Description: Allocates an array in memory and initializes the elements to 0.

Include: `<stdlib.h>`

Prototype: `void *calloc(size_t nelem, size_t size);`

Arguments: *nelem* number of elements
size length of each element

Return Value: Returns a pointer to the allocated space if successful; otherwise, returns a null pointer.

Remarks: Memory returned by `calloc` is aligned correctly for any size data element and is initialized to zero.

Example:

```
/* 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 numer, int denom);`

Arguments: *numer* numerator
denom denominator

Return Value: Returns the quotient and the remainder.

Remarks: The returned quotient will have the same sign as the numerator divided by the denominator. The sign for the remainder will be such that the quotient times the denominator plus the remainder will equal the numerator ($quot * denom + rem = numer$). Division by zero will invoke the math exception error, which by default, will cause a reset. Write a math error handler to do something else.

Standard C Libraries with Math Functions

div (Continued)

```
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);
}
```

16-Bit Language Tools Libraries

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
```

Standard C Libraries with Math Functions

free

Description: Frees memory.

Include: <stdlib.h>

Prototype: void free(void *ptr);

Argument: ptr points to memory to be freed

Remarks: Frees memory previously allocated with calloc, malloc, or realloc. If free is used on space that has already been deallocated (by a previous call to free or by realloc) or on space not allocated with calloc, malloc, or realloc, the behavior is undefined.

Example:

```
#include <stdio.h> /* for printf, sizeof, */
                        /* NULL */
#include <stdlib.h> /* for malloc, free */

int main(void)
{
    long *i;

    if ((i = (long *)malloc(50 * sizeof(long))) ==
        NULL)
        printf("Cannot allocate memory\n");
    else
    {
        printf("Memory allocated\n");
        free(i);
        printf("Memory freed\n");
    }
}
```

Output:

Memory allocated
Memory freed

getenv

Description: Get a value for an environment variable.

Include: <stdlib.h>

Prototype: char *getenv(const char *name);

Argument: name name of environment variable

Return Value: Returns a pointer to the value of the environment variable if successful; otherwise, returns a null pointer.

Remarks: This function must be customized to be used as described (see pic30-libs). By default there are no entries in the environment list for getenv to find.

16-Bit Language Tools Libraries

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

Standard C Libraries with Math Functions

ldiv

Description: Calculates the quotient and remainder of two long integers.

Include: <stdlib.h>

Prototype: `ldiv_t ldiv(long numer, long denom);`

Arguments: *numer* numerator
denom denominator

Return Value: Returns the quotient and the remainder.

Remarks: The returned quotient will have the same sign as the numerator divided by the denominator. The sign for the remainder will be such that the quotient times the denominator plus the remainder will equal the numerator ($\text{quot} * \text{denom} + \text{rem} = \text{numer}$). If the denominator is zero, the behavior is undefined.

Example:

```
#include <stdlib.h> /* for ldiv, ldiv_t */
#include <stdio.h> /* for printf */

int main(void)
{
    long x,y;
    ldiv_t z;

    x = 7;
    y = 3;
    printf("For ldiv(%ld, %ld)\n", x, y);
    z = ldiv(x, y);
    printf("The quotient is %ld and the "
           "remainder is %ld\n\n", z.quot, z.rem);

    x = 7;
    y = -3;
    printf("For ldiv(%ld, %ld)\n", x, y);
    z = ldiv(x, y);
    printf("The quotient is %ld and the "
           "remainder is %ld\n\n", z.quot, z.rem);

    x = -5;
    y = 3;
    printf("For ldiv(%ld, %ld)\n", x, y);
    z = ldiv(x, y);
    printf("The quotient is %ld and the "
           "remainder is %ld\n\n", z.quot, z.rem);

    x = 7;
    y = 7;
    printf("For ldiv(%ld, %ld)\n", x, y);
    z = ldiv(x, y);
    printf("The quotient is %ld and the "
           "remainder is %ld\n\n", z.quot, z.rem);

    x = 7;
    y = 0;
    printf("For ldiv(%ld, %ld)\n", x, y);
    z = ldiv(x, y);
    printf("The quotient is %ld and the "
           "remainder is %ld\n\n",
           z.quot, z.rem);
}
```

16-Bit Language Tools Libraries

ldiv (Continued)

Output:

```
For ldiv(7, 3)
The quotient is 2 and the remainder is 1

For ldiv(7, -3)
The quotient is -2 and the remainder is 1

For ldiv(-5, 3)
The quotient is -1 and the remainder is -2

For ldiv(7, 7)
The quotient is 1 and the remainder is 0

For ldiv(7, 0)
The quotient is -1 and the remainder is 7
```

Explanation:

In the last example (`ldiv(7,0)`) the denominator is zero, the behavior is undefined.

malloc

Description: Allocates memory.

Include: `<stdlib.h>`

Prototype: `void *malloc(size_t size);`

Argument: `size` number of characters to allocate

Return Value: Returns a pointer to the allocated space if successful; otherwise, returns a null pointer.

Remarks: `malloc` does not initialize memory it returns.

Example:

```
#include <stdio.h> /* for printf, sizeof, */
                    /* NULL */
#include <stdlib.h> /* for malloc, free */

int main(void)
{
    long *i;

    if ((i = (long *)malloc(50 * sizeof(long))) ==
        NULL)
        printf("Cannot allocate memory\n");
    else
    {
        printf("Memory allocated\n");
        free(i);
        printf("Memory freed\n");
    }
}
```

Output:

```
Memory allocated
Memory freed
```

Standard C Libraries with Math Functions

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.

16-Bit Language Tools Libraries

qsort

Description: Performs a quick sort.

Include: <stdlib.h>

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

Arguments:

- base* pointer to the start of the array
- nelem* number of elements
- size* size of the elements
- cmp* pointer to the comparison function
- e1* pointer to the key for the search
- e2* pointer to the element being compared with the key

Remarks: qsort overwrites the array with the sorted array. The comparison function is supplied by the user. In the following example, the list is sorted according to the comparison function. This *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
```

Standard C Libraries with Math Functions

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:

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

int main(void)
{
    int x;

    for (x = 0; x < 5; x++)
        printf("Number = %d\n", rand());
}
```

Output:

```
Number = 21422
Number = 2061
Number = 16443
Number = 11617
Number = 9125
```

Notice if the program is run a second time, the numbers are the same. See the example for `srand` to seed the random number generator.

realloc

Description: Reallocates memory to allow a size change.

Include: `<stdlib.h>`

Prototype: `void *realloc(void *ptr, size_t size);`

Arguments: `ptr` points to previously allocated memory
`size` new size to allocate to

Return Value: Returns a pointer to the allocated space if successful; otherwise, returns a null pointer.

Remarks: If the existing object is smaller than the new object, the entire existing object is copied to the new object and the remainder of the new object is indeterminate. If the existing object is larger than the new object, the function copies as much of the existing object as will fit in the new object. If `realloc` succeeds in allocating a new object, the existing object will be deallocated; otherwise, the existing object is left unchanged. Keep a temporary pointer to the existing object since `realloc` will return a null pointer on failure.

16-Bit Language Tools Libraries

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
```

Standard C Libraries with Math Functions

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.

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.

strtod (Continued)

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 );

    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
```

Standard C Libraries with Math Functions

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 );
}
```

Output:

```
String = "-12BGEE" long = -299
Stopped at: GEE
```

```
String = "1234Number" long = 27
Stopped at: 4Number
```

16-Bit Language Tools Libraries

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 );
}
```

Output:

```
String = "12BGET3" long = 429164
Stopped at: T3
```

```
String = "0x1234Number" long = 4660
Stopped at: Number
```

```
String = "-123abc" long = 4294967173
Stopped at: abc
```

Standard C Libraries with Math Functions

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
wcs points to the wide character string
n the number of characters to convert

Return Value: Returns the number of characters stored excluding the null character.

Remarks: wcstombs converts n number of multibyte characters unless it encounters a null character first. MPLAB C30 does not support multibyte characters with length greater than 1 character.

16-Bit Language Tools Libraries

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 `sizeof` operator.
Include: `<string.h>`

NULL

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

memchr

Description: Locates a character in a buffer.
Include: `<string.h>`
Prototype: `void *memchr(const void *s, int c, size_t n);`
Arguments:
s pointer to the buffer
c character to search for
n number of characters to check
Return Value: Returns a pointer to the location of the match if successful; otherwise, returns null.
Remarks: `memchr` stops when it finds the first occurrence of *c* or after searching *n* number of characters.

Example:

```
#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;

    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);
}
```

memchr (Continued)

```
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:

- s1 first buffer
- s2 second buffer
- n number of characters to compare

Return Value: Returns a positive number if s1 is greater than s2, zero if s1 is equal to s2, or a negative number if s1 is less than s2.

Remarks: This function compares the first n characters in s1 to the first n characters in s2 and returns a value indicating whether the buffers are less than, equal to or greater than each other.

Example:

```
#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");
}
```

memcmp (Continued)

```
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
```

Standard C Libraries with Math Functions

memcpy

Description: Copies characters from one buffer to another.

Include: <string.h>

Prototype: void *memcpy(void *dst , const void *src , size_t n);

Arguments: *dst* buffer to copy characters to
src buffer to copy characters from
n number of characters to copy

Return Value: Returns *dst*.

Remarks: memcpy copies *n* characters from the source buffer *src* to the destination buffer *dst*. If the buffers overlap, the behavior is undefined.

Example:

```
#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);
}
```

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?
```

16-Bit Language Tools Libraries

memmove

Description: Copies *n* characters of the source buffer into the destination buffer, even if the regions overlap.

Include: `<string.h>`

Prototype: `void *memmove(void *s1, const void *s2, size_t n);`

Arguments:

- s1* buffer to copy characters to (destination)
- s2* buffer to copy characters from (source)
- n* number of characters to copy from *s2* to *s1*

Return Value: Returns a pointer to the destination buffer

Remarks: If the buffers overlap, the effect is as if the characters are read first from *s2* then written to *s1* so the buffer is not corrupted.

Example:

```
#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?
```

Standard C Libraries with Math Functions

memset

Description: Copies the specified character into the destination buffer.

Include: <string.h>

Prototype: void *memset(void *s, int c, size_t n);

Arguments:
s buffer
c character to put in buffer
n number of times

Return Value: Returns the buffer with characters written to it.

Remarks: The character *c* is written to the buffer *n* times.

Example:
#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
```

16-Bit Language Tools Libraries

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(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)
```

Standard C Libraries with Math Functions

strchr

Description:	Locates the first occurrence of a specified character in a string.
Include:	<string.h>
Prototype:	char *strchr(const char *s, int c);
Arguments:	s pointer to the string c character to search for
Return Value:	Returns a pointer to the location of the match if successful; otherwise, returns a null pointer.
Remarks:	This function searches the string s to find the first occurrence of the character c.
Example:	<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; 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); }</pre>

Output:
buf1 : What time is it?

m found at position 8

y not found

16-Bit Language Tools Libraries

strcmp

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

strcmp (Continued)

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:	s1 first string s2 second string
Return Value:	Using the locale-dependent rules, it returns a positive number if s1 is greater than s2, zero if s1 is equal to s2, or a negative number if s1 is less than s2.
Remarks:	Since MPLAB C30 does not support alternate locales, this function is equivalent to strcmp.

strcpy

Description:	Copy the source string into the destination string.
Include:	<string.h>
Prototype:	char *strcpy(char *s1, const char *s2);
Arguments:	s1 destination string to copy to s2 source string to copy from
Return Value:	Returns a pointer to the destination string.
Remarks:	All characters of s2 are copied, including the null terminating character. If the strings overlap, the behavior is undefined.
Example:	<pre>#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);</pre>

strcpy (Continued)

```
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);

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
```

Standard C Libraries with Math Functions

strcspn (Continued)

Explanation:

In the first result, *e* is in *s2* so it stops counting after *h*.

In the second result, *a* is in *s2*.

In the third result, none of the characters of *s1* are in *s2* so all characters are counted.

strerror

Description: Gets an internal error message.

Include: <string.h>

Prototype: char *strerror(int *errcode*);

Argument: *errcode* number of the error code

Return Value: Returns a pointer to an internal error message string corresponding to the specified error code *errcode*.

Remarks: The array pointed to by *strerror* may be overwritten by a subsequent call to this function.

Example:

```
#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.

16-Bit Language Tools Libraries

strlen (Continued)

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",
        strlen(str3));
}
```

Output:

```
str1 : We are here
      (string length = 11 characters)

str2 :
      (string length = 0 characters)

str3 : Why me?
      (string length = 7 characters)
```

strncat

Description: Append a specified number of characters from the source string to the destination string.

Include: <string.h>

Prototype: char *strncat(char *s1, const char *s2, size_t n);

Arguments:

- s1* destination string to copy to
- s2* source string to copy from
- n* number of characters to append

Return Value: Returns a pointer to the destination string.

Remarks: This function appends up to *n* characters (a null character and characters that follow it are not appended) from the source string to the end of the destination string. If a null character is not encountered, then a terminating null character is appended to the result. If the strings overlap, the behavior is undefined.

Example:

```
#include <string.h> /* for strncat, strlen */
#include <stdio.h> /* for printf */

int main(void)
{
    char buf1[50] = "We're here";
    char buf2[50] = "Where is the time?";
    char buf3[50] = "Why?";
```

strncat (Continued)

```
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));

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)
```

16-Bit Language Tools Libraries

strncmp

Description: Compare two strings, up to a specified number of characters.

Include: <string.h>

Prototype: int strncmp(const char *s1, const char *s2, size_t n);

Arguments: s1 first string
s2 second string
n number of characters to compare

Return Value: Returns a positive number if s1 is greater than s2, zero if s1 is equal to s2, or a negative number if s1 is less than s2.

Remarks: strncmp returns a value based on the first character that differs between s1 and s2. Characters that follow a null character are not compared.

Example:

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

int main(void)
{
    char buf1[50] = "Where is the time?";
    char buf2[50] = "Where did they go?";
    char buf3[50] = "Why?";
    int res;

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

    res = strncmp(buf1, buf2, 6);
    if (res < 0)
        printf("buf1 comes before buf2\n");
    else if (res == 0)
        printf("6 characters of buf1 and buf2 "
              "are equal\n");
    else
        printf("buf2 comes before buf1\n");

    printf("\n");

    res = strncmp(buf1, buf2, 20);
    if (res < 0)
        printf("buf1 comes before buf2\n");
    else if (res == 0)
        printf("20 characters of buf1 and buf2 "
              "are equal\n");
    else
        printf("buf2 comes before buf1\n");
}
```

strncmp (Continued)

```
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);
}
```

16-Bit Language Tools Libraries

strncpy (Continued)

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

printf("\n");

strncpy(buf1, buf2, 18);
printf("buf1 after strncpy of 18 characters "
      "of buf2: \n\t%s\n", buf1);
printf("\t( %d characters)\n", strlen(buf1));

printf("\n");

strncpy(buf1, buf3, 5);
printf("buf1 after strncpy of 5 characters "
      "of buf3: \n\t%s\n", buf1);
printf("\t( %d characters)\n", strlen(buf1));

printf("\n");

strncpy(buf1, buf4, 9);
printf("buf1 after strncpy of 9 characters "
      "of buf4: \n\t%s\n", buf1);
printf("\t( %d characters)\n", strlen(buf1));
}
```

Output:

```
buf1 : We're here
buf2 : Where is the time?
buf3 : Why?
buf4 : Where?
buf1 after strncpy of 6 characters of buf2:
      Where here
      ( 10 characters)

buf1 after strncpy of 18 characters of buf2:
      Where is the time?
      ( 18 characters)

buf1 after strncpy of 5 characters of buf3:
      Why?
      ( 4 characters)

buf1 after strncpy of 9 characters of buf4:
      Where?
      ( 6 characters)
```

strncpy (Continued)

Explanation:

Each buffer contains the string shown, followed by null characters for a length of 50. Using `strlen` will find the length of the string up to but not including the first null character.

In the first example, 6 characters of `buf2` ("Where ") replace the first 6 characters of `buf1` ("We're ") and the rest of `buf1` remains the same ("here" plus null characters).

