MPLAB® C18 C COMPILER LIBRARIES

Note the following details of the code protection feature on PICmicro® MCUs.

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products 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 PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be 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 product.

If you have any further questions about this matter, please contact the local sales office nearest to you.

DNV Certification, Inc. USA — ANSI-RAB QMS ACCREDITED — DNV MSC The Netherlands Accredited by the RvA — DNV — ISO 9001 / QS-9000 REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999 and Mountain View, California in March 2002. The Company’s quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, non-volatile memory and analog products. In addition, Microchip’s quality system for the design and manufacture of development systems is ISO 9001 certified.

MPLAB® C18 C COMPILER LIBRARIES

Table of Contents

Preface

INTRODUCTION

The purpose of this document is to provide detailed information on the libraries and precompiled object files that may be used with Microchip's MPLAB® C18 C Compiler.

HIGHLIGHTS

Items discussed in this chapter are:

- About this Guide
- Warranty Registration
- Recommended Reading
- Troubleshooting
- Microchip On-Line Support
- Customer Change Notification Service
- Customer Support

ABOUT THIS GUIDE

Document Layout

This document describes MPLAB C18 libraries and precompiled object files. For a detailed discussion about using MPLAB C18 or MPLAB IDE, refer to Recommended Reading later in this chapter.

The document layout is as follows:

- **Chapter 1: Overview** – describes the libraries and precompiled object files available.
- **Chapter 2: Hardware Peripheral Functions** – describes each hardware peripheral library function.
- **Chapter 3: Software Peripheral Library** – describes each software peripheral library function.
- **Chapter 4: General Software Library** – describes each general software library function.
- **Chapter 5: Math Library** – discusses the math library functions.
- **Glossary** – A glossary of terms used in this guide.
- **Index** – Cross-reference listing of terms, features and sections of this document.
- **Worldwide Sales and Service** – gives the address, telephone and fax number for Microchip Technology Inc. sales and service locations throughout the world.

Conventions Used in This Guide

This manual uses the following documentation conventions:

Table: Documentation Conventions

Description	Represents	Examples
Code (Courier font):		
Plain characters	Sample code Filenames and paths	`#define START` `c:\autoexec.bat`
Angle brackets: < >	Variables	`<label>, <exp>`
Square brackets [ ]	Optional arguments	`MPASMWIN [main.asm]`
Curly brackets and pipe character: { | }	Choice of mutually exclusive arguments An OR selection	`errorlevel {0|1}`
Lower case characters in quotes	Type of data	`"filename"`
Ellipses...	Used to imply (but not show) additional text that is not relevant to the example	`list ["list_option..., "list_option"]`
0xnnn	A hexadecimal number where n is a hexadecimal digit	`0xFFFF, 0x007A`
Italic characters	A variable argument; it can be either a type of data (in lower case characters) or a specific example (in uppercase characters).	`char isascii (char, ch);`
Interface (Arial font):		
Underlined, italic text with right arrow	A menu selection from the menu bar	*File > Save*
Bold characters	A window or dialog button to click	**OK**, **Cancel**
Characters in angle brackets < >	A key on the keyboard	<Tab>, <Ctrl-C>
Documents (Arial font):		
Italic characters	Referenced books	*MPLAB IDE User's Guide*

Documentation Updates

All documentation becomes dated, and this user's guide is no exception. Since MPLAB IDE, MPLAB C18 and other Microchip tools are constantly evolving to meet customer needs, some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our web site to obtain the latest documentation available.

Documentation Numbering Conventions

Documents are numbered with a "DS" number. The number is located on the bottom of each page, in front of the page number. The numbering convention for the DS Number is: DSXXXXXA,

where:

XXXXX	=	The document number.
A	=	The revision level of the document.

WARRANTY REGISTRATION

Please complete the enclosed Warranty Registration Card and mail it promptly. Sending in your Warranty Registration Card entitles you to receive new product updates. Interim software releases are available at the Microchip web site.

RECOMMENDED READING

This document describes the MPLAB C18 C Compiler libraries and precompiled object files. For more information on the MPLAB C18 C compiler, the operation of MPLAB IDE and the use of other tools, the following are recommended reading.

README.C18

For the latest information on using MPLAB C18 C Compiler, read the README.C18 file (ASCII text) included with the software. This README file contains update information that may not be included in this document.

README.XXX

For the latest information on other Microchip tools (MPLAB IDE, MPLINK™ linker, etc.), read the associated README files (ASCII text file) included with the MPLAB IDE software.

MPLAB C18 C Compiler User's Guide (DS51288)

Comprehensive guide that describes the installation, operation and features of Microchip's MPLAB C18 C compiler for PIC18 devices.

MPLAB C18 C Compiler Getting Started (DS51295)

This document explains how to use MPLAB C18 with MPLAB IDE. Setting up MPLAB IDE to use the compiler and several examples of use are provided.

MPLAB IDE User's Guide (DS51025)

Comprehensive guide that describes installation and features of Microchip's MPLAB Integrated Development Environment (IDE), as well as the editor and simulator functions in the MPLAB IDE environment.

MPASM™ User's Guide with MPLINK™ and MPLIB™ (DS33014)

This user's guide describes how to use the Microchip PICmicro® MCU MPASM assembler, the MPLINK object linker and the MPLIB object librarian.

PIC18 Device Data Sheets

These documents contain information on the operation and electrical specifications of PIC18 devices. May be found on the Technical CD-ROM or our web site (see below).

Technical Library CD-ROM (DS00161)

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

Microchip Web Site

Our web site (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.

Microsoft® Windows® Manuals

This manual assumes that users are familiar with the Microsoft Windows operating system. Many excellent references exist for this software program, and should be consulted for general operation of Windows.

TROUBLESHOOTING

See the README files for information on common problems not addressed in this user's guide.

MICROCHIP ON-LINE SUPPORT

Microchip provides on-line support on the Microchip web site at:

http://www.microchip.com

A file transfer site is also available by using an FTP service connecting to:

ftp://ftp.microchip.com

The web site and file transfer site provide a variety of services. Users may download files for the latest development tools, data sheets, application notes, user' guides, articles and sample programs. A variety of Microchip specific business information is also available, including listings of Microchip sales offices and distributors. Other information available on the web site includes:

- Latest Microchip press releases
- Technical support section with FAQs
- Design tips
- Device errata
- Job postings
- Microchip consultant program member listing
- Links to other useful web sites related to Microchip products
- Conferences for products, development systems, technical information and more
- Listing of seminars and events

CUSTOMER CHANGE NOTIFICATION SERVICE

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

Go to the Microchip web site (www.microchip.com) and click on Customer Change Notification. Follow the instructions to register.

The Development Systems product group categories are:

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

Here is a description of these categories:

Compilers - The latest information on Microchip C compilers and other language tools. These include the MPLAB C17, MPLAB C18 and MPLAB C30 C Compilers; MPASM and MPLAB ASM30 assemblers; MPLINK and MPLAB LINK30 linkers; and MPLIB and MPLAB LIB30 librarians.

Emulators - The latest information on Microchip in-circuit emulators. This includes the MPLAB ICE 2000.

In-Circuit Debuggers - The latest information on Microchip in-circuit debuggers. These include the MPLAB ICD and MPLAB ICD 2.

MPLAB - The latest information on Microchip MPLAB IDE, the Windows Integrated Development Environment for development systems tools. This list is focused on the MPLAB IDE, MPLAB SIM simulator, MPLAB IDE Project Manager and general editing and debugging features.

Programmers - The latest information on Microchip device programmers. These include the PRO MATE® II device programmer and PICSTART® Plus development programmer.

CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributors
- Local Sales Office
- Field Application Engineers (FAEs)
- Corporate Applications Engineers (CAEs)
- Systems Information and Upgrade Hot Line

Customers should call their distributor or field application engineer (FAE) for support. Local sales offices are also available to help customers. See the last page of this document for a listing of sales offices and locations.

Corporate applications engineers (CAEs) may be contacted at (480) 792-7627.

Systems Information and Upgrade Line

The Systems Information and Upgrade Information Line provides system users with a listing of the latest versions of all of Microchip's development systems software products. Plus, this line provides information on how customers can receive the most current upgrade kits. The Information Line Numbers are:

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

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

NOTES:

Chapter 1. Overview

1.1 INTRODUCTION

This chapter gives an overview of the MPLAB C18 library files and precompiled object files that can be included in an application.

1.2 MPLAB C18 LIBRARIES OVERVIEW

A library is a collection of functions grouped for reference and ease of linking. See the *MPASM™ User's Guide with MPLINK™ and MPLIB™* (DS33014) for more information about creating and maintaining libraries.

The MPLAB C18 libraries are included in the `lib` subdirectory of the installation. These can be linked directly into an application using the MPLINK linker.

These files were precompiled in the `c:\mcc18\src` directory at Microchip. If you chose **not** to install the compiler and related files in the `c:\mcc18` directory, source code from the libraries will not show in the linker listing file and cannot be stepped through when using MPLAB IDE.

To include the library code in the `.lst` file and to be able to single step through library functions, follow the instructions in `README.C18` to rebuild the libraries using the supplied batch files (`.bat`) found in the `src` directory.

1.3 START-UP CODE

1.3.1 Overview

Three versions of start-up code are provided with MPLAB C18, with varying levels of initialization. In increasing order of complexity, they are:

`c018.o` initializes the C software stack and jumps to the start of the application function, `main()`.

`c018i.o` performs all of the same tasks as `c018.o` and also assigns the appropriate values to initialized data prior to calling the user's application. Initialization is required if global or static variables are set to a value when they are defined. This is the start-up code that is included in the linker script files that are provided with MPLAB C18.

`c018iz.o` performs all of the same tasks as `c018i.o` and also assigns zero to all uninitialized variables, as is required for strict ANSI compliance.

1.3.2 Source Code

The source code for the start-up routines may be found in the `src/startup` subdirectory of the compiler installation.

1.3.3 Rebuilding

Use the batch file `build.bat` to rebuild the start-up code and copy the generated object files to the `lib` directory.

Before rebuilding the start-up code with `build.bat`, verify that MPLAB C18 (`mcc18.exe`) is in your path.

1.4 PROCESSOR-INDEPENDENT LIBRARY

1.4.1 Overview

The `clib.lib` library provides functions that are supported by the core PIC18 architecture: those that are supported across all processors in the family. These functions are described in the following chapters:

- General Software Library, Chapter 4.
- Math Libraries, Chapter 5.

1.4.2 Source Code

The source code for the functions in `clib.lib` may be found in the following subdirectories of the compiler installation:

- `src\math`
- `src\delays`
- `src\ctype`
- `src\string`
- `src\stdlib`

1.4.3 Rebuilding

The batch file `makeclib.bat` may be used to rebuild the processor-independent library. Before invoking this batch file, verify that the following tools are in your path:

- MPLAB C18 (`mcc18.exe`)
- MPASM assembler (`mpasm.exe`)
- MPLIB librarian (`mplib.exe`)

Also prior to rebuilding `clib.lib`, be sure that the environment variable `MCC_INCLUDE` is set to the path of the MPLAB C18 include files (e.g., `c:\mcc18\h`).

1.5 PROCESSOR-SPECIFIC LIBRARIES

1.5.1 Overview

The processor-specific library files contain definitions that may vary across individual members of the PIC18 family. This includes all of the peripheral routines and the special function register (SFR) definitions. The peripheral routines that are provided include both those designed to use the hardware peripherals and those that implement a peripheral interface using general purpose I/O lines. The functions included in the processor-specific libraries are described in the following chapters:

- Hardware Peripheral Functions, Chapter 2.
- Software Peripheral Library, Chapter 3.

The processor-specific libraries are named:

`p` *processor*`.lib`

For example, the library file for the PIC18F8720 is named `p18f8720.lib`.

1.5.2 Source Code

The source code for the processor-specific libraries may be found in the following subdirectories of the compiler installation:

- `src\pmc`
- `src\proc`

1.5.3 Rebuilding

The batch file `makeplib.bat` may be used to rebuild the processor-independent libraries. Before invoking this batch file, verify that the following tools are in your path:

- MPLAB C18 (`mcc18.exe`)
- MPASM assembler (`mpasm.exe`)
- MPLIB librarian (`mplib.exe`)

Also prior to invoking `makeplib.bat`, be sure that the environment variable `MCC_INCLUDE` is set to the path of the MPLAB C18 include files (e.g., `c:\mcc18\h`).

NOTES:

Chapter 2. Hardware Peripheral Functions

2.1 INTRODUCTION

This chapter documents the hardware peripheral functions found in the processor-specific libraries. The source code for all of these functions is included with MPLAB C18 in the `src\pmc` subdirectory of the compiler installation.

See the *MPASM™ User's Guide with MPLINK™ and MPLIB™* (DS33014) for more information about managing libraries using the MPLIB librarian.

The following peripherals are supported by MPLAB C18 library routines:

- A/D Converter (2.2 “A/D Converter Functions”)
- Input Capture (2.3 “Input Capture Functions”)
- I^2C® (2.4 “I²C® Functions”)
- I/O Ports (2.5 “I/O Port Functions”)
- Microwire® (2.6 “Microwire® Functions”)
- Pulse Width Modulation (PWM) (2.7 “Pulse Width Modulation Functions”)
- SPI™ (2.8 “SPI™ Functions”)
- Timer (2.9 “Timer Functions”)
- USART (2.10 “USART Functions”)

2.2 A/D CONVERTER FUNCTIONS

The A/D peripheral is supported with the following functions:

Function	Description
BusyADC	Is A/D converter currently performing a conversion?
CloseADC	Disable the A/D converter.
ConvertADC	Start an A/D conversion.
OpenADC	Configure the A/D convertor.
ReadADC	Read the results of an A/D conversion.
SetChanADC	Select A/D channel to be used.

2.2.1 Function Descriptions

BusyADC

Function: Is the A/D converter currently performing a conversion?

Include: adc.h

Prototype:
```
char BusyADC( void );
```

Remarks: This function indicates if the A/D peripheral is in the process of converting a value.

Return Value: 1 if the A/D peripheral is performing a conversion.
0 if the A/D peripheral isn't performing a conversion.

File Name: adcbusy.c

CloseADC

Function: Disable the A/D converter.

Include: adc.h

Prototype:
```
void CloseADC( void );
```

Remarks: This function disables the A/D convertor and A/D interrupt mechanism.

File Name: adcclose.c

ConvertADC

Function: Starts the A/D conversion process.

Include: adc.h

Prototype:
```
void ConvertADC( void );
```

Remarks: This function starts an A/D conversion. The BusyADC() function may be used to detect completion of the conversion.

File Name: adcconv.c

OpenADC
PIC18CXX2, PIC18FXX2, PIC18FXX8

Function: Configure the A/D convertor.

Include: adc.h

Prototype:
```
void OpenADC( unsigned char config,
              unsigned char config2 );
```

Arguments:

config

A bitmask that is created by performing a bitwise AND operation ('&') with a value from each of the categories listed below. These values are defined in the file adc.h.

A/D clock source:

ADC_FOSC_2	FOSC / 2
ADC_FOSC_4	FOSC / 4
ADC_FOSC_8	FOSC / 8
ADC_FOSC_16	FOSC / 16
ADC_FOSC_32	FOSC / 32
ADC_FOSC_64	FOSC / 64
ADC_FOSC_RC	Internal RC Oscillator

A/D result justification:

ADC_RIGHT_JUST	Result in Least Significant bits
ADC_LEFT_JUST	Result in Most Significant bits

A/D voltage reference source:

ADC_8ANA_0REF	VREF+=VDD, VREF-=VSS, All analog channels
ADC_7ANA_1REF	AN3=VREF+, All analog channels except AN3
ADC_6ANA_2REF	AN3=VREF+, AN2=VREF
ADC_6ANA_0REF	VREF+=VDD, VREF-=VSS
ADC_5ANA_1REF	AN3=VREF+, VREF-=VSS
ADC_5ANA_0REF	VREF+=VDD, VREF-=VSS
ADC_4ANA_2REF	AN3=VREF+, AN2=VREF-
ADC_4ANA_1REF	AN3=VREF+
ADC_3ANA_2REF	AN3=VREF+, AN2=VREF-
ADC_3ANA_0REF	VREF+=VDD, VREF-=VSS
ADC_2ANA_2REF	AN3=VREF+, AN2=VREF-
ADC_2ANA_1REF	AN3=VREF+
ADC_1ANA_2REF	AN3=VREF+, AN2=VREF-, AN0=A
ADC_1ANA_0REF	AN0 is analog input
ADC_0ANA_0REF	All digital I/O

config2

A bitmask that is created by performing a bitwise AND operation ('&') with a value from each of the categories listed below. These values are defined in the file adc.h.

OpenADC
PIC18CXX2, PIC18FXX2, PIC18FXX8 (Continued)

Channel:

ADC_CH0	Channel 0
ADC_CH1	Channel 1
ADC_CH2	Channel 2
ADC_CH3	Channel 3
ADC_CH4	Channel 4
ADC_CH5	Channel 5
ADC_CH6	Channel 6
ADC_CH7	Channel 7

A/D Interrupts:

ADC_INT_ON	Interrupts enabled
ADC_INT_OFF	Interrupts disabled

Remarks: This function resets the A/D peripheral to the POR state and configures the A/D-related special function registers (SFRs) according to the options specified.

File Name: adcopen.c

Code Example:

```
OpenADC( ADC_FOSC_32    &
         ADC_RIGHT_JUST &
         ADC_1ANA_0REF,
         ADC_CH0        &
         ADC_INT_OFF    );
```

OpenADC
PIC18C658/858, PIC18C601/801, PIC18F6X20, PIC18F8X20

Function: Configure the A/D convertor.

Include: adc.h

Prototype:

```
void OpenADC( unsigned char config,
              unsigned char config2 );
```

Arguments: ***config***

A bitmask that is created by performing a bitwise AND operation ('&') with a value from each of the categories listed below. These values are defined in the file adc.h.

A/D clock source:

ADC_FOSC_2	FOSC / 2
ADC_FOSC_4	FOSC / 4
ADC_FOSC_8	FOSC / 8
ADC_FOSC_16	FOSC / 16
ADC_FOSC_32	FOSC / 32
ADC_FOSC_64	FOSC / 64
ADC_FOSC_RC	Internal RC Oscillator

A/D result justification:

ADC_RIGHT_JUST	Result in Least Significant bits
ADC_LEFT_JUST	Result in Most Significant bits

OpenADC
PIC18C658/858, PIC18C601/801, PIC18F6X20, PIC18F8X20 (Continued)

A/D port configuration:

ADC_0ANA	All digital	
ADC_1ANA	analog:AN0	digital:AN1-AN15
ADC_2ANA	analog:AN0-AN1	digital:AN2-AN15
ADC_3ANA	analog:AN0-AN2	digital:AN3-AN15
ADC_4ANA	analog:AN0-AN3	digital:AN4-AN15
ADC_5ANA	analog:AN0-AN4	digital:AN5-AN15
ADC_6ANA	analog:AN0-AN5	digital:AN6-AN15
ADC_7ANA	analog:AN0-AN6	digital:AN7-AN15
ADC_8ANA	analog:AN0-AN7	digital:AN8-AN15
ADC_9ANA	analog:AN0-AN8	digital:AN9-AN15
ADC_10ANA	analog:AN0-AN9	digital:AN10-AN15
ADC_11ANA	analog:AN0-AN10	digital:AN11-AN15
ADC_12ANA	analog:AN0-AN11	digital:AN12-AN15
ADC_13ANA	analog:AN0-AN12	digital:AN13-AN15
ADC_14ANA	analog:AN0-AN13	digital:AN14-AN15
ADC_15ANA	All analog	

config2

A bitmask that is created by performing a bitwise AND operation ('&') with a value from each of the categories listed below. These values are defined in the file `adc.h`.

Channel:

ADC_CH0	Channel 0
ADC_CH1	Channel 1
ADC_CH2	Channel 2
ADC_CH3	Channel 3
ADC_CH4	Channel 4
ADC_CH5	Channel 5
ADC_CH6	Channel 6
ADC_CH7	Channel 7
ADC_CH8	Channel 8
ADC_CH9	Channel 9
ADC_CH10	Channel 10
ADC_CH11	Channel 11
ADC_CH12	Channel 12
ADC_CH13	Channel 13
ADC_CH14	Channel 14
ADC_CH15	Channel 15

A/D Interrupts:

ADC_INT_ON	Interrupts enabled
ADC_INT_OFF	Interrupts disabled

A/D voltage configuration:

ADC_VREFPLUS_VDD	V_{REF+} = AV_{DD}
ADC_VREFPLUS_EXT	V_{REF+} = external
ADC_VREFMINUS_VDD	V_{REF-} = AV_{DD}
ADC_VREFMINUS_EXT	V_{REF-} = external

OpenADC
PIC18C658/858, PIC18C601/801, PIC18F6X20, PIC18F8X20 (Continued)

Remarks: This function resets the A/D-related registers to the POR state and then configures the clock, result format, voltage reference, port and channel.

File Name: adcopen.c

Code Example:
```
OpenADC( ADC_FOSC_32    &
         ADC_RIGHT_JUST &
         ADC_14ANA,
         ADC_CH0        &
         ADC_INT_OFF    );
```

OpenADC
PIC18F1X20, PIC18F2X20, PIC18F4X20

Function: Configure the A/D convertor.

Include: adc.h

Prototype:
```
void OpenADC(unsigned char config,
             unsigned char config2 ,
             unsigned char portconfig);
```

Arguments:

config

A bitmask that is created by performing a bitwise AND operation ('&') with a value from each of the categories listed below. These values are defined in the file adc.h.

A/D clock source:

ADC_FOSC_2	FOSC / 2
ADC_FOSC_4	FOSC / 4
ADC_FOSC_8	FOSC / 8
ADC_FOSC_16	FOSC / 16
ADC_FOSC_32	FOSC / 32
ADC_FOSC_64	FOSC / 64
ADC_FOSC_RC	Internal RC Oscillator

A/D result justification:

ADC_RIGHT_JUST	Result in Least Significant bits
ADC_LEFT_JUST	Result in Most Significant bits

A/D acquisition time select:

ADC_0_TAD	0 Tad
ADC_2_TAD	2 Tad
ADC_4_TAD	4 Tad
ADC_6_TAD	6 Tad
ADC_8_TAD	8 Tad
ADC_12_TAD	12 Tad
ADC_16_TAD	16 Tad
ADC_20_TAD	20 Tad

config2

A bitmask that is created by performing a bitwise AND operation ('&') with a value from each of the categories listed below. These values are defined in the file adc.h.

OpenADC
PIC18F1X20, PIC18F2X20, PIC18F4X20 (Continued)

Channel:

ADC_CH0	Channel 0
ADC_CH1	Channel 1
ADC_CH2	Channel 2
ADC_CH3	Channel 3
ADC_CH4	Channel 4
ADC_CH5	Channel 5
ADC_CH6	Channel 6
ADC_CH7	Channel 7
ADC_CH8	Channel 8
ADC_CH9	Channel 9
ADC_CH10	Channel 10
ADC_CH11	Channel 11
ADC_CH12	Channel 12
ADC_CH13	Channel 13
ADC_CH14	Channel 14
ADC_CH15	Channel 15

A/D Interrupts:

ADC_INT_ON	Interrupts enabled
ADC_INT_OFF	Interrupts disabled

A/D voltage configuration:

ADC_VREFPLUS_VDD	VREF+ = AVDD
ADC_VREFPLUS_EXT	VREF+ = external
ADC_VREFMINUS_VDD	VREF- = AVDD
ADC_VREFMINUS_EXT	VREF- = external

portconfig

The value of portconfig is any value from 0 to 127 for the PIC18F1220/1320 and 0 to 15 for the PIC18F2220/2320/4220/4320, inclusive. This is the value of bits 0 through 6 or bits 0 through 3 of the ADCON1 register, which are the port configuration bits.

Remarks: This function resets the A/D-related registers to the POR state and then configures the clock, result format, voltage reference, port and channel.

File Name: adcopen.c

Code Example:

```
OpenADC( ADC_FOSC_32    &
         ADC_RIGHT_JUST &
         ADC_12_TAD,
         ADC_CH0        &
         ADC_INT_OFF, 15  );
```

ReadADC

Function: Read the result of an A/D conversion.

Include: `adc.h`

Prototype:
```
int ReadADC( void );
```

Remarks: This function reads the 16-bit result of an A/D conversion.

Return Value: This function returns the 16-bit signed result of the A/D conversion. Based on the configuration of the A/D converter (e.g., using the `OpenADC()` function), the result will be contained in the Least Significant or Most Significant bits of the 16-bit result.

File Name: `adcread.c`

SetChanADC

Function: Select the channel used as input to the A/D converter.

Include: `adc.h`

Prototype:
```
void SetChanADC( unsigned char channel );
```

Arguments: ***channel***

One of the following values (defined in `adc.h`):

`ADC_CH0`	Channel 0
`ADC_CH1`	Channel 1
`ADC_CH2`	Channel 2
`ADC_CH3`	Channel 3
`ADC_CH4`	Channel 4
`ADC_CH5`	Channel 5
`ADC_CH6`	Channel 6
`ADC_CH7`	Channel 7
`ADC_CH8`	Channel 8
`ADC_CH9`	Channel 9
`ADC_CH10`	Channel 10
`ADC_CH11`	Channel 11

Remarks: Selects the pin that will be used as input to the A/D converter.

File Name: `adcsetch.c`

Code Example:
```
SetChanADC( ADC_CH0 );
```

2.2.2 Example Use of the A/D Converter Routines

```
#include <p18C452.h>
#include <adc.h>
#include <stdlib.h>
#include <delays.h>

int result;

void main( void )
{
  // configure A/D convertor
  OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF,
           ADC_CH0 & ADC_INT_OFF );

  Delay10TCYx( 5 );     // Delay for 50TCY
  ConvertADC();         // Start conversion
  while( BusyADC() );   // Wait for completion
  result = ReadADC();   // Read result
  CloseADC();           // Disable A/D converter
}
```

2.3 INPUT CAPTURE FUNCTIONS

The capture peripheral is supported with the following functions:

Function	Description
`CloseCapture`***x***	Disable capture peripheral ***x***.
`OpenCapture`***x***	Configure capture peripheral ***x***.
`ReadCapture`***x***	Read a value from capture peripheral ***x***.