In the second example, 18 characters replace the first 18 characters of `buf1` and the rest remain null characters.

In the third example, 5 characters of `buf3` ("Why?" plus a null terminating character) replace the first 5 characters of `buf1`. `buf1` now actually contains ("Why?", 1 null character, " is the time?", 32 null characters). `strlen` shows 4 characters because it stops when it reaches the first null character.

In the fourth example, since `buf4` is only 7 characters `strncpy` uses 2 additional null characters to replace the first 9 characters of `buf1`. The result of `buf1` is 6 characters ("Where?") followed by 3 null characters, followed by 9 characters ("the time?"), followed by 32 null characters.

strpbrk

Description: Search a string for the first occurrence of a character from a specified set of characters.

Include: `<string.h>`

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

Arguments: `s1` pointer to the string to be searched

`s2` pointer to characters to search for

Return Value: Returns a pointer to the matched character in `s1` if found; otherwise, returns a null pointer.

Remarks: This function will search `s1` for the first occurrence of a character contained in `s2`.

Example:

```
#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");
}
```

strupbrk (Continued)

```
printf("\n");

printf("strupbrk(\"%s\", \"%s\")\n", str1, str3);
ptr = strupbrk(str1, str3);
if (ptr != NULL)
{
    res = ptr - str1 + 1;
    printf("match found at position %d\n", res);
}
else
    printf("match not found\n");
}
```

Output:

```
strupbrk("What time is it?", "xyz")
match not found
```

```
strupbrk("What time is it?", "eou?")
match found at position 9
```

strrchr

Description: Search for the last occurrence of a specified character in a string.

Include: <string.h>

Prototype: char *strrchr(const char *s, int c);

Arguments: s pointer to the string to be searched
c character to search for

Return Value: Returns a pointer to the character if found; otherwise, returns a null pointer.

Remarks: The function searches the string s, including the terminating null character, to find the last occurrence of character c.

Example: #include <string.h> /* for strrchr, NULL */
#include <stdio.h> /* for printf */

```
int main(void)
{
    char buf1[50] = "What time is it?";
    char ch1 = 'm', ch2 = 'y';
    char *ptr;
    int res;

    printf("buf1 : %s\n\n", buf1);

    ptr = strrchr(buf1, ch1);
    if (ptr != NULL)
    {
        res = ptr - buf1 + 1;
        printf("%c found at position %d\n", ch1, res);
    }
    else
        printf("%c not found\n", ch1);
}
```

strchr (Continued)

```
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

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.

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] = "aeioum";
    char str3[20] = "aimnl";
    char str4[20] = "xyz";
    int res;

    res = strspn(str1, str2);
    printf("strspn(\"%s\", \"%s\") = %d\n",
           str1, str2, res);

    res = strspn(str1, str3);
    printf("strspn(\"%s\", \"%s\") = %d\n",
           str1, str3, res);

    res = strspn(str1, str4);
    printf("strspn(\"%s\", \"%s\") = %d\n",
           str1, str4, res);
}
```

strspn (Continued)

Output:

```
strspn("animal", "aeioum") = 5
strspn("animal", "aimnl") = 6
strspn("animal", "xyz") = 0
```

Explanation:

In the first result, l is not in *s2*.

In the second result, the terminating null is not in *s2*.

In the third result, a is not in *s2*, so the comparison stops.

strstr

Description: Search for the first occurrence of a string inside another string.

Include: <string.h>

Prototype: char *strstr(const char *s1, const char *s2);

Arguments: *s1* pointer to the string to be searched
s2 pointer to substring to be searched for

Return Value: Returns the address of the first element that matches the substring if found; otherwise, returns a null pointer.

Remarks: This function will find the first occurrence of the string *s2* (excluding the null terminator) within the string *s1*. If *s2* points to a zero length string, *s1* is returned.

Example:

```
#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("\n%s\" found at position %d\n",
            str2, res);
    }
    else
        printf("\n%s\" not found\n", str2);
}
```

strstr (Continued)

```
printf("\n");

ptr = strstr(str1, str3);
if (ptr != NULL)
{
    res = ptr - str1 + 1;
    printf("\n%s\" found at position %d\n",
          str3, res);
}
else
    printf("\n%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:	<i>s1</i> pointer to the null terminated string to be searched <i>s2</i> 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 <i>s1</i> that does not appear in the set of characters of <i>s2</i>). If no token is found, the null pointer is returned.
Remarks:	<p>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 <i>s1</i>. After the first time, this function can continue parsing the string from the last delimiter by invoking it with a null value passed in <i>s1</i>.</p> <p>It skips all leading characters that appear in the string <i>s2</i> (delimiters), then skips all characters not appearing in <i>s2</i> (this segment of characters is the token), and then overwrites the next character with a null character, terminating the current token. The function <code>strtok</code> then saves a pointer to the character that follows, from which the next search will start. If <code>strtok</code> finds the end of the string before it finds a delimiter, the current token extends to the end of the string pointed to by <i>s1</i>. If this is the first call to <code>strtok</code>, it does not modify the string (no null characters are written to <i>s1</i>). The set of characters that is passed in <i>s2</i> need not be the same for each call to <code>strtok</code>.</p> <p>If <code>strtok</code> is called with a non-null parameter for <i>s1</i> after the initial call, the string becomes the new string to search. The old string previously searched will be lost.</p>

16-Bit Language Tools Libraries

strtok (Continued)

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:

- `s1` destination string
- `s2` source string to be transformed
- `n` number of characters to transform

Return Value: Returns the length of the transformed string not including the terminating null character. If `n` is zero, the string is not transformed (`s1` may be a point null in this case) and the length of `s2` is returned.

Remarks: If the return value is greater than or equal to `n`, the content of `s1` is indeterminate. Since MPLAB C30 does not support alternate locales, the transformation is equivalent to `strcpy`, except that the length of the destination string is bounded by `n-1`.

Standard C Libraries with Math Functions

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 `sizeof` operator.
Include: `<time.h>`

struct tm

Description: Structure used to hold the time and date (calendar time).
Include: `<time.h>`
Prototype:

```
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 */
}
```

Remarks: If `tm_isdst` 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.

16-Bit Language Tools Libraries

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

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.

Standard C Libraries with Math Functions

clock (Continued)

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: tod pointer to stored time

Return Value: Returns the address of a string that represents the local time of the parameter passed.

Remarks: This function is equivalent to asctime(localtime(tod)).

Example:

```
#include <time.h> /* for mktime, tm, ctime */
#include <stdio.h> /* for printf */

int main(void)
{
    time_t whattime;
    struct tm nowtime;

    nowtime.tm_sec = 30;
    nowtime.tm_min = 30;
    nowtime.tm_hour = 2;
    nowtime.tm_mday = 1;
    nowtime.tm_mon = 1;
    nowtime.tm_year = 103;

    whattime = mktime(&nowtime);
    printf("Day and time %s\n", ctime(&whattime));
}

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

16-Bit Language Tools Libraries

difftime

Description: Find the difference between two times.

Include: `<time.h>`

Prototype: `double difftime(time_t t1, time_t t0);`

Arguments: `t1` ending time
`t0` beginning time

Return Value: Returns the number of seconds between `t1` and `t0`.

Remarks: By default, MPLAB C30 returns the time as instruction cycles so `difftime` returns the number of ticks between `t1` and `t0`.

Example:

```
#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.

Standard C Libraries with Math Functions

gmtime (Continued)

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: tod pointer to stored time

Return Value: Returns the address of the time structure.

Remarks: By default, MPLAB C30 returns the time as instruction cycles. With this default localtime and gmtime will be equivalent except localtime will return tm_isdst (Daylight Savings Time flag) as -1 to indicate that the status of Daylight Savings Time is not known.

Example:

```
#include <time.h> /* for localtime,      */
                    /* asctime, time_t, tm */
#include <stdio.h> /* for printf          */

int main(void)
{
    time_t timer;
    struct tm *newtime;

    timer = 1066668182; /* Mon Oct 20 16:43:02 2003 */

    newtime = localtime(&timer);
    printf("Local time = %s\n", asctime(newtime));
}

Output:
Local time = Mon Oct 20 16:43:02 2003
```

16-Bit Language Tools Libraries

mktime

Description: Converts local time to a calendar value.

Include: `<time.h>`

Prototype: `time_t mktime(struct tm *tptr);`

Argument: `tptr` a pointer to the time structure

Return Value: Returns the calendar time encoded as a value of `time_t`.

Remarks: If the calendar time cannot be represented, the function returns -1, cast as a `time_t` (i.e. `(time_t) -1`).

Example:

```
#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:

- `s` output string
- `n` maximum length of string
- `format` format-control string
- `tptr` pointer to `tm` data structure

Return Value: Returns the number of characters placed in the array `s` if the total including the terminating null is not greater than `n`. Otherwise, the function returns 0 and the contents of array `s` are indeterminate.

Remarks: The format parameters follow:

- %a** abbreviated weekday name
- %A** full weekday name
- %b** abbreviated month name
- %B** full month name
- %c** appropriate date and time representation
- %d** day of the month (01-31)
- %H** hour of the day (00-23)

Standard C Libraries with Math Functions

strftime (Continued)

%l 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
%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 = 1066668182; /* 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.
```

16-Bit Language Tools Libraries

time

Description: Calculates the current calendar time.

Include: `<time.h>`

Prototype: `time_t time(time_t *tod);`

Argument: `tod` pointer to storage location for time

Return Value: Returns the calendar time encoded as a value of `time_t`.

Remarks: If the target environment cannot determine the time, the function returns -1, cast as a `time_t`. By default, MPLAB C30 returns the time as instruction cycles. This function is customizable. See `pic30-libs`.

Example:

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

volatile int i;

int main(void)
{
    time_t ticks;

    time(0); /* start time */
    for (i = 0; i < 10; i++) /* waste time */
        time(&ticks); /* get time */
    printf("Time = %ld\n", ticks);
}
```

Output:

Time = 256

4.17 <MATH.H> MATHEMATICAL FUNCTIONS

The header file `math.h` consists of a macro and various functions that calculate common mathematical operations. Error conditions may be handled with a domain error or range error (see `errno.h`).

A domain error occurs when the input argument is outside the domain over which the function is defined. The error is reported by storing the value of `EDOM` in `errno` and returning a particular value defined for each function.

A range error occurs when the result is too large or too small to be represented in the target precision. The error is reported by storing the value of `ERANGE` in `errno` and returning `HUGE_VAL` if the result overflowed (return value was too large) or a zero if the result underflowed (return value is too small).

Responses to special values, such as NaNs, zeros, and infinities, may vary depending upon the function. Each function description includes a definition of the function's response to such values.

HUGE_VAL

Description: `HUGE_VAL` is returned by a function on a range error (e.g., the function tries to return a value too large to be represented in the target precision).

Include: `<math.h>`

Remarks: `-HUGE_VAL` is returned if a function result is negative and is too large (in magnitude) to be represented in the target precision. When the printed result is `+/- HUGE_VAL`, it will be represented by `+/- inf`.

acos

Description: Calculates the trigonometric arc cosine function of a double precision floating-point value.

Include: `<math.h>`

Prototype: `double acos (double x);`

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

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

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

Example:

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

int main(void)
{
    double x,y;

    errno = 0;
    x = -2.0;
    y = acos (x);
    if (errno)
        perror("Error");
    printf("The arccosine of %f is %f\n\n", x, y);
}
```

acos (Continued)

```
    errno = 0;
    x = 0.10;
    y = acos (x);
    if (errno)
        perror("Error");
    printf("The arccosine of %f is %f\n\n", x, y);
}
```

Output:

Error: domain error

The arccosine of -2.000000 is nan

The arccosine of 0.100000 is 1.470629

acosf

Description: Calculates the trigonometric arc cosine function of a single precision floating-point value.

Include: <math.h>

Prototype: float acosf (float x);

Argument: x value between -1 and 1

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

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

Example:

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

int main(void)
{
    float x, y;

    errno = 0;
    x = 2.0F;
    y = acosf (x);
    if (errno)
        perror("Error");
    printf("The arccosine of %f is %f\n\n", x, y);

    errno = 0;
    x = 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

Standard C Libraries with Math Functions

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 arc sine

Return Value: Returns the arc sine in radians in the range of -pi/2 to +pi/2 (inclusive).

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

Example:

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

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

    errno = 0;
    x = 2.0;
    y = asin (x);
    if (errno)
        perror("Error");
    printf("The arcsine of %f is %f\n\n", x, y);

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

Output:

```
Error: domain error
The arcsine of 2.000000 is nan

The arcsine of 0.000000 is 0.000000
```

asinf

Description: Calculates the trigonometric arc sine function of a single precision floating-point value.

Include: <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;
```

asin (Continued)

```
    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 arc tangent

Return Value: Returns the arc tangent in radians in the range of $-\pi/2$ to $+\pi/2$ (inclusive).

Remarks: No domain or range error will occur.

Example: #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
```

Standard C Libraries with Math Functions

atanf

Description: Calculates the trigonometric arc tangent function of a single precision floating-point value.

Include: <math.h>

Prototype: float atanf (float x);

Argument: x value for which to return the arc tangent

Return Value: Returns the arc tangent in radians in the range of $-\pi/2$ to $+\pi/2$ (inclusive).

Remarks: No domain or range error will occur.

Example: #include <math.h> /* for atanf */
#include <stdio.h> /* for printf */

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

    x = 2.0F;
    y = atanf (x);
    printf("The arctangent of %f is %f\n\n", x, y);

    x = -1.0F;
    y = atanf (x);
    printf("The arctangent of %f is %f\n\n", x, y);
}
```

Output:

The arctangent of 2.000000 is 1.107149

The arctangent of -1.000000 is -0.785398

atan2

Description: Calculates the trigonometric arc tangent function of y/x .

Include: <math.h>

Prototype: double atan2 (double y, double x);

Arguments: y y value for which to return the arc tangent

x x value for which to return the arc tangent

Return Value: Returns the arc tangent in radians in the range of $-\pi$ to 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 */

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

atan2 (Continued)

```
    errno = 0;
    x = 0.0;
    y = 2.0;
    z = atan2(y, x);
    if (errno)
        perror("Error");
    printf("The arctangent of %f/%f is %f\n\n",
           y, x, z);

    errno = 0;
    x = -1.0;
    y = 0.0;
    z = atan2(y, x);
    if (errno)
        perror("Error");
    printf("The arctangent of %f/%f is %f\n\n",
           y, x, z);

    errno = 0;
    x = 0.0;
    y = 0.0;
    z = atan2(y, x);
    if (errno)
        perror("Error");
    printf("The arctangent of %f/%f is %f\n\n",
           y, x, z);
}
```

Output:

The arctangent of 2.000000/0.000000 is 1.570796

The arctangent of 0.000000/-1.000000 is 3.141593

Error: domain error

The arctangent of 0.000000/0.000000 is nan

Standard C Libraries with Math Functions

atan2f

Description: Calculates the trigonometric arc tangent function of y/x .

Include: `<math.h>`

Prototype: `float atan2f (float y, float x);`

Arguments:
`y` y value for which to return the arc tangent
`x` x value for which to return the arc tangent

Return Value: Returns the arc tangent in radians in the range of $-\pi$ to π 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 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);

    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

16-Bit Language Tools Libraries

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
```

Standard C Libraries with Math Functions

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:

```
#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);
    }
}
```

Output:

```
The ceiling for 2.000000 is 2.000000
The ceiling for 1.750000 is 2.000000
The ceiling for 1.500000 is 2.000000
The ceiling for 1.250000 is 2.000000
The ceiling for -2.000000 is -2.000000
The ceiling for -1.750000 is -1.000000
The ceiling for -1.500000 is -1.000000
The ceiling for -1.250000 is -1.000000
```

COS

Description: Calculates the trigonometric cosine function of a double precision floating-point value.

Include: <math.h>

Prototype: double cos (double x);

Argument: x value for which to return the cosine

Return Value: Returns the cosine of x in radians in the ranges of -1 to 1 inclusive.

Remarks: A domain error will occur if x is a NaN or infinity.