2.3.1 Function Descriptions

CloseCapture1
CloseCapture2
CloseCapture3
CloseCapture4
CloseCapture5

Function: Disable input capture *x*.

Include: `capture.h`

Prototype:
```
void CloseCapture1( void );
void CloseCapture2( void );
void CloseCapture3( void );
void CloseCapture4( void );
void CloseCapture5( void );
```

Remarks: This function disables the interrupt corresponding to the specified input capture.

File Name:
```
cp1close.c
cp2close.c
cp3close.c
cp4close.c
cp5close.c
```

OpenCapture1
OpenCapture2
OpenCapture3
OpenCapture4
OpenCapture5

Function: Configure and enable input capture *x*.

Include: `capture.h`

Prototype:
```
void OpenCapture1( unsigned char config );
void OpenCapture2( unsigned char config );
void OpenCapture3( unsigned char config );
void OpenCapture4( unsigned char config );
void OpenCapture5( unsigned char config );
```

Arguments:
config
A bitmask that is created by performing a bitwise AND operation ('`&`') with a value from each of the categories listed below. These values are defined in the file `capture.h`:

OpenCapture1
OpenCapture2
OpenCapture3
OpenCapture4
OpenCapture5 (Continued)

Enable CCP Interrupts:

CAPTURE_INT_ON	Interrupts Enabled
CAPTURE_INT_OFF	Interrupts Disabled

Interrupt Trigger (replace *x* with CCP module number):

Cx_EVERY_FALL_EDGE	Interrupt on every falling edge
Cx_EVERY_RISE_EDGE	Interrupt on every rising edge
Cx_EVERY_4_RISE_EDGE	Interrupt on every 4th rising edge
Cx_EVERY_16_RISE_EDGE	Interrupt on every 16th rising edge

Remarks: This function first resets the capture module to the POR state and then configures the input capture for the specified edge detection.

The capture functions use a structure, defined in `capture.h`, to indicate overflow status of each of the capture modules. This structure is called `CapStatus` and has the following bit fields:

```
Cap1OVF
Cap2OVF
Cap3OVF
Cap4OVF
Cap5OVF
```

In addition to opening the capture, the appropriate timer module must be enabled before any of the captures will operate. See 2.9 “Timer Functions” for information on using the Timer runtime library functions for this.

File Name:

```
cp1open.c
cp2open.c
cp3open.c
cp4open.c
cp5open.c
```

Code Example:

```
OpenCapture1( CAPTURE_INT_ON &
              C1_EVERY_4_RISE_EDGE );
```

ReadCapture1
ReadCapture2
ReadCapture3
ReadCapture4
ReadCapture5

Function:	Read the result of a capture event from the specified input capture.
Include:	`capture.h`

Prototype:	`unsigned int ReadCapture1( void );`
	`unsigned int ReadCapture2( void );`
	`unsigned int ReadCapture3( void );`
	`unsigned int ReadCapture4( void );`
	`unsigned int ReadCapture5( void );`

Remarks:	This function reads the value of the respective input capture's SFRs.
Return Value:	This function returns the result of the capture event.

File Name:	`cp1read.c`
	`cp2read.c`
	`cp3read.c`
	`cp4read.c`
	`cp5read.c`

2.3.2 Example Use of the Capture Routines

This example demonstrates the use of the capture library routines in a “polled” (not interrupt-driven) environment.

```
#include <p18C452.h>
#include <capture.h>
#include <timers.h>
#include <usart.h>
#include <stdlib.h>

void main(void)
{
  unsigned int result;
  char str[7];

  // Configure Capture1
  OpenCapture1( C1_EVERY_4_RISE_EDGE &
                CAPTURE_INT_OFF );

  // Configure Timer3
  OpenTimer3( TIMER_INT_OFF &
              T3_SOURCE_INT );

  // Configure USART
  OpenUSART( USART_TX_INT_OFF  &
             USART_RX_INT_OFF  &
             USART_ASYNCH_MODE &
             USART_EIGHT_BIT   &
             USART_CONT_RX,
             25 );

  while(!PIR1bits.CCP1IF);  // Wait for event
  result = ReadCapture1(); // read result
  ultoa(result,str);       // convert to string

  // Write the string out to the USART if
  // an overflow condition has not occurred.
  if(!CapStatus.Cap1OVF)
  {
    putsUSART(str);
  }

  // Clean up
  CloseCapture1();
  CloseTimer3();
  CloseUSART();
}
```

2.4 I²C® FUNCTIONS

The I²C peripheral is supported with the following functions:

Function	Description
AckI2C	Generate I²C bus *Acknowledge* condition.
CloseI2C	Disable the SSP module.
DataRdyI2C	Is the data available in the I²C buffer?
getcI2C	Read a single byte from the I²C bus.
getsI2C	Read a string from the I²C bus operating in master I²C mode.
IdleI2C	Loop until I²C bus is idle.
NotAckI2C	Generate I²C bus *Not Acknowledge* condition.
OpenI2C	Configure the SSP module.
putcI2C	Write a single byte to the I²C bus.
putsI2C	Write a string to the I²C bus operating in either Master or Slave mode.
ReadI2C	Read a single byte from the I²C bus.
RestartI2C	Generate an I²C bus *Restart* condition.
StartI2C	Generate an I²C bus *START* condition.
StopI2C	Generate an I²C bus *STOP* condition.
WriteI2C	Write a single byte to the I²C bus.

The following functions are also provided for interfacing with an EE device such as the Microchip 24LC01B using the I²C interface:

Function	Description
EEAckPolling	Generate the Acknowledge polling sequence.
EEByteWrite	Write a single byte.
EECurrentAddRead	Read a single byte from the next location.
EEPageWrite	Write a string of data.
EERandomRead	Read a single byte from an arbitrary address.
EESequentialRead	Read a string of data.

2.4.1 Function Descriptions

AckI2C

Function: Generate I²C bus *Acknowledge* condition.

Include: `i2c.h`

Prototype:
```
void AckI2C( void );
```

Remarks: This function generates an I²C bus *Acknowledge* condition.

File Name: `acki2c.c`

CloseI2C

Function: Disable the SSP module.

Include: `i2c.h`

Prototype:
```
void CloseI2C( void );
```

Remarks: This function disables the SSP module.

File Name: `closei2c.c`

DataRdyI2C

Function: Is data available in the I^2C buffer?

Include: `i2c.h`

Prototype:
```
unsigned char DataRdyI2C( void );
```

Remarks: Determines if there is a byte to be read in the SSP buffer.

Return Value: 1 if there is data in the SSP buffer
0 if there is no data in the SSP buffer

File Name: `dtrdyi2c.c`

Code Example:
```
if (DataRdyI2C())
{
  var = getcI2C();
}
```

getcI2C

See **ReadI2C**.

getsI2C

Function: Read a fixed length string from the I^2C bus operating in master I^2C mode.

Include: `i2c.h`

Prototype:
```
unsigned char getsI2C(
            unsigned char  * rdptr,
            unsigned char    length );
```

Arguments:
rdptr
Character type pointer to PICmicro RAM for storage of data read from I^2C device.
length
Number of bytes to read from I^2C device.

Remarks: This routine reads a predefined data string length from the I^2C bus.

Return Value: 0 if all bytes have been sent
-1 if a bus collision has occurred

File Name: `getsi2c.c`

Code Example:
```
unsigned char string[15];
getsI2C(string, 15);
```

IdleI2C

Function: Loop until I^2C bus is IDLE.

Include: i2c.h

Prototype:
```
void IdleI2C( void );
```

Remarks: This function checks the state of the I^2C peripheral and waits for the bus to become available. The IdleI2C function is required since the hardware I^2C peripheral does not allow for spooling of bus sequences. The I^2C peripheral must be in an IDLE state before an I^2C operation can be initiated or a write collision will be generated.

File Name: idlei2c.c

NotAckI2C

Function: Generate I^2C bus *Not Acknowledge* condition.

Include: i2c.h

Prototype:
```
void NotAckI2C( void );
```

Remarks: This function generates an I^2C bus *Not Acknowledge* condition.

File Name: noacki2c.c

OpenI2C

Function: Configure the SSP module.

Include: i2c.h

Prototype:
```
void OpenI2C( unsigned char sync_mode,
              unsigned char slew );
```

Arguments:

sync_mode

One of the following values, defined in i2c.h:

SLAVE_7	I^2C Slave mode, 7-bit address
SLAVE_10	I^2C Slave mode, 10-bit address
MASTER	I^2C Master mode

slew

One of the following values, defined in i2c.h:

SLEW_OFF	Slew rate disabled for 100 kHz mode
SLEW_ON	Slew rate enabled for 400 kHz mode

Remarks: OpenI2C resets the SSP module to the POR state and then configures the module for Master/Slave mode and the selected slew rate.

File Name: openi2c.c

Code Example:
```
OpenI2C(MASTER, SLEW_ON);
```

putcI2C

See **WriteI2C.**

putsI2C

Function: Write a data string to the I^2C bus operating in either Master or Slave mode.

Include: `i2c.h`

Prototype:
```
unsigned char putsI2C(
                unsigned char *wrptr );
```

Arguments:
wrptr
Pointer to data that will be written to the I^2C bus.

Remarks: This routine writes a data string to the I^2C bus until a null character is reached. The null character itself is not transmitted. This routine can operate in both Master or Slave mode.

Return Value:
Master I^2C Mode:
0 if the null character was reached in the data string
-2 if the slave I^2C device responded with a *Not Ack*
-3 if a write collision occurred
Slave I^2C mode:
0 if the null character was reached in the data string
-2 if the master I^2C device responded with a *Not Ack* which terminated the data transfer

File Name: `putsi2c.c`

Code Example:
```
unsigned char string[] = "data to send";
putsI2C(string);
```

ReadI2C
getcI2C

Function: Read a single byte from the I^2C bus.

Include: `i2c.h`

Prototype:
```
unsigned char ReadI2C (void);
```

Remarks: This function reads in a single byte from the I^2C bus.

Return Value: The data byte read from the I^2C bus.

File Name: `readi2c.c`

Code Example:
```
unsigned char value;
value = ReadI2C();
```

RestartI2C

Function: Generate an I^2C bus *Restart* condition.

Include: `i2c.h`

Prototype:
```
void RestartI2C( void );
```

Remarks: This function generates an I^2C bus *Restart* condition.

File Name: `rstrti2c.c`

StartI2C

Function: Generate an I^2C bus *START* condition.

Include: `i2c.h`

Prototype:
```
void StartI2C( void );
```

Remarks: This function generates a I^2C bus *START* condition.

File Name: `starti2c.c`

StopI2C

Function: Generate I^2C bus *STOP* condition.

Include: `i2c.h`

Prototype:
```
void StopI2C( void );
```

Remarks: This function generates an I^2C bus *STOP* condition.

File Name: `stopi2c.c`

WriteI2C
putcI2C

Function: Write a single byte to the I^2C bus device.

Include: `i2c.h`

Prototype:
```
unsigned char WriteI2C(
      unsigned char data_out );
```

Arguments:
data_out
A single data byte to be written to the I^2C bus device.

Remarks: This function writes out a single data byte to the I^2C bus device.

Return Value:
0 if the write was successful
-1 if there was a write collision

File Name: `writei2c.c`

Code Example:
```
WriteI2C('a');
```

2.4.2 EE Memory Device Interface Function Descriptions

EEAckPolling

Function: Generate the Acknowledge polling sequence for Microchip EE I^2C memory devices.

Include: `i2c.h`

Prototype:
```
unsigned char EEAckPolling(
     unsigned char control );
```

Arguments:
control
EEPROM control / bus device select address byte.

Remarks: This function is used to generate the Acknowledge polling sequence for EE I^2C memory devices that utilize Acknowledge polling.

Return Value:
0 if there were no errors
-1 if there was a bus collision error
-3 if there was a write collision error

File Name: `i2ceeap.c`

Code Example:
```
temp = EEAckPolling(0xA0);
```

EEByteWrite

Function: Write a single byte to the I^2C bus.

Include: `i2c.h`

Prototype:
```
unsigned char EEByteWrite(
     unsigned char control,
     unsigned char address,
     unsigned char data );
```

Arguments:
control
EEPROM control / bus device select address byte.
address
EEPROM internal address location.
data
Data to write to EEPROM address specified in function parameter address.

Remarks: This function writes a single data byte to the I^2C bus. This routine can be used for any Microchip I^2C EE memory device which requires only 1 byte of address information.

Return Value:
0 if there were no errors
-1 if there was a bus collision error
-2 if there was a NOT ACK error
-3 if there was a write collision error

File Name: `i2ceebw.c`

Code Example:
```
temp = EEByteWrite(0xA0, 0x30, 0xA5);
```

EECurrentAddRead

Function: Read a single byte from the I^2C bus.

Include: i2c.h

Prototype:
```
unsigned int EECurrentAddRead(
     unsigned char control );
```

Arguments:
control
EEPROM control / bus device select address byte.