Example:

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

int main(void)
{
    double x,y;

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

16-Bit Language Tools Libraries

cos (Continued)

```
    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);
}
```

Output:

The cosine of -1.000000 is 0.540302

The cosine of 0.000000 is 1.000000

Standard C Libraries with Math Functions

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

16-Bit Language Tools Libraries

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

Standard C Libraries with Math Functions

exp

Description: Calculates the exponential function of x (e raised to the power x where x is a double precision floating-point value).

Include: <math.h>

Prototype: double exp (double x);

Argument: x value for which to return the exponential

Return Value: Returns the exponential of x . On an overflow, exp returns inf and on an underflow exp returns 0.

Remarks: A range error occurs if the magnitude of x is too large.

Example:

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

int main(void)
{
    double x, y;

    errno = 0;
    x = 1.0;
    y = exp (x);
    if (errno)
        perror("Error");
    printf("The exponential of %f is %f\n\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);
}
```

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

16-Bit Language Tools Libraries

expf

Description: Calculates the exponential function of x (e raised to the power x where x is a single precision floating-point value).

Include: <math.h>

Prototype: float expf (float x);

Argument: x floating-point value for which to return the exponential

Return Value: Returns the exponential of x . On an overflow, expf returns inf and on an underflow exp returns 0.

Remarks: A range error occurs if the magnitude of x is too large.

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

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

    errno = 0;
    x = 1.0F;
    y = expf (x);
    if (errno)
        perror("Error");
    printf("The exponential of %f is %f\n\n", x, y);

    errno = 0;
    x = 1.0E3F;
    y = expf (x);
    if (errno)
        perror("Error");
    printf("The exponential of %f is %f\n\n", x, y);

    errno = 0;
    x = -1.0E3F;
    y = expf (x);
    if (errno)
        perror("Error");
    printf("The exponential of %f is %f\n\n", x, y);
}
```

Output:

The exponential of 1.000000 is 2.718282

Error: range error

The exponential of 1000.000000 is inf

Error: range error

The exponential of -1000.000000 is 0.000000

Standard C Libraries with Math Functions

fabs

Description: Calculates the absolute value of a double precision floating-point value.

Include: <math.h>

Prototype: double fabs(double x);

Argument: x floating-point value for which to return the absolute value

Return Value: Returns the absolute value of x. (A negative number is returned as positive, a positive number is unchanged.)

Remarks: No domain or range error will occur.

Example:

```
#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
```

16-Bit Language Tools Libraries

floor

Description: Calculates the floor of a double precision floating-point value.

Include: `<math.h>`

Prototype: `double floor (double x);`

Argument: `x` floating-point value for which to return the floor.

Return Value: Returns the largest integer value less than or equal to `x`.

Remarks: No domain or range error will occur. See `ceil`.

Example:

```
#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`.

Standard C Libraries with Math Functions

floorf (Continued)

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);
    }
}
```

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 non-zero, the result will have the same sign as x and the magnitude of the result will be less than the magnitude of y .

Example:

```
#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);
}
```

16-Bit Language Tools Libraries

fmod (Continued)

```
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);
```

```
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
```

Standard C Libraries with Math Functions

fmodf

Description: Calculates the remainder of x/y as a single precision value.

Include: `<math.h>`

Prototype: `float fmodf(float x, float y);`

Arguments:
`x` a single precision floating-point value
`y` a single precision floating-point value

Return Value: Returns the remainder of x divided by y .

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

Example:

```
#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);
```

16-Bit Language Tools Libraries

fmodf (Continued)

```
    errno = 0;
    x = 7.0F;
    y = 0.0F;
    z = fmodf (x, y);
    if (errno)
        perror("Error");
    printf("For fmodf (%f, %f) the remainder is"
        " %f\n\n", x, y, z);
```

```
    errno = 0;
    x = 7.0F;
    y = 7.0F;
    z = fmodf (x, y);
    if (errno)
        perror("Error");
    printf("For fmodf (%f, %f) the remainder is"
        " %f\n\n", x, y, z);
```

```
}
```

Output:

```
For fmodf (7.000000, 3.000000) the remainder is 1.000000

For fmodf (-5.000000, 3.000000) the remainder is -2.000000

For fmodf (5.000000, -3.000000) the remainder is 2.000000

For fmodf (5.000000, -5.000000) the remainder is 0.000000

Error: domain error
For fmodf (7.000000, 0.000000) the remainder is nan

For fmodf (7.000000, 7.000000) the remainder is 0.000000
```

frexp

Description: Gets the fraction and the exponent of a double precision floating-point number.

Include: <math.h>

Prototype: double frexp (double x, int *exp);

Arguments: x floating-point value for which to return the fraction and exponent

*exp pointer to a stored integer exponent

Return Value: Returns the fraction, *exp* points to the exponent. If *x* is 0, the function returns 0 for both the fraction and exponent.

Remarks: The absolute value of the fraction is in the range of 1/2 (inclusive) to 1 (exclusive). No domain or range error will occur.

Example: #include <math.h> /* for frexp */
#include <stdio.h> /* for printf */

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

frexp (Continued)

```
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);

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: 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 frexpf */
#include <stdio.h> /* for printf */

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

16-Bit Language Tools Libraries

frexpf (Continued)

```
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);

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;
```

Standard C Libraries with Math Functions

ldexp (Continued)

```
    errno = 0;
    x = -0.625;
    n = 2;
    y = ldexp (x, n);
    if (errno)
        perror("Error");
    printf("For a number = %f and an exponent = %d\n",
           x, n);
    printf("  ldexp(%f, %d) = %f\n\n",
           x, n, y);

    errno = 0;
    x = 2.5;
    n = 3;
    y = ldexp (x, n);
    if (errno)
        perror("Error");
    printf("For a number = %f and an exponent = %d\n",
           x, n);
    printf("  ldexp(%f, %d) = %f\n\n",
           x, n, y);

    errno = 0;
    x = 15.0;
    n = 10000;
    y = ldexp (x, n);
    if (errno)
        perror("Error");
    printf("For a number = %f and an exponent = %d\n",
           x, n);
    printf("  ldexp(%f, %d) = %f\n\n",
           x, n, y);
}
```

Output:

```
For a number = -0.625000 and an exponent = 2
  ldexp(-0.625000, 2) = -2.500000
```

```
For a number = 2.500000 and an exponent = 3
  ldexp(2.500000, 3) = 20.000000
```

```
Error: range error
For a number = 15.000000 and an exponent = 10000
  ldexp(15.000000, 10000) = inf
```

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.

16-Bit Language Tools Libraries

ldexpf (Continued)

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

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

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

    errno = 0;
    x = -0.625F;
    n = 2;
    y = ldexpf (x, n);
    if (errno)
        perror("Error");
    printf("For a number = %f and an exponent = %d\n",
           x, n);
    printf(" ldexpf(%f, %d) = %f\n\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
```

Standard C Libraries with Math Functions

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 $x \leq 0$.

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

16-Bit Language Tools Libraries

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 $x \leq 0$.

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

Standard C Libraries with Math Functions

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 $x \leq 0$.

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

16-Bit Language Tools Libraries

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 $x \leq 0$.

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
```

Standard C Libraries with Math Functions

modf

Description: Splits a double precision floating-point value into fractional and integer parts.

Include: `<math.h>`

Prototype: `double modf(double x, double *pint);`

Arguments: `x` double precision floating-point value
`pint` pointer to a stored the integer part

Return Value: Returns the signed fractional part and `pint` points to the integer part.

Remarks: The absolute value of the fractional part is in the range of 0 (inclusive) to 1 (exclusive). No domain or range error will occur.

Example:

```
#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
```

16-Bit Language Tools Libraries

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);
}
```

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
```

Standard C Libraries with Math Functions

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);

    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
```

16-Bit Language Tools Libraries

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);

    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
```

Standard C Libraries with Math Functions

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

16-Bit Language Tools Libraries

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

Standard C Libraries with Math Functions

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.

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

16-Bit Language Tools Libraries

sinhf

Description: Calculates the hyperbolic sine function of a single precision floating-point value.

Include: <math.h>

Prototype: float sinhf (float x);

Argument: x value for which to return the hyperbolic sine

Return Value: Returns the hyperbolic sine of x

Remarks: A range error will occur if the magnitude of x is too large.

Example:

```
#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

Standard C Libraries with Math Functions

sqrt

Description: Calculates the square root of a double precision floating-point value.

Include: <math.h>

Prototype: double sqrt(double x);

Argument: x a non-negative floating-point value

Return Value: Returns the non-negative square root of x..

Remarks: If x is negative, a domain error occurs.

Example:

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

int main(void)
{
    double x, y;

    errno = 0;
    x = 0.0;
    y = sqrt (x);
    if (errno)
        perror("Error");
    printf("The square root of %f is %f\n\n", x, y);

    errno = 0;
    x = 9.5;
    y = sqrt (x);
    if (errno)
        perror("Error");
    printf("The square root of %f is %f\n\n", x, y);

    errno = 0;
    x = -25.0;
    y = sqrt (x);
    if (errno)
        perror("Error");
    printf("The square root of %f is %f\n\n", x, y);
}
```

Output:

The square root of 0.000000 is 0.000000

The square root of 9.500000 is 3.082207

Error: domain error

The square root of -25.000000 is nan

16-Bit Language Tools Libraries

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);

    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

Standard C Libraries with Math Functions

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;
```

tanf (Continued)

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

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

Output:

The tangent of -1.000000 is -1.557408

The tangent of 0.000000 is 0.000000

tanh

Description: Calculates the hyperbolic tangent function of a double precision floating-point value.

Include: <math.h>

Prototype: double tanh (double x);

Argument: x value for which to return the hyperbolic tangent

Return Value: Returns the hyperbolic tangent of x in the ranges of -1 to 1 inclusive.

Remarks: No domain or range error will occur.

Example:

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

int main(void)
{
    double x, y;

    x = -1.0;
    y = tanh (x);
    printf("The hyperbolic tangent of %f is %f\n\n",
        x, y);

    x = 2.0;
    y = tanh (x);
    printf("The hyperbolic tangent of %f is %f\n\n",
        x, y);
}
```

Output:

The hyperbolic tangent of -1.000000 is -0.761594

The hyperbolic tangent of 2.000000 is 0.964028

Standard C Libraries with Math Functions

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

16-Bit Language Tools Libraries

4.18 PIC30-LIBS

The following functions are standard C library helper functions:

- `_exit` terminate program execution
- `brk` set the end of the process's data space
- `close` close a file
- `lseek` move a file pointer to a specified location
- `open` open a file
- `read` read data from a file
- `sbrk` extend the process's data space by a given increment
- `write` write data to a file

These functions are called by other functions in the standard C library and must be modified for the target application. The corresponding object modules are distributed in the `libpic30-omf.a` archive and the source code (for MPLAB C30) is available in the `src\pic30` folder.

Additionally, several standard C library functions must also be modified for the target application. They are:

- `getenv` get a value for an environment variable
- `remove` remove a file
- `rename` rename a file or directory
- `system` execute a command
- `time` get the system time

Although these functions are part of the standard C library, the object modules are distributed in the `libpic30-omf.a` archive and the source code (for MPLAB C30) is available in the `src\pic30` folder. These modules are not distributed as part of `libc-omf.a`.

4.18.1 Rebuilding the `libpic30-omf.a` library

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

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

Standard C Libraries with Math Functions

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 <i>status</i>);</code>
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 <code>stdout</code> and terminates. The parameter <code>status</code> 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 *<i>endds</i>)</code>
Argument:	<i>endds</i> pointer to the end of the data segment
Return Value:	Returns '0' if successful, '-1' if not.
Remarks:	<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. Newly allocated space is uninitialized. This helper function is used by the Standard C Library function <code>malloc()</code> .

brk (Continued)

Default Behavior: If the argument *ends* is zero, the function sets the global variable `__curbrk` to the address of the start of the heap, and returns zero. If the argument *ends* 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 *ends* and returns zero. Otherwise, the global variable `__curbrk` is unchanged, and the function returns -1. The argument *ends* 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`

Standard C Libraries with Math Functions

getenv

Description:	Get a value for an environment variable
Include:	<stdlib.h>
Prototype:	char *getenv(const char *s);
Argument:	s name of environment variable
Return Value:	Returns a pointer to the value of the environment variable if successful; otherwise, returns a null pointer.
Default Behavior:	As distributed, this function returns a null pointer. There is no support for environment variables.
File:	getenv.c

lseek

Description:	Move a file pointer to a specified location.
Include:	None
Prototype:	long lseek(int handle, long offset, int origin);
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 <i>stdio.h</i>): 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 <i>fgetpos()</i> , <i>ftell()</i> , <i>fseek()</i> , <i>fsetpos()</i> , and <i>rewind()</i> .
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

16-Bit Language Tools Libraries

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.
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>

Standard C Libraries with Math Functions

read

Description:	Read data from a file.
Include:	None
Prototype:	<pre>int read(int handle, void * buffer, unsigned int len);</pre>
Argument:	<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>
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:	<pre>int rename(const char *oldname, const char *newname);</pre>
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 non-zero value.
Default Behavior:	As distributed, the parameters are passed to the host file system through the simulator. The return value is the value returned by the host file system.
File:	<code>rename.c</code>

16-Bit Language Tools Libraries

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>
Default Behavior:	<p>If the global variable <code>__curbrk</code> is zero, the function calls <code>brk()</code> to initialize the break value. If <code>brk()</code> returns -1, so does this function.</p> <p>If the <i>incr</i> is zero, the current value of the global variable <code>__curbrk</code> is returned.</p> <p>If the <i>incr</i> is non-zero, the function checks that the address (<code>__curbrk + incr</code>) is less than the end address of the heap. If it is less, the global variable <code>__curbrk</code> is updated to that value, and the function returns the unsigned value of <code>__curbrk</code>.</p> <p>Otherwise, the function returns -1.</p> <p>See the description of <code>brk()</code>.</p>
File:	<code>sbrk.c</code>

system

Description:	Execute a command.
Include:	<code><stdlib.h></code>
Prototype:	<code>int system(const char *s);</code>
Argument:	<i>s</i> command to be executed.
Default Behavior:	As distributed, this function acts as a stub or placeholder for your function. If <i>s</i> is not NULL, an error message is written to stdout and the program will reset; otherwise, a value of -1 is returned.
File:	<code>system.c</code>

Standard C Libraries with Math Functions

time

Description:	Get the system time.
Include:	<time.h>
Prototype:	<code>time_t time(time_t *timer);</code>
Argument:	<i>timer</i> 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:	<code>int write(int handle, void *buffer, unsigned int count);</code>
Argument:	<i>handle</i> refers to an opened file <i>buffer</i> points to the storage location of data to be written <i>count</i> the number of characters to write.
Return Value:	If successful, write returns the number of characters actually written. A return value of '-1' indicates an error.
Remarks:	If the actual space remaining on the disk is less than the size of the buffer the function is trying to write to the disk, write fails and does not flush any of the buffer's contents to the disk. If the file is opened in text mode, each linefeed character is replaced with a carriage return – linefeed pair in the output. The replacement does not affect the return value. This is a helper function called by the Standard C Library function <code>fflush()</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:	write.c

16-Bit Language Tools Libraries

NOTES:

Chapter 5. MPLAB C30 Built-in Functions

5.1 INTRODUCTION

This chapter describes the MPLAB C30 built-in functions that are specific to 16-bit devices.

Built-in functions give the C programmer access to assembler operators or machine instructions that are currently only accessible using inline assembly, but are sufficiently useful that they are applicable to a broad range of applications. Built-in functions are coded in C source files syntactically like function calls, but they are compiled to assembly code that directly implements the function, and do not involve function calls or library routines.

There are a number of reasons why providing built-in functions is preferable to requiring programmers to use inline assembly. They include the following:

1. Providing built-in functions for specific purposes simplifies coding.
2. Certain optimizations are disabled when inline assembly is used. This is not the case for built-in functions.
3. For machine instructions that use dedicated registers, coding inline assembly while avoiding register allocation errors can require considerable care. The built-in functions make this process simpler as you do not need to be concerned with the particular register requirements for each individual machine instruction.

This chapter is organized as follows:

- Built-In Function List

16-Bit Language Tools Libraries

5.2 BUILT-IN FUNCTION LIST

This section describes the programmer interface to the MPLAB C30 C compiler built-in functions. Since the functions are “built in”, there are no header files associated with them. Similarly, there are no command-line switches associated with the built-in functions – they are always available. The built-in function names are chosen such that they belong to the compiler's namespace (they all have the prefix `__builtin_`), so they will not conflict with function or variable names in the programmer's namespace.

`__builtin_addab`

Description: Add accumulators A and B with the result written back to the specified accumulator. For example:

```
register int result asm("A");
result = __builtin_addab();
```

will generate:

```
add A
```

Prototype: `int __builtin_addab(void);`

Argument: None

Return Value: Returns the addition result to an accumulator.

Assembler `addad`

Operator / Machine Instruction:

Error Messages An error message will be displayed if the result is not an accumulator register.

`__builtin_add`

Description: Add *value* to the accumulator specified by *result* with a shift specified by literal shift. For example:

```
register int result asm("A");
int value;
result = __builtin_add(value,0);
```

If *value* is held in *w0*, the following will be generated:

```
add w0, #0, A
```

Prototype: `int __builtin_add(int value, const int shift);`

Argument: *value* Integer number to add to accumulator value.
shift Amount to shift resultant accumulator value.

Return Value: Returns the shifted addition result to an accumulator.

Assembler `add`

Operator / Machine Instruction:

Error Messages An error message will be displayed if:

- the result is not an accumulator register
- the shift value is not a literal within range

__builtin_btg

Description: This function will generate a btg machine instruction. Some examples include:

```
int i;    /* near by default */
int l __attribute__((far));

struct foo {
    int bit1:1;
} barbits;

int bar;

void some_bittoggles() {
    register int j asm("w9");
    int k;

    k = i;

    __builtin_btg(&bar, barbits.bit1);
    __builtin_btg(&i, 1);
    __builtin_btg(&j, 3);
    __builtin_btg(&k, 4);
    __builtin_btg(&l, 11);

    return j+k;
}
```

Note that taking the address of a variable in a register will produce warning by the compiler and cause the register to be saved onto the stack (so that its address may be taken); this form is not recommended. This caution only applies to variables explicitly placed in registers by the programmer.

Prototype: void __builtin_btg(unsigned int *, unsigned int 0xn);

Argument: * A pointer to the data item for which a bit should be toggled.
0xn A literal value in the range of 0 to 15. As a convenience, it is possible to pass a bit-field name as this argument. The builtin will substitute the bit position of the identified field for the argument and toggle the appropriate bit.

Return Value: Returns a btg machine instruction.

Assembler btg

Operator / Machine Instruction:

Error Messages An error message will be displayed if the parameter values are not within range

__builtin_clr

Description: Clear the specified accumulator. For example:

```
register int result asm("A");
result = __builtin_clr();
```

will generate:

```
clr A
```

Prototype: int __builtin_clr(void);

Argument: None

16-Bit Language Tools Libraries

__builtin_clr (Continued)

Return Value:	Returns the cleared value result to an accumulator.
Assembler	clr
Operator / Machine	
Instruction:	
Error Messages	An error message will be displayed if the result is not an accumulator register.

__builtin_clr_prefetch

Description: Clear an accumulator and prefetch data ready for a future MAC operation.
xptr may be null to signify no X prefetch to be performed, in which case the values of *xincr* and *xval* are ignored, but required.
yptr may be null to signify no Y prefetch to be performed, in which case the values of *yincr* and *yval* are ignored, but required.
xval and *yval* nominate the address of a C variable where the prefetched value will be stored.
xincr and *yincr* may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.
If *AWB* is non null, the other accumulator will be written back into the referenced variable.

For example:

```
register int result asm("A");
int x_memory_buffer[256]
    __attribute__((space(xmemory)));
int y_memory_buffer[256]
    __attribute__((space(ymemory)));
int *xmemory;
int *ymemory;
int awb;
int xVal, yVal;

xmemory = x_memory_buffer;
ymemory = y_memory_buffer;
result = __builtin_clr(&xmemory, &xVal, 2,
    &ymemory, &yVal, 2, &awb);
```

might generate:

```
clr A, [w8]+=2, w4, [w10]+=2, w5, w13
```

The compiler may need to spill w13 to ensure that it is available for the write-back. It may be recommended to users that the register be claimed for this purpose.

After this instruction:

- result will be cleared
- xVal will contain `x_memory_buffer[0]`
- yVal will contain `y_memory_buffer[0]`
- `xmemory` and `ymemory` will be incremented by 2, ready for the next mac operation

Prototype:

```
int __builtin_clr_prefetch(
    int **xptr, int *xval, int xincr,
    int **yptr, int *yval, int yincr, int *AWB);
```

`__builtin_clr_prefetch` (Continued)

Argument:	<i>xptr</i>	Integer pointer to x prefetch.
	<i>xval</i>	Integer value of x prefetch.
	<i>xincr</i>	Integer increment value of x prefetch.
	<i>yptr</i>	Integer pointer to y prefetch.
	<i>yval</i>	Integer value of y prefetch.
	<i>yincr</i>	Integer increment value of y prefetch.
	<i>AWB</i>	Accumulator selection.
Return Value:	Returns the cleared value result to an accumulator.	
Assembler Operator / Machine Instruction:	<code>clr</code>	
Error Messages	An error message will be displayed if: <ul style="list-style-type: none">• the result is not an accumulator register• <i>xval</i> is a null value but <i>xptr</i> is not null• <i>yval</i> is a null value but <i>yptr</i> is not null	

`__builtin_divsd`

Description:	The function computes the quotient num / den . A math error exception occurs if <i>den</i> is zero. Function arguments are signed, as is the function result. The command-line option <code>-Wconversions</code> can be used to detect unexpected sign conversions.	
Prototype:	<code>int __builtin_divsd(const long num, const int den);</code>	
Argument:	<i>num</i>	numerator
	<i>den</i>	denominator
Return Value:	Returns the signed integer value of the quotient num / den .	
Assembler Operator / Machine Instruction:	<code>div.sd</code>	

`__builtin_divud`

Description:	The function computes the quotient num / den . A math error exception occurs if <i>den</i> is zero. Function arguments are unsigned, as is the function result. The command-line option <code>-Wconversions</code> can be used to detect unexpected sign conversions.	
Prototype:	<code>unsigned int __builtin_divud(const unsigned long num, const unsigned int den);</code>	
Argument:	<i>num</i>	numerator
	<i>den</i>	denominator
Return Value:	Returns the unsigned integer value of the quotient num / den .	
Assembler Operator / Machine Instruction:	<code>div.ud</code>	

16-Bit Language Tools Libraries

__builtin_dmaoffset

Description: Obtain the offset of a symbol within DMA memory.
For example:

```
int result;
char buffer[256] __attribute__((space(dma)));

result = __builtin_dmaoffset(buffer);
```

Might generate:

```
mov #dmaoffset(buffer), w0
```

Prototype: `int __builtin_dmaoffset(int buffer);`

Argument: *buffer* DMA address value

Return Value: Returns the offset to an accumulator.

Assembler Operator / Machine Instruction: `dmaoffset`

Error Messages An error message will be displayed if the result is not an accumulator register.

__builtin_ed

Description: Square *sqr*, returning it as the result. Also prefetch data for future square operation by computing `**xptr - **yptr` and storing the result in *distance*.
xincr and *yincr* may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.
For example:

```
register int result asm("A");
int *xmemory, *ymemory;
int distance;

result = __builtin_ed(distance,
                      &xmemory, 2,
                      &ymemory, 2,
                      &distance);
```

might generate:

```
ed w4*w4, A, [w8] += 2, [W10] += 2, w4
```

Prototype: `int __builtin_ed(int sqr, int **xptr, int xincr, int **yptr, int yincr, int *distance);`

Argument:

<i>sqr</i>	Integer squared value.
<i>xptr</i>	Integer pointer to pointer to x prefetch.
<i>xincr</i>	Integer increment value of x prefetch.
<i>yptr</i>	Integer pointer to pointer to y prefetch.
<i>yincr</i>	Integer increment value of y prefetch.
<i>distance</i>	Integer pointer to distance.

Return Value: Returns the squared result to an accumulator.

Assembler Operator / Machine Instruction: `ed`

__builtin_ed (Continued)

- Error Messages** An error message will be displayed if:
- the result is not an accumulator register
 - *xptr* is null
 - *yptr* is null
 - *distance* is null
-

__builtin_edac

Description: Square *sqr* and sum with the nominated accumulator register, returning it as the result. Also prefetch data for future square operation by computing ***xptr - **yptr* and storing the result in **distance*. *xincr* and *yincr* may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

For example:

```
register int result asm("A");
int *xmemory, *ymemory;
int distance;

result = __builtin_ed(distance,
                      &xmemory, 2,
                      &ymemory, 2,
                      &distance);
```

might generate:

```
ed w4*w4, A, [w8]+=2, [W10]+=2, w4
```

Prototype: `int __builtin_edac(int sqr, int **xptr, int xincr, int **yptr, int yincr, int *distance);`

Argument:

<i>sqr</i>	Integer squared value.
<i>xptr</i>	Integer pointer to pointer to x prefetch.
<i>xincr</i>	Integer increment value of x prefetch.
<i>yptr</i>	Integer pointer to pointer to y prefetch.
<i>yincr</i>	Integer increment value of y prefetch.
<i>distance</i>	Integer pointer to distance.

Return Value: Returns the squared result to specified accumulator.

Assembler edac

Operator / Machine Instruction:

- Error Messages** An error message will be displayed if:
- the result is not an accumulator register
 - *xptr* is null
 - *yptr* is null
 - *distance* is null
-

__builtin_fbcl

Description: Find the first bit change from left in value. This function is useful for dynamic scaling of fixed-point data. For example:

```
int result, value;
result = __builtin_fbcl(value);
```

might generate:

```
fbcl w4, w5
```

Prototype: `int __builtin_fbcl(int value);`

16-Bit Language Tools Libraries

__builtin_fbcl (Continued)

Argument:	<i>value</i> Integer number of first bit change.
Return Value:	Returns the shifted addition result to an accumulator.
Assembler Operator / Machine Instruction:	<i>fbcl</i>
Error Messages	An error message will be displayed if the result is not an accumulator register.

__builtin_lac

Description:	Shift value by <i>shift</i> (a literal between -8 and 7) and return the value to be stored into the accumulator register. For example: <pre>register int result asm("A"); int value; result = __builtin_lac(value,3);</pre>
	Might generate: <pre>lac w4, #3, A</pre>
Prototype:	<code>int __builtin_lac(int <i>value</i>, int <i>shift</i>);</code>
Argument:	<i>value</i> Integer number to be shifted. <i>shift</i> Literal amount to shift.
Return Value:	Returns the shifted addition result to an accumulator.
Assembler Operator / Machine Instruction:	<i>lac</i>
Error Messages	An error message will be displayed if: <ul style="list-style-type: none">• the result is not an accumulator register• the shift value is not a literal within range

`__builtin_mac`

Description:	<p>Compute $a \times b$ and sum with accumulator; also prefetch data ready for a future MAC operation.</p> <p><i>xptr</i> may be null to signify no X prefetch to be performed, in which case the values of <i>xincr</i> and <i>xval</i> are ignored, but required.</p> <p><i>yptr</i> may be null to signify no Y prefetch to be performed, in which case the values of <i>yincr</i> and <i>yval</i> are ignored, but required.</p> <p><i>xval</i> and <i>yval</i> nominate the address of a C variable where the prefetched value will be stored.</p> <p><i>xincr</i> and <i>yincr</i> may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.</p> <p>If <i>AWB</i> is non null, the other accumulator will be written back into the referenced variable.</p> <p>For example:</p> <pre>register int result asm("A"); int *xmemory; int *ymemory; int xVal, yVal; result = __builtin_mac(xVal, yVal, &xmemory, &xVal, 2, &ymemory, &yVal, 2, 0);</pre> <p>might generate:</p> <pre>mac w4*w5, A, [w8]+=2, w4, [w10]+=2, w5</pre>
Prototype:	<pre>int __builtin_mac(int a, int b, int **xptr, int *xval, int xincr, int **yptr, int *yval, int yincr, int *AWB);</pre>
Argument:	<p><i>a</i> Integer multiplicand.</p> <p><i>b</i> Integer multiplier.</p> <p><i>xptr</i> Integer pointer to pointer to x prefetch.</p> <p><i>xval</i> Integer pointer to value of x prefetch.</p> <p><i>xincr</i> Integer increment value of x prefetch.</p> <p><i>yptr</i> Integer pointer to pointer to y prefetch.</p> <p><i>yval</i> Integer pointer to value of y prefetch.</p> <p><i>yincr</i> Integer increment value of y prefetch.</p> <p><i>AWB</i> Integer pointer to accumulator selection.</p>
Return Value:	Returns the cleared value result to an accumulator.
Assembler Operator / Machine Instruction:	<code>mac</code>
Error Messages	<p>An error message will be displayed if:</p> <ul style="list-style-type: none">• the result is not an accumulator register• <i>xval</i> is a null value but <i>xptr</i> is not null• <i>yval</i> is a null value but <i>yptr</i> is not null

__builtin_movsac

Description:	<p>Compute nothing, but prefetch data ready for a future MAC operation. <i>xptr</i> may be null to signify no X prefetch to be performed, in which case the values of <i>xincr</i> and <i>xval</i> are ignored, but required. <i>yptr</i> may be null to signify no Y prefetch to be performed, in which case the values of <i>yincr</i> and <i>yval</i> are ignored, but required. <i>xval</i> and <i>yval</i> nominate the address of a C variable where the prefetched value will be stored. <i>xincr</i> and <i>yincr</i> may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value. If <i>AWB</i> is non null, the other accumulator will be written back into the referenced variable.</p> <p>For example:</p> <pre>register int result asm("A"); int *xmemory; int *ymemory; int xVal, yVal; result = __builtin_movsac(&xmemory, &xVal, 2, &ymemory, &yVal, 2, 0);</pre> <p>might generate:</p> <pre>movsac A, [w8]+=2, w4, [w10]+=2, w5</pre>														
Prototype:	<pre>int __builtin_movsac(int **xptr, int *xval, int xincr, int **yptr, int *yval, int yincr, int *AWB);</pre>														
Argument:	<table><tr><td><i>xptr</i></td><td>Integer pointer to pointer to x prefetch.</td></tr><tr><td><i>xval</i></td><td>Integer pointer to value of x prefetch.</td></tr><tr><td><i>xincr</i></td><td>Integer increment value of x prefetch.</td></tr><tr><td><i>yptr</i></td><td>Integer pointer to pointer to y prefetch.</td></tr><tr><td><i>yval</i></td><td>Integer pointer to value of y prefetch.</td></tr><tr><td><i>yincr</i></td><td>Integer increment value of y prefetch.</td></tr><tr><td><i>AWB</i></td><td>Integer pointer to accumulator selection.</td></tr></table>	<i>xptr</i>	Integer pointer to pointer to x prefetch.	<i>xval</i>	Integer pointer to value of x prefetch.	<i>xincr</i>	Integer increment value of x prefetch.	<i>yptr</i>	Integer pointer to pointer to y prefetch.	<i>yval</i>	Integer pointer to value of y prefetch.	<i>yincr</i>	Integer increment value of y prefetch.	<i>AWB</i>	Integer pointer to accumulator selection.
<i>xptr</i>	Integer pointer to pointer to x prefetch.														
<i>xval</i>	Integer pointer to value of x prefetch.														
<i>xincr</i>	Integer increment value of x prefetch.														
<i>yptr</i>	Integer pointer to pointer to y prefetch.														
<i>yval</i>	Integer pointer to value of y prefetch.														
<i>yincr</i>	Integer increment value of y prefetch.														
<i>AWB</i>	Integer pointer to accumulator selection.														
Return Value:	Returns prefetch data.														
Assembler Operator / Machine Instruction:	<i>movsac</i>														
Error Messages	<p>An error message will be displayed if:</p> <ul style="list-style-type: none">• the result is not an accumulator register• <i>xval</i> is a null value but <i>xptr</i> is not null• <i>yval</i> is a null value but <i>yptr</i> is not null														

`__builtin_mpy`

Description: compute $a \times b$; also prefetch data ready for a future MAC operation. *xptr* may be null to signify no X prefetch to be performed, in which case the values of *xincr* and *xval* are ignored, but required. *yptr* may be null to signify no Y prefetch to be performed, in which case the values of *yincr* and *yval* are ignored, but required. *xval* and *yval* nominate the address of a C variable where the prefetched value will be stored. *xincr* and *yincr* may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

For example:

```
register int result asm("A");
int *xmemory;
int *ymemory;
int xVal, yVal;

result = __builtin_mpy(xVal, yVal,
                      &xmemory, &xVal, 2,
                      &ymemory, &yVal, 2);
```

might generate:

```
mac w4*w5, A, [w8]+=2, w4, [w10]+=2, w5
```

Prototype:

```
int __builtin_mpy(int a, int b,
                 int **xptr, int *xval, int xincr,
                 int **yptr, int *yval, int yincr);
```

Argument:

<i>a</i>	Integer multiplicand.
<i>b</i>	Integer multiplier.
<i>xptr</i>	Integer pointer to pointer to x prefetch.
<i>xval</i>	Integer pointer to value of x prefetch.
<i>xincr</i>	Integer increment value of x prefetch.
<i>yptr</i>	Integer pointer to pointer to y prefetch.
<i>yval</i>	Integer pointer to value of y prefetch.
<i>yincr</i>	Integer increment value of y prefetch.
<i>AWB</i>	Integer pointer to accumulator selection.

Return Value: Returns the cleared value result to an accumulator.

Assembler `mpy`

Operator / Machine Instruction:

Error Messages An error message will be displayed if:

- the result is not an accumulator register
- *xval* is a null value but *xptr* is not null
- *yval* is a null value but *yptr* is not null

16-Bit Language Tools Libraries

__builtin_mpy

Description:	<p>compute $-a \times b$; also prefetch data ready for a future MAC operation. <i>xptr</i> may be null to signify no X prefetch to be performed, in which case the values of <i>xincr</i> and <i>xval</i> are ignored, but required. <i>yptr</i> may be null to signify no Y prefetch to be performed, in which case the values of <i>yincr</i> and <i>yval</i> are ignored, but required. <i>xval</i> and <i>yval</i> nominate the address of a C variable where the prefetched value will be stored. <i>xincr</i> and <i>yincr</i> may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.</p> <p>For example:</p> <pre>register int result asm("A"); int *xmemory; int *ymemory; int xVal, yVal; result = __builtin_mpy(xVal, yVal, &xmemory, &xVal, 2, &ymemory, &yVal, 2);</pre> <p>might generate:</p> <pre>mac w4*w5, A, [w8]+=2, w4, [w10]+=2, w5</pre>
Prototype:	<pre>int __builtin_mpy(int a, int b, int **xptr, int *xval, int xincr, int **yptr, int *yval, int yincr);</pre>
Argument:	<p><i>a</i> Integer multiplicand.</p> <p><i>b</i> Integer multiplier.</p> <p><i>xptr</i> Integer pointer to pointer to x prefetch.</p> <p><i>xval</i> Integer pointer to value of x prefetch.</p> <p><i>xincr</i> Integer increment value of x prefetch.</p> <p><i>yptr</i> Integer pointer to pointer to y prefetch.</p> <p><i>yval</i> Integer pointer to value of y prefetch.</p> <p><i>yincr</i> Integer increment value of y prefetch.</p> <p><i>AWB</i> Integer pointer to accumulator selection.</p>
Return Value:	Returns the cleared value result to an accumulator.
Assembler Operator / Machine Instruction:	<i>mpyn</i>
Error Messages	An error message will be displayed if:
	<ul style="list-style-type: none">• the result is not an accumulator register• <i>xval</i> is a null value but <i>xptr</i> is not null• <i>yval</i> is a null value but <i>yptr</i> is not null

__builtin_msc

Description: compute $a \times b$ and subtract from accumulator; also prefetch data ready for a future MAC operation.
xptr may be null to signify no X prefetch to be performed, in which case the values of *xincr* and *xval* are ignored, but required.
yptr may be null to signify no Y prefetch to be performed, in which case the values of *yincr* and *yval* are ignored, but required.
xval and *yval* nominate the address of a C variable where the prefetched value will be stored.
xincr and *yincr* may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.
If *AWB* is non null, the other accumulator will be written back into the referenced variable.

For example:

```
register int result asm("A");
int *xmemory;
int *ymemory;
int xVal, yVal;

result = __builtin_msc(xVal, yVal,
                      &xmemory, &xVal, 2,
                      &ymemory, &yVal, 2, 0);
```

might generate:

```
msc w4*w5, A, [w8]+=2, w4, [w10]+=2, w5
```

Prototype:

```
int __builtin_msc(int a, int b,
                 int **xptr, int *xval, int xincr,
                 int **yptr, int *yval, int yincr, int *AWB);
```

Argument:

<i>a</i>	Integer multiplicand.
<i>b</i>	Integer multiplier.
<i>xptr</i>	Integer pointer to pointer to x prefetch.
<i>xval</i>	Integer pointer to value of x prefetch.
<i>xincr</i>	Integer increment value of x prefetch.
<i>yptr</i>	Integer pointer to pointer to y prefetch.
<i>yval</i>	Integer pointer to value of y prefetch.
<i>yincr</i>	Integer increment value of y prefetch.
<i>AWB</i>	Integer pointer to accumulator selection.

Return Value: Returns the cleared value result to an accumulator.

Assembler msc

Operator / Machine

Instruction:

Error Messages An error message will be displayed if:

- the result is not an accumulator register
- *xval* is a null value but *xptr* is not null
- *yval* is a null value but *yptr* is not null

__builtin_mulss

Description: The function computes the product $p0 \times p1$. Function arguments are signed integers, and the function result is a signed long integer. The command-line option `-Wconversions` can be used to detect unexpected sign conversions.

Prototype:

```
signed long __builtin_mulss(const signed int p0,
                           const signed int p1);
```

16-Bit Language Tools Libraries

__builtin_mulss (Continued)

Argument: *p0* multiplicand
p1 multiplier

Return Value: Returns the signed long integer value of the product $p0 \times p1$.

Assembler `mul.ss`

Operator / Machine

Instruction:

__builtin_mulsu

Description: The function computes the product $p0 \times p1$. Function arguments are integers with mixed signs, and the function result is a signed long integer. The command-line option `-Wconversions` can be used to detect unexpected sign conversions. This function supports the full range of addressing modes of the instruction, including immediate mode for operand *p1*.

Prototype: `signed long __builtin_mulsu(const signed int p0, const unsigned int p1);`

Argument: *p0* multiplicand
p1 multiplier

Return Value: Returns the signed long integer value of the product $p0 \times p1$.

Assembler `mul.su`

Operator / Machine

Instruction:

__builtin_mulus

Description: The function computes the product $p0 \times p1$. Function arguments are integers with mixed signs, and the function result is a signed long integer. The command-line option `-Wconversions` can be used to detect unexpected sign conversions. This function supports the full range of addressing modes of the instruction.

Prototype: `signed long __builtin_mulus(const unsigned int p0, const signed int p1);`

Argument: *p0* multiplicand
p1 multiplier

Return Value: Returns the signed long integer value of the product $p0 \times p1$.

Assembler `mul.us`

Operator / Machine

Instruction:

__builtin_muluu

Description: The function computes the product $p0 \times p1$. Function arguments are unsigned integers, and the function result is an unsigned long integer. The command-line option `-Wconversions` can be used to detect unexpected sign conversions. This function supports the full range of addressing modes of the instruction, including immediate mode for operand *p1*.

Prototype: `unsigned long __builtin_muluu(const unsigned int p0, const unsigned int p1);`

Argument: *p0* multiplicand
p1 multiplier

`__builtin_muluu` (Continued)

Return Value: Returns the signed long integer value of the product $p0 \times p1$.
Assembler `mul.uu`
Operator / Machine
Instruction:

`__builtin_nop`

Description: This function will generate a `nop` instruction.
Prototype: `void __builtin_nop(void);`
Argument: None.
Return Value: Returns a no operation (`nop`).
Assembler `nop`
Operator / Machine
Instruction:

`__builtin_psvpage`

Description: The function returns the psv page number of the object whose address is given as a parameter. The argument `p` must be the address of an object in an EE data, PSV or executable memory space; otherwise an error message is produced and the compilation fails. See the `space` attribute in the "MPLAB[®] C30 C Compiler User's Guide" (DS51284).
Prototype: `unsigned int __builtin_psvpage(const void *p);`
Argument: `p` object address
Return Value: Returns the psv page number of the object whose address is given as a parameter.
Assembler `psvpage`
Operator / Machine
Instruction:
Error Messages The following error message is produced when this function is used incorrectly:
"Argument to `__builtin_psvpage()` is not the address of an object in code, psv, or eedata section".
The argument must be an explicit object address.
For example, if `obj` is object in an executable or read-only section, the following syntax is valid:
`unsigned page = __builtin_psvpage(&obj);`

`__builtin_psvoffset`

Description: The function returns the psv page offset of the object whose address is given as a parameter. The argument `p` must be the address of an object in an EE data, PSV or executable memory space; otherwise an error message is produced and the compilation fails. See the `space` attribute in the "MPLAB[®] C30 C Compiler User's Guide" (DS51284).
Prototype: `unsigned int __builtin_psvoffset(const void *p);`
Argument: `p` object address
Return Value: Returns the psv page number offset of the object whose address is given as a parameter.

16-Bit Language Tools Libraries

__builtin_psvoffset (Continued)

Assembler psvoffset

Operator / Machine

Instruction:

Error Messages The following error message is produced when this function is used incorrectly:
"Argument to __builtin_psvoffset() is not the address of an object in code, psv, or eedata section".
The argument must be an explicit object address.
For example, if *obj* is object in an executable or read-only section, the following syntax is valid:

```
unsigned page = __builtin_psvoffset(&obj);
```

__builtin_return_address

Description: This function returns the return address of the current function, or of one of its callers. For the *level* argument, a value of 0 yields the return address of the current function, a value of 1 yields the return address of the caller of the current function, and so forth. When level exceeds the current stack depth, 0 will be returned. This function should only be used with a non-zero argument for debugging purposes.

Prototype:

```
int __builtin_return_address (const int level);
```

Argument: *level* Number of frames to scan up the call stack.

Return Value: Returns the return address of the current function, or of one of its callers.

Assembler return_address

Operator / Machine

Instruction:

__builtin_sac

Description: Shift value by *shift* (a literal between -8 and 7) and return the value. For example:

```
register int value asm("A");  
int result;
```

```
result = __builtin_sac(value,3);
```

Might generate:

```
sac A, #3, w0
```

Prototype:

```
int __builtin_sac(int value, int shift);
```

Argument: *value* Integer number to be shifted.

shift Literal amount to shift.

Return Value: Returns the shifted result to an accumulator.

Assembler sac

Operator / Machine

Instruction:

Error Messages An error message will be displayed if:

- the result is not an accumulator register
- the shift value is not a literal within range

`__builtin_sacr`

Description:	Shift value by <i>shift</i> (a literal between -8 and 7) and return the value which is rounded using the rounding mode determined by the CORCONbits.RND control bit. For example: <pre>register int value asm("A"); int result; result = __builtin_sacr(value,3);</pre> Might generate: <pre>sac.r A, #3, w0</pre>
Prototype:	<pre>int __builtin_sacr(int value, int shift);</pre>
Argument:	<i>value</i> Integer number to be shifted. <i>shift</i> Literal amount to shift.
Return Value:	Returns the shifted result to CORCON register.
Assembler Operator / Machine Instruction:	<pre>sacr</pre>
Error Messages	An error message will be displayed if: <ul style="list-style-type: none">• the result is not an accumulator register• the shift value is not a literal within range

`__builtin_sftac`

Description:	Shift accumulator by <i>shift</i> . The valid shift range is -16 to 16. For example: <pre>register int result asm("A"); int i; result = __builtin_sftac(i);</pre> Might generate: <pre>sftac A, w0</pre>
Prototype:	<pre>int __builtin_sftac(int shift);</pre>
Argument:	<i>shift</i> Literal amount to shift.
Return Value:	Returns the shifted result to an accumulator.
Assembler Operator / Machine Instruction:	<pre>sftac</pre>
Error Messages	An error message will be displayed if: <ul style="list-style-type: none">• the result is not an accumulator register• the shift value is not a literal within range

`__builtin_subab`

Description:	Subtract accumulators A and B with the result written back to the specified accumulator. For example: <pre>register int result asm("A"); result = __builtin_subab();</pre> will generate: <pre>sub A</pre>
Prototype:	<pre>int __builtin_subab(void);</pre>
Argument:	None

16-Bit Language Tools Libraries

__builtin_subab (Continued)

Return Value:	Returns the subtraction result to an accumulator.
Assembler Operator / Machine Instruction:	subad
Error Messages	An error message will be displayed if the result is not an accumulator register.

__builtin_tblpage

Description:	The function returns the table page number of the object whose address is given as a parameter. The argument <i>p</i> must be the address of an object in an EE data, PSV or executable memory space; otherwise an error message is produced and the compilation fails. See the <i>space</i> attribute in the “MPLAB® C30 C Compiler User’s Guide” (DS51284).
Prototype:	<code>unsigned int __builtin_tblpage(const void *p);</code>
Argument:	<i>p</i> object address
Return Value:	Returns the table page number of the object whose address is given as a parameter.
Assembler Operator / Machine Instruction:	tblpage
Error Messages	The following error message is produced when this function is used incorrectly: “Argument to <code>__builtin_tblpage()</code> is not the address of an object in code, psv, or eedata section”. The argument must be an explicit object address. For example, if <i>obj</i> is object in an executable or read-only section, the following syntax is valid: <code>unsigned page = __builtin_tblpage(&obj);</code>

__builtin_tboffset

Description:	The function returns the table page offset of the object whose address is given as a parameter. The argument <i>p</i> must be the address of an object in an EE data, PSV or executable memory space; otherwise an error message is produced and the compilation fails. See the <i>space</i> attribute in the <i>MPLAB® C30 C Compiler User’s Guide</i> .
Prototype:	<code>unsigned int __builtin_tboffset(const void *p);</code>
Argument:	<i>p</i> object address
Return Value:	Returns the table page number offset of the object whose address is given as a parameter.
Assembler Operator / Machine Instruction:	tboffset
Error Messages	The following error message is produced when this function is used incorrectly: “Argument to <code>__builtin_tboffset()</code> is not the address of an object in code, psv, or eedata section”. The argument must be an explicit object address. For example, if <i>obj</i> is object in an executable or read-only section, the following syntax is valid: <code>unsigned page = __builtin_tboffset(&obj);</code>

Appendix A. ASCII Character Set

TABLE A-1: ASCII CHARACTER SET

		Most 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	

Least Significant Character

16-Bit Language Tools Libraries

NOTES:

Index

Symbols

#define 10, 82, 84, 107, 114, 155, 171, 193
 #if 219
 #include 10, 82, 202
 %, Percent 262, 267, 268, 331
 -, Dash 268
 \f, Form Feed 209
 \n, Newline 209, 233, 243, 248, 259, 260, 264
 \r, Carriage Return 209
 \t, Horizontal Tab 209
 \v, Vertical Tab 209
 ^, Caret 268
 __builtin_add 384
 __builtin_addab 384
 __builtin_btg 385
 __builtin_clr 385
 __builtin_clr_prefect 386
 __builtin_divsd 387
 __builtin_divud 387
 __builtin_dmaoffset 388
 __builtin_ed 388
 __builtin_edac 389
 __builtin_fbcl 389
 __builtin_lac 390
 __builtin_mac 390
 __builtin_movsac 391
 __builtin_mpy 392
 __builtin_mbyn 393
 __builtin_msc 394
 __builtin_mulss 394
 __builtin_mulsu 395
 __builtin_mulus 395
 __builtin_muluu 395
 __builtin_nop 396
 __builtin_psvoffset 396
 __builtin_psvpage 396
 __builtin_return_address 397
 __builtin_sac 397
 __builtin_sacr 398
 __builtin_sftac 398
 __builtin_subab 398
 __builtin_tbloffset 399
 __builtin_tblpage 399
 __FILE__ 203
 __LINE__ 203
 _exit 375
 _IOFBF 234, 269, 270
 _IOLBF 234, 270
 _IONBF 234, 269, 270
 _MathError 227
 _NSETJMP 222

Numerics

0x 210, 261, 299, 300
 16-Bit Peripheral Libraries 81

A

Abnormal Termination Signal 224
 abort 203, 279
 abs 280
 Absolute Value
 Double Floating Point 347
 Integer 280
 Long Integer 290
 Single Floating Point 347
 Absolute Value Function
 abs 280
 fabs 347
 fabsf 347
 labs 290
 Access Mode
 Binary 244
 Text 244
 AckI2C 192
 acos 333
 acosf 334
 ADC, 10-Bit
 Busy 110
 Close 110
 Configure Interrupt 110
 Disable Interrupt Macro 115
 Enable Interrupt Macro 115
 Example of Use 116
 Open 111
 Read 114
 Set Channel 114
 Set Interrupt Priority Macro 115
 Start Conversion 111
 Stop Sampling 114
 ADC, 12-Bit
 Busy 103
 Close 103
 Configure Interrupt 103
 Disable Interrupt Macro 108
 Enable Interrupt Macro 108
 Example of Use 109
 Open 104
 Read 107
 Set Channel 107
 Set Interrupt Priority Macro 108
 Start Conversion 104
 Stop Sampling 107

16-Bit Language Tools Libraries

Allocate Memory.....	292	Buffer Size.....	234, 270
calloc.....	286	BufferEmptyDCI	158
Free.....	289	Buffering Modes	270
realloc	295	Buffering, See File Buffering	
Alphabetic Character		BUFSIZ.....	234, 269
Defined.....	204	Built-In Functions	
Test for.....	204	_builtin_add.....	384
Alphanumeric Character		_builtin_addab.....	384
Defined.....	204	_builtin_btg.....	385
Test for.....	204	_builtin_clr.....	385
AM/PM	331	_builtin_clr_prefect.....	386
Append.....	308, 314	_builtin_divsd.....	387
arccosine		_builtin_divud.....	387
Double Floating Point.....	333	_builtin_dmaoffset.....	388
Single Floating Point	334	_builtin_ed.....	388
arcsine		_builtin_edac.....	389
Double Floating Point.....	335	_builtin_fbcl.....	389
Single Floating Point	335	_builtin_lac.....	390
arctangent		_builtin_mac.....	390
Double Floating Point.....	336	_builtin_movsac.....	391
Single Floating Point	337	_builtin_mpy.....	392
arctangent of y/x		_builtin_mbyn.....	393
Double Floating Point.....	337	_builtin_msc.....	394
Single Floating Point	339	_builtin_mulss.....	394
Argument List.....	229, 275, 276, 277	_builtin_mulsu.....	395
Arithmetic Error Message.....	224	_builtin_mulus.....	395
ASCII Character Set.....	401	_builtin_muluu.....	395
asctime.....	326	_builtin_nop.....	396
asin.....	335	_builtin_psvoffset.....	396
asinf.....	335	_builtin_psvpage.....	396
assert.....	203	_builtin_return_address.....	397
assert.h.....	203	_builtin_sac.....	397
Assignment Suppression.....	267	_builtin_sacr.....	398
Asterisk	261, 267	_builtin_sftac.....	398
atan.....	336	_builtin_subab.....	398
atan2.....	337	_builtin_tbloffset.....	399
atan2f.....	339	_builtin_tblpage.....	399
atanf.....	337	BusyADC10.....	110
atexit.....	280, 288	BusyADC12.....	103
atof.....	282	BusyUartx.....	149
atoi.....	283	BusyXLCD.....	83
atol.....	283		
B		C	
BartlettInit.....	26	C Locale	204, 221
Base.....	299, 300	Calendar Time.....	325, 327, 328, 330, 332
10.....	215, 216, 217, 218, 358, 359	calloc.....	286, 289
2.....	217	CAN Interrupts	
e.....	357, 360	Configure.....	99
FLT_RADIX.....	214, 215, 216, 217, 218, 219	Disable Macro	100
Binary		Enable Macro	100
Base.....	217	Set Priority Macro.....	100
Mode.....	244, 272	CAN, Example of Use	101
Search.....	284	CANxAbortAll	89
Streams.....	233	CANxGetRXErrorCount.....	89
Bitfields.....	232	CANxGetTXErrorCount.....	90
BitReverseComplex.....	59	CANxInitialize	98
BlackmanInit.....	27	CANxIsBusOff	90
BORStatReset.....	127	CANxIsRXPassive.....	91
brk.....	375, 380	CANxIsRXReady.....	90
bsearch.....	284	CANxIsTXPassive.....	91
		CANxIsTXReady.....	92

CANxRecieveMessage	92	Character Testing Functions	
CANxSendMessage	93	isalnum	204
CANxSetFilter	94	isalpha	204
CANxSetMask	94	iscntrl	205
CANxSetOperationMode	95	isdigit	206
CANxSetOperationModeNoWait	96	isgraph	206
CANxSetRXMode	96	islower	207
CANxSetTXMode	97	isprint	208
Caret (^)	268	ispunct	208
Carriage Return	209	isspace	209
ceil	340	isupper	210
ceilf	341	isxdigit	210
ceiling		Characters	
Double Floating Point	340	Alphabetic	204
Single Floating Point	341	Alphanumeric	204
char		Control	205
Maximum Value	219	Convert to Lower Case Alphabetic	211
Minimum Value	219	Convert to Upper Case Alphabetic	212
Number of Bits	219	Decimal Digit	206
CHAR_BIT	219	Graphical	206
CHAR_MAX	219	Hexadecimal Digit	210
CHAR_MIN	219	Lower Case Alphabetic	207
Character Array	268	Printable	208
Character Case Mapping		Punctuation	208
Lower Case Alphabetic Character	211	Upper Case Alphabetic	210
Upper Case Alphabetic Character	212	White-Space	209
Character Case Mapping Functions		Classifying Characters	204
tolower	211	clearerr	237
toupper	212	Clearing Error Indicator	237
Character Handling, See ctype.h		clock	326
Character Input/Output Functions		clock_t	325, 326
fgetc	241	CLOCKS_PER_SEC	325
fgets	243	close	376
fputc	247	CloseADC10	110
fputs	247	CloseADC12	103
getc	258	CloseCapturex	133
getchar	259	CloseDCI	158
gets	259	CloseI2C	191
putc	263	CloseINTx	128
putchar	264	CloseMCPWM	179
puts	264	CloseOCx	139
ungetc	273	CloseQEI	174
Character Testing		CloseSPIx	166
Alphabetic Character	204	CloseTimerx	117
Alphanumeric Character	204	CloseTimerxx	117
Control Character	205	CloseUARTx	149
Decimal Digit	206	Common Definitions, See stddef.h	
Graphical Character	206	Compare Strings	310
Hexadecimal Digit	210	Comparison Function	284, 294
Lower Case Alphabetic Character	207	Comparison Functions	
Printable Character	208	memcmp	303
Punctuation Character	208	strcmp	310
Upper Case Alphabetic Character	210	strcoll	311
White-Space Character	209	strncmp	316
		strxfrm	324
		Compiler Options	
		-fno-short-double	233
		-msmart-io	233

16-Bit Language Tools Libraries

Concatenation Functions		
strcat	308	
strncat	314	
ConfigCNPullups	130	
ConfigIntADC10	110	
ConfigIntADC12	103	
ConfigIntCANx	99	
ConfigIntCapturex	134	
ConfigIntCN	130	
ConfigIntDCI	159	
ConfigIntI2C	191	
ConfigIntIOCx	140	
ConfigIntMCPWM	179	
ConfigIntQEI	174	
ConfigIntSplx	166	
ConfigIntTimerx	118	
ConfigIntTimerxx	119	
ConfigIntUARTx	150	
ConfigINTx	129	
Control Character		
Defined	205	
Test for	205	
Control Transfers	222	
Conversion	261, 267, 271	
Convert		
Character to Multibyte Character	301	
Multibyte Character to Wide Character	293	
Multibyte String to Wide Character String	293	
String to Double Floating Point	282, 297	
String to Integer	283	
String to Long Integer	283, 299	
String to Unsigned Long Integer	300	
To Lower Case Alphabetic Character	211	
To Upper Case Alphabetic Character	212	
Wide Character String to Multibyte String	301	
ConvertADC10	111	
ConvertADC12	104	
Copying Functions		
memcpy	305	
memmove	306	
memset	307	
strcpy	311	
strncpy	317	
cos	341	
cosf	342	
CosFactorInit	61	
cosh	343	
coshf	344	
cosine		
Double Floating Point	341	
Single Floating Point	342	
crt0, crt1	8	
ctime	327	
ctype.h	204	
isalnum	204	
iscntrl	205	
isdigit	206	
isgraph	206	
isalpha	204	
islower	207	
ispring	208	
ispunct	208	
isspace	209	
isupper	210	
isxdigit	210	
tolower	211	
toupper	212	
Current Argument	229	
Customer Notification Service	5	
Customer Support	6	
Customized Function	289	
D		
Dash (-)	268	
DataRdyDCI	159	
DataRdyI2C	192	
DataRdySPIx	167	
DataRdyUARTx	151	
Date and Time	330	
Date and Time Functions, See time.h		
Day of the Month	325, 326, 330	
Day of the Week	325, 326, 330	
Day of the Year	325, 331	
Daylight Savings Time	325, 328, 329	
DBL_DIG	214	
DBL_EPSILON	214	
DBL_MANT_DIG	214	
DBL_MAX	214	
DBL_MAX_10_EXP	215	
DBL_MAX_EXP	215	
DBL_MIN	215	
DBL_MIN_10_EXP	215	
DBL_MIN_EXP	216	
DCI Functions		
Close DCI	158	
Configure DCI	159	
Configure DCI Interrupt	159	
DCI RX Buffer Status	159	
DCI TX Buffer Status	158	
Example of Use	164	
Read DCI RX Buffer	162	
Write DCI TX Buffer	162	
DCI Macros		
Disable DCI Interrupt	163	
Enable DCI Interrupt	163	
Set DCI Interrupt Priority	163	
DCT	61	
DCTIP	63	
Deallocate Memory	289, 295	
Debugging Logic Errors	203	
Decimal	262, 268, 299, 300	

Decimal Digit	
Defined	206
Number Of	214, 216, 217
Test for	206
Decimal Point	261
Default Handler	223
Diagnostics, See assert.h	
difftime	328
Digit, Decimal, See Decimal Digit	
Digit, Hexadecimal, See Hexadecimal Digit	
Direct Input/Output Functions	
fread	248
fwrite	256
DisableCNx	131
DisableIntADC	108, 115
DisableIntCANx	100
DisableIntDCI	163
DisableInterrupts	127
DisableIntFLTA	188
DisableIntFLTB	189
DisableIntIC1	137
DisableIntUxRX	155
DisableIntMCPWM	188
DisableIntMI2C	198
DisableIntOCx	147
DisableIntQEI	177
DisableIntSI2C	199