Remarks: This function reads in a single byte from the I^2C bus. The address location of the data to read is that of the current pointer within the I^2C EE device. The memory device contains an address counter that maintains the address of the last word accessed, incremented by one.

Return Value:
-1 if a bus collision error occurred
-2 if a NOT ACK error occurred
-3 if a write collision error occurred
Otherwise, the result is returned as an unsigned 16-bit quantity. Since the buffer itself is only 8-bits wide, this means that the Most Significant Byte will be zero and the Least Significant Byte will contain the read buffer contents.

File Name: i2ceecar.c

Code Example:
```
temp = EECurrentAddRead(0xA1);
```

EEPageWrite

Function: Write a string of data to the EE device from the I^2C bus.

Include: i2c.h

Prototype:
```
unsigned char EEPageWrite(
     unsigned char control,
     unsigned char address,
     unsigned char * wrptr );
```

Arguments:
control
EEPROM control / bus device select address byte.
address
EEPROM internal address location.
wrptr
Character type pointer in PICmicro RAM. The data objects pointed to by ***wrptr*** will be written to the EE device.

Remarks: This function writes a null terminated string of data to the I^2C EE memory device. The null character itself is not transmitted.

Return Value:
0 if there were no errors
-1 if there was a bus collision error
-2 if there was a NOT ACK error
-3 if there was a write collision error

File Name: i2ceepw.c

Code Example:
```
temp = EEPageWrite(0xA0, 0x70, wrptr);
```

EERandomRead

Function: Read a single byte from the I^2C bus.

Include: i2c.h

Prototype:
```
unsigned int EERandomRead(
     unsigned char control,
     unsigned char address );
```

Arguments:
control
EEPROM control / bus device select address byte.
address
EEPROM internal address location.

Remarks: This function reads in a single byte from the I^2C bus. The routine can be used for Microchip I^2C EE memory devices which only require 1 byte of address information.

Return Value: The return value contains the value read in the Least Significant Byte and the error condition in the Most Significant Byte. The error condition is:
-1 if there was a bus collision error
-2 if there was a NOT ACK error
-3 if there was a write collision error

File Name: i2ceerr.c

Code Example:
```
unsigned int temp;
temp = EERandomRead(0xA0,0x30);
```

EESequentialRead

Function: Read a string of data from the I^2C bus.

Include: i2c.h

Prototype:
```
unsigned char EESequentialRead(
     unsigned char control,
     unsigned char address,
     unsigned char * rdptr,
     unsigned char length );
```

Arguments:
control
EEPROM control / bus device select address byte.
address
EEPROM internal address location.
rdptr
Character type pointer to PICmicro RAM area for placement of data read from EEPROM device.
length
Number of bytes to read from EEPROM device.

Remarks: This function reads in a predefined string length of data from the I^2C bus. The routine can be used for Microchip I^2C EE memory devices which only require 1 byte of address information.

Return Value: 0 if there were no errors
-1 if there was a bus collision error
-2 if there was a NOT ACK error
-3 if there was a write collision error

EESequentialRead (Continued)

File Name: i2ceesr.c

Code Example:

```
unsigned char err;
err = EESequentialRead(0xA0,
                       0x70,
                       rdptr,
                       15);
```

2.4.3 Example of Use

The following is a simple code example illustrating the SSP module configured for I^2C master communication. The routine illustrates I^2C communications with a Microchip 24LC01B I^2C EE Memory Device.

```
#include "p18cxx.h"
#include "i2c.h"

unsigned char arraywr[] = {1,2,3,4,5,6,7,8,0};
unsigned char arrayrd[20];

//************************************************
void main(void)
{
  OpenI2C(MASTER, SLEW_ON);// Initialize I2C module
  SSPADD = 9;              //400Khz Baud clock(9) @16MHz
                           //100khz Baud clock(39) @16MHz

  while(1)
  {
    EEByteWrite(0xA0, 0x30, 0xA5);
    EEAckPolling(0xA0);
    EECurrentAddRead(0xA0);
    EEPageWrite(0xA0, 0x70, arraywr);
    EEAckPolling(0xA0);
    EESequentialRead(0xA0, 0x70, arrayrd, 20);
    EERandomRead(0xA0,0x30);
  }
}
```

2.5 I/O PORT FUNCTIONS

PORTB is supported with the following functions:

Function	Description
ClosePORTB	Disable the interrupts and internal pull-up resistors for PORTB.
CloseRB***x***INT	Disable interrupts for PORTB pin ***x*** .
DisablePullups	Disable the internal pull-up resistors on PORTB.
EnablePullups	Enable the internal pull-up resistors on PORTB.
OpenPORTB	Configure the interrupts and internal pull-up resistors on PORTB.
OpenRB***x***INT	Enable interrupts for PORTB pin ***x*** .

2.5.1 Function Descriptions

ClosePORTB

Function: Disable the interrupts and internal pull-up resistors for PORTB.

Include: portb.h

Prototype:
```
void ClosePORTB( void );
```

Remarks: This function disables the PORTB interrupt on change and the internal pull-up resistors.

File Name: pbclose.c

CloseRB0INT
CloseRB1INT
CloseRB2INT

Function: Disable the interrupts for the specified PORTB pin.

Include: portb.h

Prototype:
```
void CloseRB0INT( void );
void CloseRB1INT( void );
void CloseRB2INT( void );
```

Remarks: This function disables the PORTB interrupt-on-change.

File Name:
```
rb0close.c
rb1close.c
rb2close.c
```

DisablePullups

Function: Disable the internal pull-up resistors on PORTB.

Include: portb.h

Prototype:
```
void DisablePullups( void );
```

Remarks: This function disables the internal pull-up resistors on PORTB.

File Name: pulldis.c

EnablePullups

Function: Enable the internal pull-up resistors on PORTB.

Include: portb.h

Prototype:
```
void EnablePullups( void );
```

Remarks: This function enables the internal pull-up resistors on PORTB.

File Name: pullen.c

OpenPORTB

Function: Configure the interrupts and internal pull-up resistors on PORTB.

Include: portb.h

Prototype:
```
void OpenPORTB( unsigned char config);
```

Arguments: ***config***

A bitmask that is created by performing a bitwise AND operation ('&') with a value from each of the categories listed below. These values are defined in the file portb.h.

Interrupt-on-Change:

PORTB_CHANGE_INT_ON	Interrupt enabled
PORTB_CHANGE_INT_OFF	Interrupt disabled

Enable Pullups:

PORTB_PULLUPS_ON	pull-up resistors enabled
PORTB_PULLUPS_OFF	pull-up resistors disabled

Remarks: This function configures the interrupts and internal pull-up resistors on PORTB.

File Name: pbopen.c

Code Example:
```
OpenPORTB( PORTB_CHANGE_INT_ON & PORTB_PULLUPS_ON);
```

OpenRB0INT
OpenRB1INT
OpenRB2INT

Function: Enable interrupts for the specified PORTB pin.

Include: portb.h

Prototype:
```
void OpenRB0INT( unsigned char config );
void OpenRB1INT( unsigned char config );
void OpenRB2INT( unsigned char config );
```

OpenRB0INT
OpenRB1INT
OpenRB2INT (Continued)

Arguments: ***config***

A bitmask that is created by performing a bitwise AND operation ('&') with a value from each of the categories listed below. These values are defined in the file `portb.h`.

Interrupt-on-Change:

`PORTB_CHANGE_INT_ON`	Interrupt enabled
`PORTB_CHANGE_INT_OFF`	Interrupt disabled

Interrupt on Edge:

`RISING_EDGE_INT`	Interrupt on rising edge
`FALLING_EDGE_INT`	Interrupt on falling edge

Enable Pullups:

`PORTB_PULLUPS_ON`	pull-up resistors enabled
`PORTB_PULLUPS_OFF`	pull-up resistors disabled

Remarks: This function configures the interrupts and internal pull-up resistors on `PORTB`.

File Name:
`rb0open.c`
`rb1open.c`
`rb2open.c`

Code Example:

```
OpenRB0INT( PORTB_CHANGE_INT_ON &
PORTB_CHANGE_INT_ON & RISING_EDGE_INT &
PORTB_PULLUPS_ON);
```

2.6 MICROWIRE® FUNCTIONS

Microwire communication is supported with the following functions:

Function	Description
CloseMwire	Disable the SSP module used for Microwire communication.
DataRdyMwire	Indicate completion the internal write cycle.
getcMwire	Read a byte from the Microwire device.
getsMwire	Read a string from the Microwire device.
OpenMwire	Configure the SSP module for Microwire use.
putcMwire	Write a byte to the Microwire device.
ReadMwire	Read a byte from the Microwire device.
WriteMwire	Write a byte to the Microwire device.

2.6.1 Function Descriptions

CloseMwire

Function: Disable the SSP module.

Include: mwire.h

Prototype: void CloseMwire(void);

Remarks: Pin I/O returns under control of the TRISC and LATC register settings.

File Name: closmwir.c

DataRdyMwire

Function: Indicate whether the Microwire device has completed the internal write cycle.

Include: mwire.h

Prototype: unsigned char DataRdyMwire(void);

Remarks: Determines if Microwire device is ready.

Return Value: 1 if the Microwire device is ready
0 if the internal write cycle is not complete or a bus error occurred

File Name: drdymwir.c

Code Example:

```
while (!DataRdyMwire());
```

getcMwire

See **ReadMwire**.

getsMwire

Function: Read a string from the Microwire device.

Include: mwire.h

Prototype:
```
void getsMwire( unsigned char * rdptr,
                unsigned char length);
```

Arguments:

rdptr
Pointer to PICmicro RAM for placement of data read from Microwire device.

length
Number of bytes to read from Microwire device.

Remarks: This function is used to read a predetermined length of data from a Microwire device. Before using this function, a READ command with the appropriate address must be issued.

File Name: getsmwir.c

Code Example:
```
unsigned char arryrd[LENGTH];
putcMwire(READ);
putcMwire(address);
getsMwire(arrayrd, LENGTH);
```

OpenMwire

Function: Configure the SSP module.

Include: mwire.h

Prototype:
```
void OpenMwire(
        unsigned char sync_mode );
```

Arguments:

sync_mode
One of the following values defined in mwire.h:

Fosc_4	clock = FOSC/4
Fosc_16	clock = FOSC/16
Fosc_64	clock = FOSC/64
Fosc_TMR2	clock = TMR2 output/2

Remarks: OpenMwire resets the SSP module to the POR state and then configures the module for Microwire communications.

File Name: openmwir.c

Code Example:
```
OpenMwire(FOSC_16);
```

putcMwire

See **WriteMwire**.

ReadMwire
getcMwire

Function: Read a byte from a Microwire device.

Include: mwire.h

Prototype:
```
unsigned char ReadMwire(
        unsigned char high_byte,
        unsigned char low_byte );
```

Arguments:
high_byte
First byte of 16-bit instruction word.
low_byte
Second byte of 16-bit instruction word.

Remarks: This function reads in a single byte from a Microwire device. The START bit, opcode and address compose the high and low bytes passed into this function.

Return Value: The return value is the data byte read from the Microwire device.

File Name: readmwir.c

Code Example:
```
ReadMwire(0x03, 0x00);
```

WriteMwire
putcMwire

Function: This function is used to write out a single data byte (one character).

Include: mwire.h

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

Arguments:
data_out
Single byte of data to write to Microwire device.

Remarks: This function writes out single data byte to a Microwire device utilizing the SSP module.