DisableIntSPIx	171
DisableIntTx	123
DisableIntUxTX	156
DisableINTx	132
div	278, 286
div_t	278
Divide	
Integer	286
Long Integer	291
Divide by Zero	224, 227, 286
Document	
Layout	2
Documentation	
Conventions	3
Domain Error ...	213, 333, 334, 335, 337, 339, 341, 342, 349, 351, 357, 358, 359, 360, 365, 366, 369, 370, 371
dot	261
Double Precision Floating Point	
Machine Epsilon	214
Maximum Exponent (base 10)	215
Maximum Exponent (base 2)	215
Maximum Value	214
Minimum Exponent (base 10)	215
Minimum Exponent (base 2)	216
Minimum Value	215
Number of Binary Digits	214
Number of Decimal Digits	214
double Type	233
Dream Function	261
DSP Libraries	9

E	
EDOM	213
edom	333
Ellipses (...)	229, 268
Empty Binary File	244
Empty Text File	244
EnableCNx	131
EnableIntADC	108, 115
EnableIntCANx	100
EnableIntDCI	163
EnableIntFLTA	188
EnableIntFLTB	189
EnableIntICx	137
EnableIntUxTX	155
EnableIntMCPWM	188
EnableIntMI2C	198
EnableIntOCx	147
EnableIntQEI	177
EnableIntSI2C	199
EnableIntSPIx	171
EnableIntTx	123
EnableIntUxRX	155
EnableINTx	131
End Of File	235
Indicator	233
Seek	252
Test For	239
Environment Function	
getenv	289
Environment Variable	377
EOF	235
ERANGE	213
erange	333
errno	213, 333
errno.h	213, 333, 378
EDOM	213
ERANGE	213
errno	213
Error Codes	213, 313
Error Conditions	333
Error Handler	286
Error Handling Functions	
clearerr	237
feof	239
ferror	240
perror	260
Error Indicator	233
Error Indicators	
Clearing	237, 266
End Of File	237, 243
Error	237, 243
Test For	240
Error Signal	223
Errors, See errno.h	
Errors, Testing For	213

16-Bit Language Tools Libraries

Example of Use		
ADC, 10-Bit	116	
ADC, 12-Bit	109	
CAN	101	
DCI	164	
I2C	200	
Input Capture	138	
Output Compare	148	
PWM	190	
QEI Functions	178	
SPI	172	
Timers	124	
UART	157	
Exception Error	286	
exit	272, 278, 280, 288, 375	
EXIT_FAILURE	278	
EXIT_SUCCESS	278	
exp	345	
expf	346	
Exponential and Logarithmic Functions		
exp	345	
expf	346	
frexp	352	
frexpf	353	
ldexp	354	
ldexpf	355	
log	357	
log10	358	
log10f	359	
logf	360	
modf	361	
modff	362	
Exponential Function		
Double Floating Point	345	
Single Floating Point	346	
F		
fabs	347	
fabsf	347	
fclose	238, 376	
feof	237, 239	
ferror	237, 240	
fflush	241, 381	
FFTComplex	64	
FFTComplexIP	66	
fgetc	241, 379	
fgetpos	242, 377	
fgets	243, 379	
Field Width	261	
FILE	233, 234	
File Access Functions		
fclose	238	
fflush	241	
fopen	244	
freopen	250	
setbuf	269	
setvbuf	270	
File Access Modes	233, 244	
File Buffering		
Fully Buffered	233, 234	
Line Buffered	233, 234	
Unbuffered	233, 234	
File Operations		
Remove	265	
Rename	265	
File Positioning Functions		
fgetpos	242	
fseek	252	
fsetpos	253	
ftell	255	
rewind	266	
FILENAME_MAX	235	
File-Position Indicator	233, 234, 241, 242, 247, 248, 253,	256
Files, Maximum Number Open	235	
Filtering Functions	38	
FIR	41	
FIRDecimate	42	
FIRDelayInit	43	
FIRInterpDelayInit	45	
FIRInterpolate	44	
FIRLattice	45	
FIRLMS	46	
FIRLMSNorm	47	
FIRStruct	40	
FIRStructInit	49	
IIRCanonic	50	
IIRCanonicInit	51	
IIRLattice	52	
IIRLattice OCTAVE model	56	
IIRLatticeInit	53	
IIRTransposed	54	
IIRTransposedInit	55	
FIR	41	
FIRDecimate	42	
FIRDelayInit	43	
FIRInterpDelayInit	45	
FIRInterpolate	44	
FIRLattice	45	
FIRLMS	46	
FIRLMSNorm	47	
FIRStruct	40	
FIRStructInit	49	
flags	261	

float.h	214	Format Specifiers	261, 267
DBL_DIG	214	Formatted I/O Routines	233
DBL_EPSILON	214	Formatted Input/Output Functions	
DBL_MANT_DIG	214	fprintf	246
DBL_MAX	214	fscanf	250
DBL_MAX_10_EXP	215	printf	261
DBL_MAX_EXP	215	scanf	267
DBL_MIN	215	sprintf	271
DBL_MIN_10_EXP	215	sscanf	271
DBL_MIN_EXP	216	vfprintf	275
FLT_DIG	216	vprintf	276
FLT_EPSILON	216	vsprintf	277
FLT_MANT_DIG	216	Formatted Text	
FLT_MAX	216	Printing	271
FLT_MAX_10_EXP	216	Scanning	271
FLT_MAX_EXP	217	fpos_t	234
FLT_MIN	217	fprintf	233, 246
FLT_MIN_10_EXP	217	fputc	247
FLT_MIN_EXP	217	fputs	247
FLT_RADIX	217	fraction and exponent function	
FLT_ROUNDS	217	Double Floating Point	352
LDBL_DIG	217	Single Floating Point	353
LDBL_EPSILON	218	Fraction Digits	261
LDBL_MANT_DIG	218	fread	248, 379
LDBL_MAX	218	free	289
LDBL_MAX_10_EXP	218	Free Memory	289
LDBL_MAX_EXP	218	freopen	233, 250, 378
LDBL_MIN	218	frexp	352
LDBL_MIN_10_EXP	218	frexpf	353
LDBL_MIN_EXP	219	fscanf	233, 250
Floating Point		fseek	252, 273, 377
Limits	214	fsetpos	253, 273, 377
No Conversion	233	ftell	255, 377
Types, Properties Of	214	Full Buffering	269, 270
Floating Point, See float.h		Fully Buffered	233, 234
Floating-Point Error Signal	224	fwrite	256
floor	348	G	
Double Floating Point	348	getc	258
Single Floating Point	348	getchar	259
floorf	348	getcSPIx	171
FLT_DIG	216	getcUARTx	155
FLT_EPSILON	216	getenv	289, 377
FLT_MANT_DIG	216	gets	259, 379
FLT_MAX	216	getsSPIx	170
FLT_MAX_10_EXP	216	getsUARTx	154
FLT_MAX_EXP	217	GMT	328
FLT_MIN	217	gmtime	328, 329
FLT_MIN_10_EXP	217	Graphical Character	
FLT_MIN_EXP	217	Defined	206
FLT_RADIX	217	Test for	206
FLT_RADIX Digit		Greenwich Mean Time	328
Number Of	214, 216, 218		
FLT_ROUNDS	217		
Flush	241, 288		
fmod	349		
fmodf	351		
-fno-short-double	214, 215, 216, 233		
fopen	233, 244, 270, 378		
FOPEN_MAX	235		
Form Feed	209		

16-Bit Language Tools Libraries

H

h modifier	262, 267
HammingInit	28
Handler	
Default	223
Error	286
Interrupt	227
Nested	222
Signal	223, 228
Signal Type	223
Handling	
Interrupt Signal	228
HanningInit	28
Header Files	
assert.h	203
ctype.h	204
errno.h	213, 333, 378
float.h	214
limits.h	219
locale.h	221
math.h	333
setjmp.h	222
signal.h	223
stdarg.h	229
stddef.h	231
stdio.h	233, 379
stdlib.h	278, 377, 380
string.h	302
time.h	325, 381
Heap	376
Helper Functions	374
Hexadecimal	262, 268, 299, 300
Hexadecimal Conversion	261
Hexadecimal Digit	
Defined	210
Test for	210
Horizontal Tab	209
Hour	325, 326, 330
HUGE_VAL	333
Hyperbolic Cosine	
Double Floating Point	343
Single Floating Point	344
Hyperbolic Functions	
cosh	343
coshf	344
sinh	367
sinhf	368
tanh	372
tanhf	373
Hyperbolic Sine	
Double Floating Point	367
Single Floating Point	368
Hyperbolic Tangent	
Double Floating Point	372
hyperbolic tangent	
Single Floating Point	373

I

I/O Port Functions	
Configure CN Interrupts	130
Configure CN Pull-Ups	130
Configure INT	129
Disable INT	128
I/O Port Macros	
Disable CN Interrupts	131
Disable Interrupts	132
Enable CN Interrupts	131
Enable Interrupts	131
Set Interrupts Priority	132
I2C Functions	
Acknowledge I2C	192
Close I2C	191
Configure I2C	195
Configure I2C Interrupt	191
Data Ready I2C	192
Example of Use	200
Idle I2C	192
Master I2C Get String	193
Master I2C Put String	193
Not Acknowledge I2C	194
Read Master I2C	194
Read Slave I2C	197
Restart I2C	196
Slave I2C Get String	196
Slave I2C Put String	197
Start I2C	198
Stop I2C	198
Write Master I2C	194
Write Slave I2C	197
I2C Macros	
Disable Master I2C Interrupt	198
Disable Slave I2C Interrupt	199
Enable Master I2C Interrupt	198
Enable Slave I2C Interrupt	199
Set Master I2C Interrupt Priority	199
Set Slave I2C Interrupt Priority	199
IdleI2C	192
IFFTComplex	67
IFFTComplexIP	69
Ignore Signal	223
IIRCanonic	50
IIRCanonicInit	51
IIRLattice	52
IIRLattice OCTAVE model	56
IIRLatticeInit	53
IIRTransposed	54
IIRTransposedInit	55
Illegal Instruction Signal	225
Implementation-Defined Limits, See limits.h	
Indicator	
End Of File	233, 235
Error	233, 240
File Position	233, 241, 242, 247, 248, 253, 256
Infinity	333
Input and Output, See stdio.h	

Input Capture Functions		
Close Input Capture	133	
Configure Input Capture	135	
Configure Input Capture Interrupt	134	
Example of Use	138	
Read All Input Captures	136	
Input Capture Macros		
Disable Capture Interrupt	137	
Enable Capture Interrupt	137	
Set Capture Interrupt Priority	137	
Input Formats	233	
Instruction Cycles	328, 329, 332	
int		
Maximum Value	219	
Minimum Value	219	
INT_MAX	219	
INT_MIN	219	
Integer Limits	219	
Internal Error Message	313	
Internet Address	5	
Interrupt Handler	227	
Interrupt Signal	225	
Interrupt Signal Handling	228	
Interruption Message	225	
Invalid Executable Code Message	225	
Invalid Storage Request Message	226	
Inverse Cosine, <i>See</i> arccosine		
Inverse Sine, <i>See</i> arcsine		
Inverse Tangent, <i>See</i> arctangent		
isalnum	204	
isBOR	125	
iscntrl	205	
isdigit	206	
isgraph	206	
isalpha	204	
islower	207	
isLVD	125	
isMCLR	126	
isPOR	125	
isprint	208	
ispunct	208	
isspace	209	
isupper	210	
isWDTTO	126	
isWDTWU	126	
isWU	127	
isxdigit	210	
J		
jmp_buf	222	
Justify	261	
K		
KaiserInit	29	
L		
L modifier	262, 267	
l modifier	262, 267	
L_tmpnam	235, 273	
labs	290	
LC_ALL	221	
LC_COLLATE	221	
LC_CTYPE	221	
LC_MONETARY	221	
LC_NUMERIC	221	
LC_TIME	221	
LCD, External	82	
Busy	83	
Example of Use	88	
Open	83	
Put String	84	
Read Address	85	
Read Data	85	
Set Character Generator Address	86	
Set Display Data Address	86	
Write Command	87	
Write Data	86	
Iconv, struct	221	
LDBL_DIG	217	
LDBL_EPSILON	218	
LDBL_MANT_DIG	218	
LDBL_MAX	218	
LDBL_MAX_10_EXP	218	
LDBL_MAX_EXP	218	
LDBL_MIN	218	
LDBL_MIN_10_EXP	218	
LDBL_MIN_EXP	219	
ldexp	354	
ldexpf	355	
ldiv	278, 291	
ldiv_t	278	
Leap Second	325, 331	
Left Justify	261	
libpic30, Rebuilding	374	
Libraries		
16-Bit Peripheral	81	
DSP	9	
Standard C	201	
Standard C Math	333	
Limits		
Floating Point	214	
Integer	219	
limits.h	219	
CHAR_BITS	219	
CHAR_MAX	219	
CHAR_MIN	219	
INT_MAX	219	
INT_MIN	219	
LLONG_MAX	219	
LLONG_MIN	220	
LONG_MAX	220	
LONG_MIN	220	
MB_LEN_MAX	220	
SCHAR_MAX	220	
SCHAR_MIN	220	
SHRT_MAX	220	
SHRT_MIN	220	
UCHAR_MAX	221	
UINT_MAX	221	
ULLONG_MAX	221	
ULONG_MAX	221	
USHRT_MAX	221	

16-Bit Language Tools Libraries

Line Buffered	233, 234
Line Buffering	270
ll modifier	262, 267
LLONG_MAX	219
LLONG_MIN	220
Load Exponent Function	
Double Floating Point	354
Single Floating Point	355
Local Time	327, 329, 330
Locale, C	204, 221
Locale, Other	221
locale.h	221
localeconv	221
Localization, See locale.h	
localtime	327, 328, 329
Locate Character	309
log	357
log10	358
log10f	359
Logarithm Function	
Double Floating Point	358
Single Floating Point	359
Logarithm Function, Natural	
Double Floating Point	357
Single Floating Point	360
logf	360
Logic Errors, Debugging	203
Long Double Precision Floating Point	
Machine Epsilon	218
Maximum Exponent (base 10)	218
Maximum Exponent (base 2)	218
Maximum Value	218
Minimum Exponent (base 10)	218
Minimum Exponent (base 2)	219
Minimum Value	218
Number of Binary Digits	218
Number of Decimal Digits	217
long double Type	233
long int	
Maximum Value	220
Minimum Value	220
long long int	
Maximum Value	219
Minimum Value	220
long long unsigned int	
Maximum Value	221
long unsigned int	
Maximum Value	221
LONG_MAX	220
LONG_MIN	220
longjmp	222
Lower Case Alphabetic Character	
Convert To	211
Defined	207
Test for	207
lseek	377

M

Machine Epsilon	
Double Floating Point	214
Long Double Floating Point	218
Single Floating Point	216
Magnitude	333, 345, 346, 349, 351, 367, 368
malloc	289, 292, 375, 380
Mapping Characters	204
MastergetsI2C	193
MasterputsI2C	193
MasterReadI2C	194
MasterWriteI2C	194
Math Exception Error	286
math.h	333
acos	333
acosf	334
asin	335
asinf	335
atan	336
atan2	337
atan2f	339
atanf	337
ceil	340
ceilf	341
cos	341
cosf	342
cosh	343
coshf	344
exp	345
expf	346
fabs	347
fabsf	347
floor	348
floorf	348
fmod	349
fmodf	351
frexp	352
frexpf	353
HUGE_VAL	333
ldexp	354
ldexpf	355
log	357
log10	358
log10f	359
logf	360
modf	361
modff	362
pow	363
powf	364
sin	365
sinf	366
sinh	367
sinhf	368
sqrt	369
sqrtf	370
tan	371
tanf	371
tanh	372
tanhf	373

Mathematical Functions, See math.h

Matrix Functions.....	31	Minimum Value	
MatrixAdd.....	33	Double Floating-Point Exponent (base 10)	215
MatrixInvert.....	37	Double Floating-Point Exponent (base 2)	216
MatrixMultiply.....	34	Long Double Floating-Point Exponent (base 10) ..	218
MatrixScale.....	35	Long Double Floating-Point Exponent (base 2)	219
MatrixSubtract.....	35	Single Floating-Point Exponent (base 10)	217
MatrixTranspose.....	36	Single Floating-Point Exponent (base 2)	217
Maximum		Type char	219
Multibyte Character.....	279	Type Double.....	215
Maximum Value		Type int	219
Double Floating-Point Exponent (base 10)	215	Type Long Double.....	218
Double Floating-Point Exponent (base 2)	215	Type long int	220
Long Double Floating-Point Exponent (base 10) ..	218	Type long long int.....	220
Long Double Floating-Point Exponent (base 2)	218	Type short int	220
Multibyte Character.....	220	Type signed char	220
rand.....	279	Type Single	217
Single Floating-Point Exponent (base 10)	216	Minute	325, 326, 331
Single Floating-Point Exponent (base 2)	217	mktime.....	330
Type char	219	modf	361
Type Double	214	modff	362
Type int	219	modulus function	
Type Long Double	218	Double Floating Point.....	361
Type long int	220	Single Floating Point	362
Type long long int	219	Month	325, 326, 330, 331
Type long long unsigned int.....	221	-msmart-io	233
Type long unsigned int.....	221	Multibyte Character	279, 293, 301
Type short int	220	Maximum Number of Bytes.....	220
Type signed char	220	Multibyte String	293, 301
Type Single.....	216	N	
Type unsigned char	221	NaN	333
Type unsigned int	221	Natural Logarithm	
Type unsigned short int	221	Double Floating Point.....	357
MB_CUR_MAX.....	279	Single Floating Point	360
MB_LEN_MAX.....	220	NDEBUG	203
mblen	293	Nearest Integer Functions	
mbstowcs	293	ceil.....	340
mbtowc.....	293	ceilf.....	341
memchr	302	floor	348
memcmp	303	floorf	348
memcpy	305	Nested Signal Handler	222
memmove	306	Newline	209, 233, 243, 248, 259, 260, 264
Memory		No Buffering	233, 234, 269, 270
Allocate	286, 292	Non-Local Jumps, See setjmp.h	
Deallocate	289	NotAckI2C	194
Free	289	NULL	221, 231, 235, 279, 302, 326
Reallocate.....	295	O	
memset	307	Object Module Format.....	7
Message		Octal	262, 268, 299, 300
Arithmetic Error	224	Octal Conversion.....	261
Interrupt	225	offsetof	232
Invalid Executable Code	225	OMF	7
Invalid Storage Request	226	open	378
Termination Request	226	OpenADC10.....	111
Microchip Internet Web Site.....	5	OpenADC12.....	104
		OpenCapturex.....	135
		OpenDCI	159
		OpenI2C.....	195

16-Bit Language Tools Libraries

OpenMCPWM	180	printf	233, 261
OpenOCx	141	Processor Clocks per Second	325
OpenQEI	175	Processor Time	325, 326
OpenSPIx	168	Pseudo-Random Number.....	295, 297
OpenTimerx	119	ptrdiff_t	231
OpenTimerxx.....	120	Punctuation Character	
OpenUART	151	Defined	208
OpenXLCD	83	Test for	208
Output Compare Functions		Pushed Back	273
Close Compare	139	putc.....	263
Configure Compare.....	141	putchar	264
Configure Compare Interrupt	140	putcSPIx	171
Example of Use.....	148	putcUARTx	155
Read Compare Duty Cycle	144	putrsXLCD.....	84
Read Compare Duty Cycle - PWM mode	143	puts.....	264
Set Compare Duty Cycle.....	146	putsSPIx	170
Set Compare Duty Cycle - PWM mode.....	145	putsUARTx	154
Output Compare Macros		putsXLCD	84
Disable Compare Interrupt.....	147	PvrrideMCPWM.....	182
Enable Compare Interrupt.....	147	PWM Functions	
Set Compare Interrupt Priority	147	Close PWM	179
Output Formats	233	Configure Override.....	182
Overflow Errors 213, 333, 345, 346, 354, 356, 363, 364		Configure PWM	180
Overlap.....	305, 306, 308, 311, 314, 317	Configure PWM Interrupt	179
P		Example of Use	190
Pad Characters	261	Set PWM Dead Time Assignment.....	184
Percent	262, 267, 268, 331	Set PWM Dead Time Generation.....	185
Peripheral Libraries, 16-Bit.....	81	Set PWM Duty Cycle.....	183
perror.....	260	Set PWM FaultA.....	186
pic30-libs	374	Set PWM FaultB.....	187
_exit	375	PWM Macros	
brk	375	Disable FLTA Interrupt	188
close.....	376	Disable FLTB Interrupt	189
getenv	377	Disable PWM Interrupt	188
lseek.....	377	Enable FLTA Interrupt.....	188
open	378	Enable FLTB Interrupt.....	189
read.....	379	Enable PWM Interrupt.....	188
remove	379	Set FLTA Interrupt Priority	189
rename.....	379	Set FLTB Interrupt Priority	189
sbrk	380	Set PWM Interrupt Priority	188
system.....	380	Q	
time	381	QEI Functions	
write	381	Close QEI	174
Plus Sign	261	Configure QEI	175
Pointer, Temporary.....	295	Configure QEI Interrupt	174
PORStatReset.....	127	Example of Use	178
pow.....	363	Read QEI Position Count	176
Power Function		Write QEI Position Count	177
Double Floating Point.....	363	QEI Macros	
Single Floating Point	364	Disable QEI Interrupt.....	177
Power Functions		Enable QEI Interrupt	177
pow	363	Set QEI Interrupt Priority	177
powf	364	qsort	284, 294
powf.....	364	Quick Sort.....	294
precision.....	261		
Prefix	210, 261		
Print Formats.....	233		
Printable Character			
Defined.....	208		
Test for	208		

R

Radix	217
raise	223, 224, 225, 226, 227, 228
rand	295, 297
RAND_MAX	279, 295
Range	268
Range Error	213, 299, 300, 343, 344, 345, 346, 354, 356, 363, 364, 367, 368
read	379
ReadADC10	114
ReadADC12	107
ReadAddrXLCD	85
ReadCapture	136
ReadDataXLCD	85
ReadDCI	162
ReadDCOCxPWM	143
Reading, Recommended	4
ReadQEI	176
ReadRegOCx	144
ReadSPIx	167
ReadTimerx	121
ReadTimerxx	122
ReadUARTx	153
realloc	289, 295
Reallocate Memory	295
Rebuilding the libpic30 library	374
Registered Functions	280, 288
Remainder	
Double Floating Point	349
Single Floating Point	351
Remainder Functions	
fmod	349
fmodf	351
remove	265, 379
rename	265, 379
Reset	279, 301
Reset File Pointer	266
Reset/Control Functions	
Low Voltage Detect	125
Master Clear Reset	126
Reset from Brown-out	125
Reset from Power-on	125
Wake-up from Sleep	127
Watchdog Timer Time-out	126
Watchdog Timer Wake-up	126
Reset/Control Macros	
Disable All Interrupts	127
Disable Watchdog Timer	128
Enable Watchdog Timer	128
Reset BOR bit	127
Reset POR bit	127
RestartI2C	196
rewind	266, 273, 377
Rounding Mode	217

S

sbrk	376, 380
Scan Formats	233
scanf	233, 267
SCHAR_MAX	220

SCHAR_MIN	220
Search Functions	
memchr	302
strchr	309
strcspn	312
strpbrk	319
strrchr	320
strspn	321
strstr	322
strtok	323
Second	325, 326, 328, 331
Seed	295, 297
Seek	
From Beginning of File	252
From Current Position	252
From End Of File	252
SEEK_CUR	235, 252
SEEK_END	236, 252
SEEK_SET	236, 252
setbuf	233, 234, 269
SetCGRamAddr	86
SetChanADC10	114
SetChanADC12	107
SetDCMCPWM	183
SetDCOCxPWM	145
SetDDRamAddr	86
setjmp	222
setjmp.h	222
jmp_buf	222
longjmp	222
setjmp	222
setlocale	221
SetMCPWMDeadTimeAssignment	184
SetMCPWMDeadTimeGeneration	185
SetMCPWMFaultA	186
SetMCPWMFaultB	187
SetPointIntUxRX	156
SetPriorityIntADC	108, 115
SetPriorityIntCANx	100
SetPriorityIntDCI	163
SetPriorityIntFLTA	189
SetPriorityIntFLTb	189
SetPriorityIntICx	137, 147
SetPriorityIntMCPWM	188
SetPriorityIntMI2C	199
SetPriorityIntQEI	177
SetPriorityIntSI2C	199
SetPriorityIntSPIx	171
SetPriorityIntTx	124
SetPriorityIntUxTX	156
SetPriorityIntx	132
SetPulseOCx	146
setvbuf	233, 234, 270
short int	
Maximum Value	220
Minimum Value	220
SHRT_MAX	220
SHRT_MIN	220
sig_atomic_t	223
SIG_DFL	223

16-Bit Language Tools Libraries

SIG_ERR	223	SlaveReadI2C	197
SIG_IGN.....	223	SlaveWriteI2C	197
SIGABRT	224	Sort, Quick.....	294
SIGFPE	224	Source File Name.....	203
SIGILL	225	Source Line Number	203
SIGINT	225	Space	261
Signal		Space Character	
Abnormal Termination.....	224	Defined.....	209
Error.....	223	Test for	209
Floating-Point Error.....	224	Specifiers.....	261, 267
Ignore.....	223	SPI Functions	
Illegal Instruction	225	Close SPI	166
Interrupt.....	225	Configure SPI.....	168
Reporting	227	Configure SPI Interrupt	166
Termination Request.....	226	Example of Use.....	172
signal.....	224, 225, 226, 228	Read SPI RX Buffer	167
Signal Handler.....	223, 228	SPI Buffer Status.....	167
Signal Handler Type.....	223	SPI Get Character.....	171
Signal Handling, See signal.h		SPI Get String	170
signal.h.....	223	SPI Put Character	171
raise	227	SPI Put String.....	170
sig_atomic_t.....	223	Write SPI TX Buffer.....	168
SIG_DFL	223	SPI Macros	
SIG_ERR	223	Disable SPI Interrupt	171
SIG_IGN	223	Enable SPI Interrupt.....	171
SIGABRT	224	Set SPI Interrupt Priority	171
SIGFPE	224	sprintf.....	233, 271
SIGILL	225	sqrt	369
SIGINT	225	sqrtf	370
signal.....	228	Square Root Function	
SIGSEGV	226	Double Floating Point.....	369
SIGTERM.....	226	Single Floating Point	370
signed char		Square Root Functions	
Maximum Value	220	sqrt	369
Minimum Value	220	sqrtf	370
SIGSEGV	226	srand	297
SIGTERM.....	226	sscanf	233, 271
sim30 simulator	374	Stack	376
sin.....	365	Standard C Library	201
sine		Standard C Locale.....	204
Double Floating Point.....	365	Standard Error.....	233, 236
Single Floating Point	366	Standard Input.....	233, 236
sinf.....	366	Standard Output	233, 236
Single Precision Floating Point		StartI2C	198
Machine Epsilon.....	216	Start-up.....	233
Maximum Exponent (base 10)	216	Module, Alternate	8
Maximum Exponent (base 2)	217	Module, Primary	8
Maximum Value	216	stdarg.h	229
Minimum Exponent (base 10)	217	va_arg	229
Minimum Exponent (base 2)	217	va_end	231
Minimum Value	217	va_list.....	229
Number of Binary Digits	216	va_start	231
Number of Decimal Digits	216	stddef.h.....	231
sinh.....	367	NULL	231
sinhf.....	368	offsetof	232
size	262	ptrdiff_t	231
size_t.....	231, 234, 278, 302, 325	size_t.....	231
sizeof.....	231, 234, 278, 302, 325	wchar_t.....	231
SlavegetsI2C	196	stderr	203, 233, 235, 236, 260
SlaveputsI2C	197	stdin.....	233, 235, 236, 259, 267

stdio.h	233, 379	vsprintf	277
_IOFBF	234	stdlib.h	278, 377, 380
_IOLBF	234	abort	279
_IONBF	234	abs	280
BUFSIZ	234	atexit	280
clearerr	237	atof	282
EOF	235	atoi	283
fclose	238	atol	283
feof	239	bsearch	284
ferror	240	calloc	286
fflush	241	div	286
fgetc	241	div_t	278
fgetpos	242	exit	288
fgets	243	EXIT_FAILURE	278
FILE	234	EXIT_SUCCESS	278
FILENAME_MAX	235	free	289
fopen	244	getenv	289
FOPEN_MAX	235	labs	290
fpos_t	234	ldiv	291
fprintf	246	ldiv_t	278
fputc	247	malloc	292
fputs	247	MB_CUR_MAX	279
fread	248	mblen	293
freopen	250	mbstowcs	293
fscanf	250	mbtowc	293
fseek	252	NULL	279
fsetpos	253	qsort	294
ftell	255	rand	295
fwrite	256	RAND_MAX	279
getc	258	realloc	295
getchar	259	size_t	278
gets	259	srand	297
L_tmpnam	235	strtod	297
NULL	235	strtol	299
perror	260	strtoul	300
printf	261	system	301
putc	263	wchar_t	278
putchar	264	wctomb	301
puts	264	wxstombs	301
remove	265	stdout	233, 235, 236, 261, 264
rename	265	StopI2C	198
rewind	266	StopSampADC10	114
scanf	267	StopSampADC12	107
SEEK_CUR	235	strcat	308
SEEK_END	236	strchr	309
SEEK_SET	236	strcmp	310
setbuf	269	strcoll	311
setvbuf	270	strcpy	311
size_t	234	strcspn	312
sprintf	271	Streams	233
sscanf	271	Binary	233
stderr	236	Buffering	270
stdin	236	Closing	238, 288
stdout	236	Opening	244
TMP_MAX	236	Reading From	258
tmpfile	272	Text	233
tmpnam	273	Writing To	256, 263
ungetc	273	strerror	313
vfprintf	275	strftime	330
vprintf	276		

16-Bit Language Tools Libraries

String		
Length	313	
Search	322	
Transform	324	
String Functions, See string.h		
string.h	302	
memchr	302	
memcmp	303	
memcpy	305	
memmove	306	
memset	307	
NULL	302	
size_t	302	
strcat	308	
strchr	309	
strcmp	310	
strcoll	311	
strcpy	311	
strcspn	312	
strerror	313	
strlen	313	
strncat	314	
strncmp	316	
strncpy	317	
strpbrk	319	
strrchr	320	
strspn	321	
strstr	322	
strtok	323	
strxfrm	324	
strlen	313	
strncat	314	
strncmp	316	
strncpy	317	
strpbrk	319	
strrchr	320	
strspn	321	
strstr	322	
strtod	282, 297	
strtok	323	
strtol	283, 299	
strtoul	300	
struct lconv	221	
struct tm	325	
strxfrm	324	
Substrings	323	
Subtracting Pointers	231	
Successful Termination	278	
system	301, 380	
T		
Tab	209	
tan	371	
tanf	371	
tangent		
Double Floating Point	371	
Single Floating Point	371	
tanh	372	
tanhf	373	
Temporary		
File	272, 288	
Filename	235, 273	
Pointer	295	
Termination		
Request Message	226	
Request Signal	226	
Successful	278	
Unsuccessful	278	
Text Mode	244	
Text Streams	233	
Ticks	325, 326, 328	
time	332, 381	
Time Difference	328	
Time Structure	325, 330	
Time Zone	331	
time.h	325, 381	
asctime	326	
clock	326	
clock_t	325	
CLOCKS_PER_SEC	325	
ctime	327	
difftime	328	
gmtime	328	
localtime	329	
mktime	330	
NULL	326	
size_t	325	
strftime	330	
struct tm	325	
time	332	
time_t	325	
time_t	325, 330, 332	
Timer Functions		
Close	117	
Close 32-bit	117	
Configure Interrupt	118	
Configure Interrupt 32-bit	119	
Example of Use	124	
Open	119	
Open 32-bit	120	
Read	121	
Read 32-bit	122	
Write	122	
Write 32-bit	123	
Timer Macros		
Disable Interrupt	123	
Enable Interrupt	123	
Set Interrupt Priority	124	
TMP_MAX	236	
tmpfile	272	
tmpnam	273	
Tokens	323	
tolower	211	
toupper	212	
Transferring Control	222	

Transform Functions	58, 73, 78	unsigned int	
BitReverseComplex	59	Maximum Value	221
CosFactorInit	61	unsigned short int	
DCT	61	Maximum Value	221
DCTIP	63	Unsuccessful Termination	278
FFTComplex	64	Upper Case Alphabetic Character	
FFTComplexIP	66	Convert To	212
IFFTComplex	67	Defined	210
IFFTComplexIP	69	Test for	210
TwidFactorInit	71, 78	USHRT_MAX	221
Transform String	324	UTC	328
Trigonometric Functions		Utility Functions, See <code>stdlib.h</code>	
acos	333	V	
acosf	334	va_arg	229, 231, 275, 276, 277
asin	335	va_end	231, 275, 276, 277
asinf	335	va_list	229
atan	336	va_start	231, 275, 276, 277
atan2	337	Variable Argument Lists, See <code>stdarg.h</code>	
atan2f	339	Variable Length Argument List	229, 231, 275, 276, 277
atanf	337	Vector Functions	13
cos	341	VectorAdd	15
cosf	342	VectorConvolve	16
sin	365	VectorCopy	17
sinf	366	VectorCorrelate	18
tan	371	VectorDotProduct	19
tanf	371	VectorMax	19
TwidFactorInit	71, 78	VectorMin	20
type	262, 268	VectorMultiply	21
U		VectorNegate	21
UART Functions		VectorPower	22, 70, 75, 76, 77
Close UART	149	VectorScale	23
Configure UART	151	VectorSubtract	23
Configure UART Interrupt	150	VectorWindow	29
Example of Use	157	VectorZeroPad	24
Read UART	153	VERBOSE_DEBUGGING	203
UART Buffer Status	151	Vertical Tab	209
UART Get Character	155	vfprintf	233, 275
UART Get String	154	vprintf	233, 276
UART Put Character	155	vsprintf	233, 277
UART Put String	154	W	
UART Status	149	wchar_t	231, 278
Write UART	153	wcstombs	301
UART Macros		wctomb	301
Disable UART RX Interrupt	155	WDTSWDisable	128
Disable UART TX Interrupt	156	WDTSWEnable	128
Enable UART RX Interrupt	155	Week	331
Enable UART TX Interrupt	155	White Space	267, 282, 283, 297
Set UART RX Interrupt Priority	156	White-Space Character	
Set UART TX Interrupt Priority	156	Defined	209
UCHAR_MAX	221	Test for	209
UINT_MAX	221	wide	278
ULLONG_MAX	221	Wide Character	293, 301
ULONG_MAX	221	Wide Character String	293, 301
Underflow Errors 213, 333, 345, 346, 354, 356, 363, 364		Wide Character Value	231
ungetc	273	Width	261
Universal Time Coordinated	328	width	261, 267
unsigned char			
Maximum Value	221		

16-Bit Language Tools Libraries

Window Functions

BartlettInit	26
BlackmanInit	27
HammingInit	28
HanningInit	28
KaiserInit	29
VectorWindow	29
write	381
WriteCmdXLCD	87
WriteDataXLCD	86
WriteDCI	162
WriteQEI	177
WriteSPIx	168
WriteTimerx	122
WriteTimerxx	123
WriteUARTx	153
WWW Address	5

Y

Year	325, 326, 331
------------	---------------

Z

Zero	333
Zero, divide by	224, 227, 286

NOTES:

16-Bit Language Tools Libraries

NOTES:

NOTES:



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:
<http://support.microchip.com>
Web Address:
www.microchip.com

Atlanta
Alpharetta, GA
Tel: 770-640-0034
Fax: 770-640-0307

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

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

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

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

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

San Jose
Mountain View, CA
Tel: 650-215-1444
Fax: 650-961-0286

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8676-6200
Fax: 86-28-8676-6599

China - Fuzhou
Tel: 86-591-8750-3506
Fax: 86-591-8750-3521

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Shunde
Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7250
Fax: 86-29-8833-7256

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-2229-0061
Fax: 91-80-2229-0062

India - New Delhi
Tel: 91-11-5160-8631
Fax: 91-11-5160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471-6166
Fax: 81-45-471-6122

Korea - Gumi
Tel: 82-54-473-4301
Fax: 82-54-473-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Penang
Tel: 60-4-646-8870
Fax: 60-4-646-5086

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

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

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

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

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

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

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820