Return Value: 0 if the write was successful
-1 if there was a write collision

File Name: writmwir.c

Code Example:
```
WriteMwire(0x55);
```

2.6.2 Example of Use

The following is a simple code example illustrating the SSP module communicating with a Microchip 93LC66 Microwire EE Memory Device.

```
#include "p18cxxx.h"
#include "mwire.h"

// 93LC66 x 8
// FUNCTION Prototypes
void main(void);
void ew_enable(void);
void erase_all(void);
void busy_poll(void);
void write_all(unsigned char data);
void byte_read(unsigned char address);
void read_mult(unsigned char address,
               unsigned char *rdptr,
               unsigned char length);
void write_byte(unsigned char address,
                unsigned char data);

// VARIABLE Definitions
unsigned char arrayrd[20];
unsigned char var;

// DEFINE 93LC66 MACROS -- see datasheet for details
#define  READ   0x0C
#define  WRITE  0x0A
#define  ERASE  0x0E
#define  EWEN1  0x09
#define  EWEN2  0x80
#define  ERAL1  0x09
#define  ERAL2  0x00
#define  WRAL1  0x08
#define  WRAL2  0x80
#define  EWDS1  0x08
#define  EWDS2  0x00
#define  W_CS   LATCbits.LATC2

void main(void)
{
  TRISCbits.TRISC2 = 0;
  W_CS = 0;               //ensure CS is negated
  OpenMwire(FOSC_16);     //enable SSP perpiheral
  ew_enable();            //send erase/write enable
  write_byte(0x13, 0x34); //write byte (address,data)
  busy_poll();
  Nop();
  byte_read(0x13);        //read single byte (address)
  read_mult(0x10, arrayrd, 10); //read multiple bytes
  erase_all();                  //erase entire array
  CloseMwire();                 //disable SSP peripheral
}

void ew_enable(void)
{
   W_CS = 1;         //assert chip select
   putcMwire(EWEN1); //enable write command byte 1
   putcMwire(EWEN2); //enable write command byte 2
   W_CS = 0;         //negate chip select
}
```

```
void busy_poll(void)
{
  W_CS = 1;
  while(! DataRdyMwire() );
  W_CS = 0;
}

void write_byte(unsigned char address,
                unsigned char data)
{
  W_CS = 1;
  putcMwire(WRITE);    //write command
  putcMwire(address);  //address
  putcMwire(data);     //write single byte
  W_CS = 0;
}

void byte_read(unsigned char address)
{
  W_CS = 1;
  getcMwire(READ,address);  //read one byte
  W_CS = 0;
}

void read_mult(unsigned char address,
               unsigned char *rdptr,
               unsigned char length)
{
  W_CS = 1;
  putcMwire(READ);          //read command
  putcMwire(address);       //address (A7 - A0)
  getsMwire(rdptr, length); //read multiple bytes
  W_CS = 0;
}

void erase_all(void)
{
  W_CS = 1;
  putcMwire(ERAL1); //erase all command byte 1
  putcMwire(ERAL2); //erase all command byte 2
  W_CS = 0;
}
```

2.7 PULSE WIDTH MODULATION FUNCTIONS

The PWM peripheral is supported with the following functions:

Function	Description
ClosePWM*x*	Disable PWM channel *x*.
OpenPWM*x*	Configure PWM channel *x*.
SetDCPWM*x*	Write a new duty cycle value to PWM channel *x*.
SetOutputPWM*x*	Sets the PWM output configuration bits for ECCP.

ClosePWM1
ClosePWM2

Function: Disable PWM channel.

Include: pwm.h

Prototype:
```
void ClosePWM1( void );
void ClosePWM2( void );
```

Remarks: This function disables the specified PWM channel.

File Name:
pw1close.c
pw2close.c

OpenPWM1
OpenPWM2

Function: Configure PWM channel.

Include: pwm.h

Prototype:
```
void OpenPWM1( char period );
void OpenPWM2( char period );
```

Arguments: *period*
Can be any value from 0x00 to 0xff. This value determines the PWM frequency by using the following formula:
PWM period = [(*period*) + 1] x 4 x TOSC x
TMR2 prescaler

Remarks: This function configures the specified PWM channel for period and for time-base. PWM uses only Timer2.

In addition to opening the PWM, Timer2 must also be opened with an **OpenTimer2(...)** statement before the PWM will operate.

File Name:
pw1open.c
pw2open.c

Code Example:
```
OpenPWM1(0xff);
```

SetDCPWM1
SetDCPWM2

Function: Write a new duty cycle value to the specified PWM channel duty-cycle registers.

Include: pwm.h

Prototype:
```
void SetDCPWM1( unsigned int dutycycle );
void SetDCPWM2( unsigned int dutycycle );
```

Arguments: ***dutycycle***
The value of *dutycycle* can be any 10-bit number. Only the lower 10-bits of *dutycycle* are written into the duty cycle registers. The duty cycle, or more specifically the high time of the PWM waveform, can be calculated from the following formula:

PWM x Duty cycle = (DCx<9:0>) x TOSC

where DCx<9:0> is the 10-bit value specified in the call to this function.

Remarks: This function writes the new value for *dutycycle* to the specified PWM channel duty cycle registers.

The maximum resolution of the PWM waveform can be calculated from the period using the following formula:

Resolution (bits) = log(FOSC/Fpwm) / log(2)

File Name: pw1setdc.c
pw2setdc.c

Code Example:
```
SetDCPWM1(0);
```

SetOutputPWM1
PIC18F1X20, PIC18F4X20

Function: Sets the PWM output configuration bits for ECCP.

Include: pwm.h

Prototype:
```
void SetOutputPWM1 (unsigned char
outputconfig, unsigned char outputmode);
```

Arguments: ***outputconfig***
The value of outputconfig can be any one of the following values (defined in pwm.h):

SINGLE_OUT	single output
FULL_OUT_FWD	full-bridge output foward
HALF_OUT	half-bridge output
FULL_OUT_REV	full-bridge output reverse

outputmode
The value of outputmode can be any one of the following values (defined in pwm.h):

PWM_MODE_1	P1A and P1C active high, P1B and P1D active high
PWM_MODE_2	P1A and P1C active high, P1B and P1D active low
PWM_MODE_3	P1A and P1C active low, P1B and P1D active high
PWM_MODE_4	P1A and P1C active low, P1B and P1D active low

SetOutputPWM1
PIC18F1X20, PIC18F4X20 (Continued)

Remarks:	This is only applicable to those devices with extended CCP (ECCP).
File Name:	`pw1setoc.c`
Code Example:	`SetOutputPWM1 (SINGLE_OUT, PWM_MODE_1);`

2.8 SPI™ FUNCTIONS

SPI communication is supported with the following functions:

Function	Description
CloseSPI	Disable the SSP module used for SPI communications.
DataRdySPI	Determine if a new value is available from the SPI buffer.
getcSPI	Read a byte from the SPI bus.
getsSPI	Read a string from the SPI bus.
OpenSPI	Initialize the SSP module used for SPI communications.
putcSPI	Write a byte to the SPI bus.
putsSPI	Write a string to the SPI bus.
ReadSPI	Read a byte from the SPI bus.
WriteSPI	Write a byte to the SPI bus.

2.8.1 Function Descriptions

CloseSPI

Function: Disable the SSP module.

Include: spi.h

Prototype:
```
void CloseSPI( void );
```

Remarks: This function disables the SSP module. Pin I/O returns under the control of the TRISC and LATC Registers.

File Name: closespi.c

DataRdySPI

Function: Determine if the SSPBUF contains data.

Include: spi.h

Prototype:
```
unsigned char DataRdySPI( void );
```

Remarks: This function determines if there is a byte to be read from the SSPBUF register.

Return Value: 0 if there is no data in the SSPBUF register
1 if there is data in the SSPBUF register

File Name: dtrdyspi.c

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

getcSPI

See **ReadSPI**.

getsSPI

Function: Read a string from the SPI bus.

Include: spi.h

Prototype:
```
void getsSPI( unsigned char *rdptr,
              unsigned char length );
```

Arguments:

rdptr
Pointer to location to store data read from SPI device.

length
Number of bytes to read from SPI device.

Remarks: This function reads in a predetermined data string length from the SPI bus.

File Name: getsspi.c

Code Example:
```
unsigned char wrptr(10);
getsSPI(wrptr, 10);
```

OpenSPI

Function: Initialize the SSP module.

Include: spi.h

Prototype:
```
void OpenSPI( unsigned char sync_mode,
              unsigned char bus_mode,
              unsigned char smp_phase);
```

Arguments:

sync_mode
One of the following values, defined in spi.h:

FOSC_4	SPI Master mode, clock = FOSC/4
FOSC_16	SPI Master mode, clock = FOSC/16
FOSC_64	SPI Master mode, clock = FOSC/64
FOSC_TMR2	SPI Master mode, clock = TMR2 output/2
SLV_SSON	SPI Slave mode, /SS pin control enabled
SLV_SSOFF	SPI Slave mode, /SS pin control disabled

bus_mode
One of the following values, defined in spi.h:

MODE_00	Setting for SPI bus Mode 0,0
MODE_01	Setting for SPI bus Mode 0,1
MODE_10	Setting for SPI bus Mode 1,0
MODE_11	Setting for SPI bus Mode 1,1

smp_phase
One of the following values, defined in spi.h:

SMPEND	Input data sample at end of data out
SMPMID	Input data sample at middle of data out

Remarks: This function sets up the SSP module for use with a SPI bus device.

File Name: openspi.c

Code Example:
```
OpenSPI(FOSC_16, MODE_00, SMPEND);
```

putcSPI

See **WriteSPI**.

putsSPI

Function: Write a string to the SPI bus.

Include: `spi.h`

Prototype:
```
void putsSPI( unsigned char *wrptr );
```

Arguments: ***wrptr***
Pointer to value that will be written to the SPI bus.

Remarks: This function writes out a data string to the SPI bus device. The routine is terminated by reading a null character in the data string (the null character is not written to the bus).

File Name: `putsspi.c`

Code Example:
```
unsigned char wrptr[] = "Hello!";
putsSPI(wrptr);
```

ReadSPI
getcSPI

Function: Read a byte from the SPI bus.

Include: `spi.h`

Prototype:
```
unsigned char ReadSPI( void );
```

Remarks: This function initiates a SPI bus cycle for the acquisition of a byte of data.

Return Value: This function returns a byte of data read during a SPI read cycle.

File Name: `readspi.c`

Code Example:
```
char x;
x = ReadSPI();
```

WriteSPI
putcSPI

Function: Write a byte to the SPI bus.

Include: `spi.h`

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

Arguments: ***data_out***
Value to be written to the SPI bus.

Remarks: This function writes a single data byte out and then checks for a write collision.

Return Value: 0 if no write collision occurred
-1 if a write collision occurred

File Name: `writespi.c`

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

2.8.2 Example of Use

The following example demonstrates the use of SSP module to communicate with a Microchip 24C080 SPI EE Memory Device.

```
#include <p18cxxx.h>
#include <spi.h>

// FUNCTION Prototypes
void main(void);
void set_wren(void);
void busy_polling(void);
unsigned char status_read(void);
void status_write(unsigned char data);
void byte_write(unsigned char addhigh,
                unsigned char addlow,
                unsigned char data);
void page_write(unsigned char addhigh,
                unsigned char addlow,
                unsigned char *wrptr);
void array_read(unsigned char addhigh,
                unsigned char addlow,
                unsigned char *rdptr,
                unsigned char count);
unsigned char byte_read(unsigned char addhigh,
                        unsigned char addlow);

// VARIABLE Definitions
unsigned char arraywr[] = {1,2,3,4,5,6,7,8,9,10,11,
                           12,13,14,15,16,0};

//24C040/080/160 page write size
unsigned char arrayrd[16];
unsigned char var;

#define SPI_CS  LATCbits.LATC2

//***********************************************
void main(void)
{
  TRISCbits.TRISC2 = 0;
  SPI_CS = 1;  // ensure SPI memory device
               // Chip Select is reset
  OpenSPI(FOSC_16, MODE_00, SMPEND);
  set_wren();
  status_write(0);

  busy_polling();
  set_wren();
  byte_write(0x00, 0x61, 'E');

  busy_polling();
  var = byte_read(0x00, 0x61);

  set_wren();
  page_write(0x00, 0x30, arraywr);
  busy_polling();

  array_read(0x00, 0x30, arrayrd, 16);
  var = status_read();
```

```
  CloseSPI();
  while(1);
}

void set_wren(void)
{
  SPI_CS = 0;               //assert chip select
  var = putcSPI(SPI_WREN);  //send write enable command
  SPI_CS = 1;               //negate chip select
}

void page_write (unsigned char addhigh,
                 unsigned char addlow,
                 unsigned char *wrptr)
{
  SPI_CS = 0;                 //assert chip select
  var = putcSPI(SPI_WRITE);   //send write command
  var = putcSPI(addhigh);     //send high byte of address
  var = putcSPI(addlow);      //send low byte of address
  putsSPI(wrptr);             //send data byte
  SPI_CS = 1;                 //negate chip select
}

void array_read (unsigned char addhigh,
                 unsigned char addlow,
                 unsigned char *rdptr,
                 unsigned char count)
{
  SPI_CS = 0;              //assert chip select
  var = putcSPI(SPI_READ); //send read command
  var = putcSPI(addhigh);  //send high byte of address
  var = putcSPI(addlow);   //send low byte of address
  getsSPI(rdptr, count);   //read multiple bytes
  SPI_CS = 1;
}

void byte_write (unsigned char addhigh,
                 unsigned char addlow,
                 unsigned char data)
{
  SPI_CS = 0;               //assert chip select
  var = putcSPI(SPI_WRITE); //send write command
  var = putcSPI(addhigh);   //send high byte of address
  var = putcSPI(addlow);    //send low byte of address
  var = putcSPI(data);      //send data byte
  SPI_CS = 1;               //negate chip select
}

unsigned char byte_read (unsigned char addhigh,
                         unsigned char addlow)
{
  SPI_CS = 0;               //assert chip select
  var = putcSPI(SPI_READ);  //send read command
  var = putcSPI(addhigh);   //send high byte of address
  var = putcSPI(addlow);    //send low byte of address
  var = getcSPI();          //read single byte
  SPI_CS = 1;
  return (var);
}
```

```
unsigned char status_read (void)
{
  SPI_CS = 0;               //assert chip select
  var = putcSPI(SPI_RDSR); //send read status command
  var = getcSPI();          //read data byte
  SPI_CS = 1;               //negate chip select
  return (var);
}

void status_write (unsigned char data)
{
  SPI_CS = 0;
  var = putcSPI(SPI_WRSR); //write status command
  var = putcSPI(data);     //status byte to write
  SPI_CS = 1;               //negate chip select
}

void busy_polling (void)
{
  do
  {
    SPI_CS = 0;               //assert chip select
    var = putcSPI(SPI_RDSR); //send read status command
    var = fetcSPI();          //read data byte
    SPI_CS = 1;               //negate chip select
  } while (var & 0x01);      //stay in loop until !busy
}
```

2.9 TIMER FUNCTIONS

The timer peripherals are supported with the following functions:

Function	Description
CloseTimer***x***	Disable timer ***x***.
OpenTimer***x***	Configure timer ***x***.
ReadTimer***x***	Read the value of timer ***x***.
WriteTimer***x***	Write a value into timer ***x***.

2.9.1 Function Descriptions

CloseTimer0
CloseTimer1
CloseTimer2
CloseTimer3
CloseTimer4

Function: Disable the specified timer.

Include: timers.h

Prototype:
```
void CloseTimer0( void );
void CloseTimer1( void );
void CloseTimer2( void );
void CloseTimer3( void );
void CloseTimer4( void );
```

Remarks: This function disables the interrupt and the specified timer.

File Name:
t0close.c
t1close.c
t2close.c
t3close.c
t4close.c

OpenTimer0

Function: Configure timer0.

Include: timers.h

Prototype: `void OpenTimer0( unsigned char config );`

Arguments: ***config***
A bitmask that is created by performing a bitwise AND operation ('&') with a value from each of the categories listed below. These values are defined in the file timers.h.

Enable Timer0 Interrupt:

TIMER_INT_ON	Interrupt enabled
TIMER_INT_OFF	Interrupt disabled

OpenTimer0 (Continued)

Timer Width:

T0_8BIT	8-bit mode
T0_16BIT	16-bit mode

Clock Source:

T0_SOURCE_EXT	External clock source (I/O pin)
T0_SOURCE_INT	Internal clock source (TOSC)

External Clock Trigger (for T0_SOURCE_EXT):

T0_EDGE_FALL	External clock on falling edge
T0_EDGE_RISE	External clock on rising edge

Prescale Value:

T0_PS_1_1	1:1 prescale
T0_PS_1_2	1:2 prescale
T0_PS_1_4	1:4 prescale
T0_PS_1_8	1:8 prescale
T0_PS_1_16	1:16 prescale
T0_PS_1_32	1:32 prescale
T0_PS_1_64	1:64 prescale
T0_PS_1_128	1:128 prescale
T0_PS_1_256	1:256 prescale

Remarks: This function configures timer0 according to the options specified.

File Name: t0open.c

Code Example:

```
OpenTimer0( TIMER_INT_OFF &
            T0_8BIT &
            T0_SOURCE_INT &
            T0_PS_1_32 );
```

OpenTimer1

Function: Configure timer1.

Include: timers.h

Prototype: `void OpenTimer1( unsigned char config );`

Arguments: *config*

A bitmask that is created by performing a bitwise AND operation ('&') with a value from each of the categories listed below. These values are defined in the file timers.h.

Enable Timer1 Interrupt:

TIMER_INT_ON	Interrupt enabled
TIMER_INT_OFF	Interrupt disabled

OpenTimer1 (Continued)

Timer Width:

T1_8BIT_RW	8-bit mode
T1_16BIT_RW	16-bit mode

Clock Source:

T1_SOURCE_EXT	External clock source (I/O pin)
T1_SOURCE_INT	Internal clock source (TOSC)

Prescaler:

T1_PS_1_1	1:1 prescale
T1_PS_1_2	1:2 prescale
T1_PS_1_4	1:4 prescale
T1_PS_1_8	1:8 prescale

Oscillator Use:

T1_OSC1EN_ON	Enable Timer1 oscillator
T1_OSC1EN_OFF	Disable Timer1 oscillator

Synchronize Clock Input:

T1_SYNC_EXT_ON	Sync external clock input
T1_SYNC_EXT_OFF	Don't sync external clock input

Remarks: This function configures timer1 according to the options specified.

File Name: t1open.c

Code Example:

```
OpenTimer1( TIMER_INT_ON    &
            T1_8BIT_RW      &
            T1_SOURCE_EXT   &
            T1_PS_1_1       &
            T1_OSC1EN_OFF   &
            T1_SYNC_EXT_OFF &
            T1_SOURCE_CCP   );
```

OpenTimer2

Function: Configure timer2.

Include: timers.h

Prototype: `void OpenTimer2( unsigned char config );`

Arguments:

config

A bitmask that is created by performing a bitwise AND operation ('&') with a value from each of the categories listed below. These values are defined in the file timers.h.

Enable Timer2 Interrupt:

TIMER_INT_ON	Interrupt enabled
TIMER_INT_OFF	Interrupt disabled

Prescale Value:

T2_PS_1_1	1:1 prescale
T2_PS_1_4	1:4 prescale
T2_PS_1_16	1:16 prescale

Postscale Value:

T2_POST_1_1	1:1 postscale
T2_POST_1_2	1:2 postscale
:	:
T2_POST_1_15	1:15 postscale
T2_POST_1_16	1:16 postscale

OpenTimer2 (Continued)

Remarks: This function configures timer2 according to the options specified.

File Name: `t2open.c`

Code Example:

```
OpenTimer2( TIMER_INT_OFF &
            T2_PS_1_1     &
            T2_POST_1_8   );
```

OpenTimer3

Function: Configure timer3.

Include: `timers.h`

Prototype: `void OpenTimer3( unsigned char config );`

Arguments: ***config***

A bitmask that is created by performing a bitwise AND operation ('&') with a value from each of the categories listed below. These values are defined in the file `timers.h`.

Enable Timer3 Interrupt:

`TIMER_INT_ON`	Interrupt enabled
`TIMER_INT_OFF`	Interrupt disabled

Timer Width:

`T3_8BIT_RW`	8-bit mode
`T3_16BIT_RW`	16-bit mode

Clock Source:

`T3_SOURCE_EXT`	External clock source (I/O pin)
`T3_SOURCE_INT`	Internal clock source (TOSC)

Prescale Value:

`T3_PS_1_1`	1:1 prescale
`T3_PS_1_2`	1:2 prescale
`T3_PS_1_4`	1:4 prescale
`T3_PS_1_8`	1:8 prescale

Synchronize Clock Input:

`T3_SYNC_EXT_ON`	Sync external clock input
`T3_SYNC_EXT_OFF`	Don't sync external clock input

Use With CCP:

Use With CCP:

`T1_SOURCE_CCP`	Timer1 source for both CCP's
`T3_SOURCE_CCP`	Timer3 source for both CCP's
`T1_CCP1_T3_CCP2`	Timer1 source for CCP1 and Timer3 source for CCP2

Remarks: This function configures timer3 according to the options specified.

File Name: `t3open.c`

Code Example:

```
OpenTimer3( TIMER_INT_ON    &
            T3_8BIT_RW      &
            T3_SOURCE_EXT   &
            T3_PS_1_1       &
            T3_OSC1EN_OFF   &
            T3_SYNC_EXT_OFF &
            T3_SOURCE_CCP   );
```

OpenTimer4

Function: Configure timer4.

Include: `timers.h`

Prototype:
```
void OpenTimer4( unsigned char config );
```

Arguments: ***config***

A bitmask that is created by performing a bitwise AND operation ('&') with a value from each of the categories listed below. These values are defined in the file `timers.h`.

Enable Timer4 Interrupt:

`TIMER_INT_ON`	Interrupt enabled
`TIMER_INT_OFF`	Interrupt disabled

Prescale Value:

`T4_PS_1_1`	1:1 prescale
`T4_PS_1_4`	1:4 prescale
`T4_PS_1_16`	1:16 prescale

Postscale Value:

`T4_POST_1_1`	1:1 postscale
`T4_POST_1_2`	1:2 postscale
:	:
`T4_POST_1_15`	1:15 postscale
`T4_POST_1_16`	1:16 postscale

Remarks: This function configures timer4 according to the options specified.

File Name: `t4open.c`

Code Example:
```
OpenTimer4( TIMER_INT_OFF &
            T4_PS_1_1     &
            T4_POST_1_8   );
```

ReadTimer0
ReadTimer1
ReadTimer2
ReadTimer3
ReadTimer4

Function: Read the value of the specified timer.

Include: `timers.h`

Prototype:
```
unsigned int  ReadTimer0( void );
unsigned int  ReadTimer1( void );
unsigned char ReadTimer2( void );
unsigned int  ReadTimer3( void );
unsigned char ReadTimer4( void );
```

ReadTimer0
ReadTimer1
ReadTimer2
ReadTimer3
ReadTimer4 (Continued)

Remarks: These functions read the value of the respective timer register(s).

Timer0: `TMR0L,TMR0H`
Timer1: `TMR1L,TMR1H`
Timer2: `TMR2`
Timer3: `TMR3L,TMR3H`
Timer4: `TMR4`

Note: When using a timer in 8-bit mode that may be configured in 16-bit mode (e.g., timer0), the upper byte is not guaranteed to be zero. The user may wish to cast the result to a char for correct results. For example:

```
// Example of reading a 16-bit result
// from a 16-bit timer operating in
// 8-bit mode:
unsigned int result;
result = (unsigned char) ReadTimer0();
```

Return Value: The current value of the timer.

File Name:
`t0read.c`
`t1read.c`
`t2read.c`
`t3read.c`
`t4read.c`

WriteTimer0
WriteTimer1
WriteTimer2
WriteTimer3
WriteTime4

Function: Write a value into the specified timer.

Include: `timers.h`

Prototype:

```
void WriteTimer0( unsigned int  timer );
void WriteTimer1( unsigned int  timer );
void WriteTimer2( unsigned char timer );
void WriteTimer3( unsigned int  timer );
void WriteTimer4( unsigned char timer );
```

Arguments: ***timer***
The value that will be loaded into the specified timer.

Remarks: These functions write a value to the respective timer register(s):

Timer0: `TMR0L,TMR0H`
Timer1: `TMR1L,TMR1H`
Timer2: `TMR2`
Timer3: `TMR3L,TMR3H`
Timer4: `TMR4`

WriteTimer0
WriteTimer1
WriteTimer2
WriteTimer3
WriteTime4 (Continued)

File Name: t0write.c
t1write.c
t2write.c
t3write.c
t4write.c

Code Example: `WriteTimer0( 10000 );`

2.9.2 Example of Use

```
#include <p18C452.h>
#include <timers.h>
#include <usart.h>
#include <stdlib.h>

void main( void )
{
  int result;
  char str[7];

  // configure timer0
  OpenTimer0( TIMER_INT_OFF &
              T0_SOURCE_INT  &
              T0_PS_1_32 );

  // configure USART
  OpenUSART( USART_TX_INT_OFF  &
             USART_RX_INT_OFF  &
             USART_ASYNCH_MODE &
             USART_EIGHT_BIT   &
             USART_CONT_RX,
             25                );

  while( 1 )
  {
    while( ! PORTBbits.RB3 ); // wait for RB3 high
    result = ReadTimer0();    // read timer

    if( result > 0xc000 )     // exit loop if value
      break;                  //   is out of range

    WriteTimer0( 0 );         // restart timer

    ultoa( result, str );     // convert timer to string
    putsUSART( str );         // print string
  }

  CloseTimer0();              // close modules
  CloseUSART();
}
```

2.10 USART FUNCTIONS

The following routines are provided for devices with a single USART peripheral:

Function	Description
BusyUSART	Is the USART transmitting?
CloseUSART	Disable the USART.
DataRdyUSART	Is data available in the USART read buffer?
getcUSART	Read a byte from the USART.
getsUSART	Read a string from the USART.
OpenUSART	Configure the USART.
putcUSART	Write a byte to the USART.
putsUSART	Write a string from data memory to the USART.
putrsUSART	Write a string from program memory to the USART.
ReadUSART	Read a byte from the USART.
WriteUSART	Write a byte to the USART.

The following routines are provided for devices with multiple USART peripherals:

Function	Description
Busy***x***USART	Is USART ***x*** transmitting?
Close***x***USART	Disable USART ***x***.
DataRdy***x***USART	Is data available in the read buffer of USART ***x***?
getc***x***USART	Read a byte from USART ***x***.
gets***x***USART	Read a string from USART ***x***.
Open***x***USART	Configure USART ***x***.
putc***x***USART	Write a byte to USART ***x***.
puts***x***USART	Write a string from data memory to USART ***x***.
putrs***x***USART	Write a string from program memory to USART ***x***.
Read***x***USART	Read a byte from USART ***x***.
Write***x***USART	Write a byte to USART ***x***.

2.10.1 Function Descriptions

BusyUSART
Busy1USART
Busy2USART

Function: Is the USART transmitting?

Include: `usart.h`

Prototype:
```
char BusyUSART(  void );
char Busy1USART( void );
char Busy2USART( void );
```

Remarks: Returns a value indicating if the USART transmitter is currently busy. This function should be used prior to commencing a new transmission.
`BusyUSART` should be used on parts with a single USART peripheral. `Busy1USART` and `Busy2USART` should be used on parts with multiple USART peripherals.

Return Value: 0 if the USART transmitter is idle
1 if the USART transmitter is in use

File Name:
```
ubusy.c
u1busy.c
u2busy.c
```

Code Example:
```
while (BusyUSART());
```

CloseUSART
Close1USART
Close2USART

Function: Disable the specified USART.

Include: `usart.h`

Prototype:
```
void CloseUSART(  void );
void Close1USART( void );
void Close2USART( void );
```

Remarks: This function disables the interrupts, transmitter and receiver for the specified USART.
`CloseUSART` should be used on parts with a single USART peripheral. `Close1USART` and `Close2USART` should be used on parts with multiple USART peripherals.

File Name:
```
uclose.c
u1close.c
u2close.c
```

DataRdyUSART
DataRdy1USART
DataRdy2USART

Function: Is data available in the read buffer?

Include: `usart.h`

Prototype:
```
char DataRdyUSART(  void );
char DataRdy1USART( void );
char DataRdy2USART( void );
```

Remarks: This function returns the status of the `RCIF` flag bit in the `PIR` register.
`DataRdyUSART` should be used on parts with a single USART peripheral. `DataRdy1USART` and `DataRdy2USART` should be used on parts with multiple USART peripherals.

Return Value: 1 if data is available
0 if data is not available

File Name:
```
udrdy.c
u1drdy.c
u2drdy.c
```

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

getcUSART
getc1USART
getc2USART

See **ReadUSART**

getsUSART
gets1USART
gets2USART

Function: Read a fixed-length string of characters from the specified USART.

Include: `usart.h`

Prototype:
```
void getsUSART ( char * buffer,
                 unsigned char len );
void gets1USART ( char * buffer,
                  unsigned char len );
void gets2USART ( char * buffer,
                  unsigned char len );
```

Arguments:
buffer
A pointer to the location where incoming characters are to be stored.
len
The number of characters to read from the USART.

Remarks: This function waits for and reads *len* number of characters out of the specified USART. There is no time out when waiting for characters to arrive.
`getsUSART` should be used on parts with a single USART peripheral. `gets1USART` and `gets2USART` should be used on parts with multiple USART peripherals.

getsUSART
gets1USART
gets2USART (Continued)

File Name: ugets.c
u1gets.c
u2gets.c

Code Example:
```
char inputstr[10];
getsUSART( inputstr, 5 );
```

OpenUSART
Open1USART
Open2USART

Function: Configure the specified USART module.

Include: usart.h

Prototype:
```
void OpenUSART( unsigned char config,
                char spbrg);
void Open1USART( unsigned char config,
                 char spbrg);
void Open2USART( unsigned char config,
                 char spbrg);
```

Arguments: *config*
A bitmask that is created by performing a bitwise AND operation ('&') with a value from each of the categories listed below. These values are defined in the file usart.h.

Interrupt on Transmission:

USART_TX_INT_ON	Transmit interrupt ON
USART_TX_INT_OFF	Transmit interrupt OFF

Interrupt on Receipt:

USART_RX_INT_ON	Receive interrupt ON
USART_RX_INT_OFF	Receive interrupt OFF

USART Mode:

USART_ASYNCH_MODE	Asynchronous Mode
USART_SYNCH_MODE	Synchronous Mode

Transmission Width:

USART_EIGHT_BIT	8-bit transmit/receive
USART_NINE_BIT	9-bit transmit/receive

Slave/Master Select*:

USART_SYNC_SLAVE	Synchronous Slave mode
USART_SYNC_MASTER	Synchronous Master mode

Reception mode:

USART_SINGLE_RX	Single reception
USART_CONT_RX	Continuous reception

Baud rate*:

USART_BRGH_HIGH	High baud rate
USART_BRGH_LOW	Low baud rate

* Applies to Synchronous mode only

OpenUSART
Open1USART
Open2USART (Continued)

spbrg
This is the value that is written to the baud rate generator register which determines the baud rate at which the USART operates. The formulas for baud rate are:

Asynchronous mode, high speed:
FOSC / (64 * (***spbrg*** + 1))

Asynchronous mode, low speed:
FOSC / (16 * (***spbrg*** + 1))

Synchronous mode:
FOSC / (4 * (***spbrg*** + 1))

Where FOSC is the oscillator frequency.

Remarks: This function configures the USART module according to the specified configuration options.
`OpenUSART` should be used on parts with a single USART peripheral. `Open1USART` and `Open2USART` should be used on parts with multiple USART peripherals.

File Name:
`uopen.c`
`u1open.c`
`u2open.c`

Code Example:
```
OpenUSART1( USART_TX_INT_OFF  &
            USART_RX_INT_OFF  &
            USART_ASYNCH_MODE &
            USART_EIGHT_BIT   &
            USART_CONT_RX     &
            USART_BRGH_HIGH,
            25                );
```

putcUSART
putc1USART
putc2USART

See **WriteUSART**

putsUSART
puts1USART
puts2USART
putrsUSART
putrs1USART
putrs2USART

Function: Writes a string of characters to the USART including the null character.

Include: usart.h

Prototype:
```
void putsUSART(  char *data );
void puts1USART( char *data );
void puts2USART( char *data );
void putrsUSART(  const rom char *data );
void putrs1USART( const rom char *data );
void putrs2USART( const rom char *data );
```

Arguments:
data
Pointer to a null-terminated string of data.

Remarks: This function writes a string of data to the USART including the null character.
Strings located in data memory should be used with the “puts” versions of these functions.
Strings located in program memory, including string literals, should be used with the “putrs” versions of these functions.
`putsUSART` and `putrsUSART` should be used on parts with a single USART peripheral. The other functions should be used on parts with multiple USART peripherals.

File Name:
uputs.c
u1puts.c
u2puts.c
uputrs.c
u1putrs.c
u2putrs.c

Code Example:
```
putrsUSART( “Hello World!” );
```

ReadUSART
Read1USART
Read2USART
getcUSART
getc1USART
getc2USART

Function: Read a byte (one character) out of the USART receive buffer, including the 9th bit if enabled.

Include: usart.h

Prototype:
```
char getcUSART(  void );
char getc1USART( void );
char getc2USART( void );
char ReadUSART(  void );
char Read1USART( void );
char Read2USART( void );
```

ReadUSART
Read1USART
Read2USART
getcUSART
getc1USART
getc2USART (Continued)

Remarks: This function reads a byte out of the USART receive buffer. The status bits and the 9th data bits are saved in a union with the following declaration:

```
union USART
{
  unsigned char val;
  struct
  {
    unsigned RX_NINE:1;
    unsigned TX_NINE:1;
    unsigned FRAME_ERROR:1;
    unsigned OVERRUN_ERROR:1;
    unsigned fill:4;
  };
};
```

The 9th bit is read only if 9-bit mode is enabled. The status bits are always read.
On a part with a single USART peripheral, the `getcUSART` and `ReadUSART` functions should be used and the status information is read into a variable named `USART_Status` which is of the type `USART` described above.
On a part with multiple USART peripherals, the `getcxUSART` and `ReadxUSART` functions should be used and the status information is read into a variable named `USARTx_Status` which is of the type `USART` described above.

Return Value: This function returns the next character in the USART receive buffer.

File Name:
```
uread.c
u1read.c
u2read.c
```

Code Example:
```
int result;
result = ReadUSART();
result |= (unsigned int)
          USART_Status.RX_NINE << 8;
```

WriteUSART
Write1USART
Write2USART
putcUSART
putc1USART
putc2USART

Function: Write a byte (one character) to the USART transmit buffer, including the 9th bit if enabled.

Include: usart.h

Prototype:
```
void putcUSART(  char data );
void putc1USART( char data );
void putc2USART( char data );
void WriteUSART(  char data );
void Write1USART( char data );
void Write2USART( char data );
```

Arguments:
data
The value to be written to the USART.

Remarks: This function writes a byte to the USART transmit buffer. If 9-bit mode is enabled, the 9th bit is written from the field TX_NINE, found in a variable of type USART:

```
union USART
{
  unsigned char val;
  struct
  {
    unsigned RX_NINE:1;
    unsigned TX_NINE:1;
    unsigned FRAME_ERROR:1;
    unsigned OVERRUN_ERROR:1;
    unsigned fill:4;
  };
};
```

On a part with a single USART peripheral, the putcUSART and WriteUSART functions should be used and the status register is named USART_Status which is of the type USART described above.
On a part with multiple USART peripherals, the putc*x*USART and Write*x*USART functions should be used and the status register is named named USART*x*_Status which is of the type USART described above.

File Name:
uwrite.c
u1write.c
u2write.c

Code Example:
```
unsigned int outval;
USART1_Status.TX_NINE = (outval & 0x0100)
                        >> 8;
WriteUSART( (char) outval );
```

2.10.2 Example of Use

```
#include <p18C452.h>
#include <usart.h>

void main(void)
{
  // configure USART
  OpenUSART( USART_TX_INT_OFF  &
             USART_RX_INT_OFF  &
             USART_ASYNCH_MODE &
             USART_EIGHT_BIT   &
             USART_CONT_RX     &
             USART_BRGH_HIGH,
             25 );

  while(1)
  {
    while( ! PORTAbits.RA0 );  //wait for RA0 high

    WriteUSART( PORTD );       //write value of PORTD

    if(PORTD == 0x80)          // check for termination
      break;                   //   value
  }

  CloseUSART();
}
```

NOTES:

Chapter 3. Software Peripheral Library

3.1 INTRODUCTION

This chapter documents software peripheral library functions.The source code for all of these functions is included with MPLAB C18 in the `src\pmc` subdirectory of the compiler installation.

See the *MPASM™ User's Guide with MPLINK™ and MPLIB™* (DS33014) for more information about building libraries.

The following peripherals are supported by MPLAB C18 library routines

- External LCD Functions (3.2 “External LCD Functions”)
- External CAN2510 Functions (3.3 “External CAN2510 Functions”)
- Software I^{2}C Functions (3.4 “Software I²C Functions”)
- Software SPI Functions (3.5 “Software SPI Functions”)
- Software UART Functions (3.6 “Software UART Functions”)

3.2 EXTERNAL LCD FUNCTIONS

These functions are designed to allow the control of a Hitachi HD44780 LCD controller using I/O pins from a PIC18 microcontroller. The following functions are provided:

Function	Description
`BusyXLCD`	Is the LCD controller busy?
`OpenXLCD`	Configure the I/O lines used for controlling the LCD and initialize the LCD.
`putcXLCD`	Write a byte to the LCD controller.
`putsXLCD`	Write a string from data memory to the LCD.
`putrsXLCD`	Write a string from program memory to the LCD.
`ReadAddrXLCD`	Read the address byte from the LCD controller.
`ReadDataXLCD`	Read a byte from the LCD controller.
`SetCGRamAddr`	Set the character generator address.
`SetDDRamAddr`	Set the display data address.
`WriteCmdXLCD`	Write a command to the LCD controller.
`WriteDataXLCD`	Write a byte to the LCD controller.

The precompiled versions of these functions use default pin assignments that can be changed by redefining the following macro assignments in the file `xlcd.h`, found in the `h` subdirectory of the compiler installation:

LCD Controller Line	Macros	Default Value	Use
E Pin	E_PIN	PORTBbits.RB4	Pin used for the E line.
	TRIS_E	DDRBbits.RB4	Bit that controls the direction of the pin associated with the E line.
RS Pin	RS_PIN	PORTBbits.RB5	Pin used for the RS line.
	TRIS_RS	DDRBbits.RB5	Bit that controls the direction of the pin associated with the RS line.
RW Pin	RW_PIN	PORTBbits.RB6	Pin used for the RW line.
	TRIS_RW	DDRBbits.RB6	Bit that controls the direction of the pin associated with the RW line.
Data Lines	DATA_PORT	PORTB	Pins used for DATA lines. These routines assume all pins are on a single port.
	TRIS_DATA_PORT	DDRB	Data direction register associated with the DATA lines.

The libraries that are provided can operate in either a 4-bit mode or 8-bit mode. When operating in 8-bit mode, all the lines of a single port are used. When operating in 4-bit mode, either the upper 4 bits or lower 4 bits of a single port are used. The table below lists the macros used for selecting between 4- or 8- bit mode and for selecting which bits of a port are used when operating in 4-bit mode

Macro	Default Value	Use
BIT8	not defined	If this value is defined when the library functions are built, they will operate in 8-bit Transfer mode. Otherwise, they will operate in 4-bit Transfer mode.
UPPER	not defined	When BIT8 is not defined, this value determines which nibble of the DATA_PORT is used for data transfer. If UPPER is defined, the upper 4 bits (4:7) of DATA_PORT are used. If UPPER is not defined, the lower 4 bits (0:3) of DATA_PORT are used.

After these definitions have been made, the user must recompile the XLCD routines and then include the updated files in the project. This can be accomplished by adding the XLCD source files into the project or by recompiling the library files using the provided batch files.

The XLCD libraries also require that the following functions be defined by the user to provide the appropriate delays:

Function	Behavior
DelayFor18TCY	Delay for 18 cycles.
DelayPORXLCD	Delay for 15 ms.
DelayXLCD	Delay for 5 ms.

3.2.1 Function Descriptions

BusyXLCD

Function: Is the LCD controller busy?

Include: `xlcd.h`

Prototype:
```
unsigned char BusyXLCD( void );
```

Remarks: This function returns the status of the busy flag of the Hitachi HD44780 LCD controller.

Return Value: 1 if the controller is busy
0 otherwise.

File Name: `busyxlcd.c`

Code Example:
```
while( BusyXLCD() );
```

OpenXLCD

Function: Configure the PIC® I/O pins and initialize the LCD controller.

Include: `xlcd.h`

Prototype:
```
void OpenXLCD( unsigned char lcdtype );
```

Arguments: ***lcdtype***
A bitmask that is created by performing a bitwise AND operation ('&') with a value from each of the categories listed below. These values are defined in the file `xlcd.h`.

Data Interface:

`FOUR_BIT`	4-bit Data Interface mode
`EIGHT_BIT`	8-bit Data Interface mode

LCD Configuration:

`LINE_5X7`	5x7 characters, single line display
`LINE_5X10`	5x10 characters display
`LINES_5X7`	5x7 characters, multiple line display

Remarks: This function configures the PIC18 I/O pins used to control the Hitachi HD44780 LCD controller. It also initializes this controller.

File Name: `openxlcd.c`

Code Example:
```
OpenXLCD( EIGHT_BIT & LINES_5X7 );
```

putcXLCD

See **WriteDataXLCD.**

putsXLCD
putrsXLCD

Function: Write a string to the Hitachi HD44780 LCD controller.

Include: `xlcd.h`

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

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

putsXLCD
putrsXLCD (Continued)

Remarks: This function writes a string of characters located in *buffer* to the Hitachi HD44780 LCD controller. It stops transmission when a null character is encountered. The null character is not transmitted.
Strings located in data memory should be used with the “puts” versions of these functions.
Strings located in program memory, including string literals, should be used with the “putrs” versions of these functions.

File Name: putsxlcd.c
putrxlcd.c

Code Example:
```
char mybuff [20];
putrsXLCD( "Hello World" );
putsXLCD( mybuff );
```

ReadAddrXLCD

Function: Read the address byte from the Hitachi HD44780 LCD controller.

Include: xlcd.h

Prototype:
```
unsigned char ReadAddrXLCD( void );
```

Remarks: This function reads the address byte from the Hitachi HD44780 LCD controller. The LCD controller should not be busy when this operation is performed – this can be verified using 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.

Return Value: This function returns an 8-bit quantity. The address is contained in the lower order 7 bits and the BUSY status flag in the Most Significant bit.

File Name: readaddr.c

Code Example:
```
char addr;
while ( BusyXLCD() );
addr = ReadAddrXLCD();
```

ReadDataXLCD

Function: Read a data byte from the Hitachi HD44780 LCD controller.

Include: xlcd.h

Prototype:
```
char ReadDataXLCD( void );
```

Remarks: This function reads a data byte from the Hitachi HD44780 LCD controller. The LCD controller should not be busy when this operation is performed – this can be verified using 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.

ReadDataXLCD (Continued)

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

File Name: readdata.c

Code Example:
```
char data;
while ( BusyXLCD() );
data = ReadAddrXLCD();
```

SetCGRamAddr

Function: Set the character generator address.

Include: xlcd.h

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

Arguments:
addr
Character generator address.

Remarks: This function sets the character generator address of the Hitachi HD44780 LCD controller. The LCD controller should not be busy when this operation is performed – this can be verified using the BusyXLCD function.

File Name: setcgram.c

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

SetDDRamAddr

Function: Set the display data address.

Include: xlcd.h

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

Arguments:
addr
Display data address.

Remarks: This function sets the display data address of the Hitachi HD44780 LCD controller. The LCD controller should not be busy when this operation is performed – this can be verified using the BusyXLCD function.

File Name: setddram.c

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

WriteCmdXLCD

Function: Write a command to the Hitachi HD44780 LCD controller.

Include: xlcd.h

Prototype: `void WriteCmdXLCD( unsigned char cmd );`

Arguments:
cmd
Specifies the command to be performed. The command may be one of the following values defined in xlcd.h:

WriteCmdXLCD (Continued)

DOFF	Turn display off
CURSOR_OFF	Enable display with no cursor
BLINK_ON	Enable display with blinking cursor
BLINK_OFF	Enable display with unblinking cursor
SHIFT_CUR_LEFT	Cursor shifts to the left
SHIFT_CUR_RIGHT	Cursor shifts to the right
SHIFT_DISP_LEFT	Display shifts to the left
SHIFT_DISP_RIGHT	Display shifts to the right

Alternatively, the command may be a bitmask that is created by performing a bitwise AND operation ('&') with a value from each of the categories listed below. These values are defined in the file xlcd.h.

Data Transfer mode:

FOUR_BIT	4-bit Data Interface mode
EIGHT_BIT	8-bit Data Interface mode

Display Type:

LINE_5X7	5x7 characters, single line
LINE_5X10	5x10 characters display
LINES_5X7	5x7 characters, multiple lines

Remarks: This function writes the command byte to the Hitachi HD44780 LCD controller. The LCD controller should not be busy when this operation is performed – this can be verified using the BusyXLCD function.

File Name: wcmdxlcd.c

Code Example:

```
while( BusyXLCD() );
WriteCmdXLCD( EIGHT_BIT & LINES_5X7 );
WriteCmdXLCD( BLINK_ON );
WriteCmdXLCD( SHIFT_DISP_LEFT );
```

putcXLCD
WriteDataXLCD

Function: Writes a byte to the Hitachi HD44780 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 HD44780 LCD controller.

Remarks: This function writes a data byte to the Hitachi HD44780 LCD controller. The LCD controller should not be busy when this operation is performed – this can be verified using 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.

File Name: writdata.c

3.2.2 Example of Use

```
#include <p18C452.h>
#include <xlcd.h>
#include <delays.h>
#include <usart.h>

void DelayFor18TCY( void )
{
  Nop();
  Nop();
  Nop();
  Nop();
  Nop();
  Nop();
  Nop();
  Nop();
  Nop();
  Nop();
  Nop();
  Nop();
}

void DelayPORXLCD( void )
{
  Delay1KTCYx(60); //Delay of 15ms
  return;
}

void DelayXLCD( void )
{
  Delay1KTCYx(20); //Delay of 5ms
  return;
}

void main( void )
{
  char data;

  // configure external LCD
  OpenXLCD( EIGHT_BIT & LINES_5X7 );

  // configure USART
  OpenUSART( USART_TX_INT_OFF  & USART_RX_INT_OFF &
             USART_ASYNCH_MODE & USART_EIGHT_BIT  &
             USART_CONT_RX,
             25);

  while(1)
  {
    while(!DataRdyUSART());  //wait for data
    data = ReadUSART();      //read data
    WriteDataXLCD(data);     //write to LCD
    if(data=='Q')
      break;
  }

  CloseUSART();
}
```

3.3 EXTERNAL CAN2510 FUNCTIONS

This section documents the MCP2510 external peripheral library functions. The following functions are provided:

Function	Description
CAN2510BitModify	Modifies the specified bits in a register to the new values.
CAN2510ByteRead	Reads the MCP2510 register specified by the address.
CAN2510ByteWrite	Writes a value to the MCP2510 register specified by the address.
CAN2510DataRead	Reads a message from the specified receive buffer.
CAN2510DataReady	Determines if data is waiting in the specified receive buffer.
CAN2510Disable	Drives the selected PIC18CXXX I/O pin high to disable the Chip Select of the MCP2510.*
CAN2510Enable	Drives the selected PIC18CXXX I/O pin low to Chip Select the MCP2510.*
CAN2510ErrorState	Reads the current Error State of the CAN bus.
CAN2510Init	Initialize the PIC18CXXX SPI port for communications to the MCP2510 and then configures the MCP2510 registers to interface with the CAN bus.
CAN2510InterruptEnable	Modifies the CAN2510 interrupt enable bits (CANINTE register) to the new values.
CAN2510InterruptStatus	Indicates the source of the CAN2510 interrupt.
CAN2510LoadBufferStd	Loads a Standard data frame into the specified transfer buffer.
CAN2510LoadBufferXtd	Loads an Extended data frame into the specified transfer buffer.
CAN2510LoadRTRStd	Loads a Standard remote frame into the specified transfer buffer.
CAN2510LoadRTRXtd	Loads an Extended remote frame into the specified transfer buffer.
CAN2510ReadMode	Reads the MCP2510 current mode of operation.
CAN2510ReadStatus	Reads the status of the MCP2510 Transmit and Receive Buffers.
CAN2510Reset	Resets the MCP2510.
CAN2510SendBuffer	Requests message transmission for the specified transmit buffer(s).
CAN2510SequentialRead	Reads the number of specified bytes in the MCP2510, starting at the specified address. These values will be stored in DataArray.
CAN2510SequentialWrite	Writes the number of specified bytes in the MCP2510, starting at the specified address. These values will be written from DataArray.
CAN2510SetBufferPriority	Loads the specified priority for the specified transmit buffer.
CAN2510SetMode	Configures the MCP2510 mode of operation.
CAN2510SetMsgFilterStd	Configures ALL of the filter and mask values of the specific receive buffer for a standard message.
CAN2510SetMsgFilterXtd	Configures ALL of the filter and mask values of the specific receive buffer for a extended message.
CAN2510SetSingleFilterStd	Configures the specified Receive filter with a filter value for a Standard (Std) message.

Function	Description
CAN2510SetSingleFilterXtd	Configures the specified Receive filter with a filter value for a Extended (Xtd) message.
CAN2510SetSingleMaskStd	Configures the specified Receive buffer mask with a mask value for a Standard (Std) format message.
CAN2510SetSingleMaskXtd	Configures the specified Receive buffer mask with a mask value for an Extended (Xtd) message.
CAN2510WriteStd	Writes a Standard format message out to the CAN bus using the first available transmit buffer.
CAN2510WriteXtd	Writes an Extended format message out to the CAN bus using the first available transmit buffer.

* The functions CAN2510Enable and CAN2510Disable will need to be recompiled if:
  - the PICmicro MCU assignment of the $\overline{CS}$ pin is modified from RC2
  - the device header file needs to be changed

3.3.1 Function Descriptions

CAN2510BitModify

Function: Modifies the specified bits in a register to the new values.

Required CAN Mode(s): All

Include: can2510.h

Prototype:
```
void CAN2510BitModify(
    unsigned char addr
    unsigned char mask
    unsigned char data );
```

Arguments:

addr
The value of ***addr*** specifies the address of the MCP2510 register to modify.

mask
The value of ***mask*** specifies the bits that will be modified.

data
The value of ***data*** specifies the new state of the bits.

Remarks: This function modifies the contents of the register specified by address, the mask specifies which bits are to be modified and the data specifies the new value to load into those bits. Only specific registers can be modified with the Bit Modify command.

File Name: canbmod.c

CAN2510ByteRead

Function: Reads the MCP2510 register specified by the address.

Required CAN Mode(s): All

Include: can2510.h

Prototype:
```
unsigned char CAN2510ByteRead(
    unsigned char address );
```

Arguments:

address
The address of the MCP2510 that is to be read.

Remarks: This function reads a single byte from the MCP2510 at the specified address.

Return Value: The contents of the specified address.

File Name: readbyte.c

CAN2510ByteWrite

Function: Writes a value to the MCP2510 register specified by the address.

Required CAN Mode(s): All

Include: can2510.h

Prototype:
```
void CAN2510ByteWrite(
    unsigned char address,
    unsigned char value );
```

CAN2510ByteWrite (Continued)

Arguments:

address
The address of the MCP2510 that is to be written.

value
The value that is to be written.

Remarks: This function writes a single byte from the MCP2510 at the specified address.

File Name: `wrtbyte.c`

CAN2510DataRead

Function: Reads a message from the specified receive buffer.

Required CAN Mode(s): All (except Configuration mode)

Include: `can2510.h`

Prototype:

```
unsigned char CAN2510DataRead(
    unsigned char bufferNum,
    unsigned long *msgId,
    unsigned char *numBytes,
    unsigned char *data );
```

Arguments:

bufferNum
Receive buffer from which to read the message. One of the following values:

`CAN2510_RXB0`	Read receive buffer 0
`CAN2510_RXB1`	Read receive buffer 1

msgId
Points to a location that will be modified by the function to contain the CAN standard message identifier.

numBytes
Points to a location that will be modified by the function to contain the number of bytes in this message.

data
Points to an array that will be modified by the function to contain the message data. This array should be at least 8 bytes long, since that is the maximum message data length.

Remarks: This function determines if the message is a standard or extended message, decodes the ID and message length, and fills in the user-supplied locations with the appropriate information. The `CAN2510DataReady` function should be used to determine if a specified buffer has data to read.

Return Value: Function returns one of the following values:

`CAN2510_XTDMSG`	Extended format message
`CAN2510_STDMSG`	Standard format message
`CAN2510_XTDRTR`	Remote transmit request (XTD message)
`CAN2510_STDRTR`	Remote transmit request (STD message)

File Name: `canread.c`

CAN2510DataReady

Function: Determines if data is waiting in the specified receive buffer.

Required CAN Mode(s): All (except Configuration mode)

Include: `can2510.h`

Prototype:
```
unsigned char CAN2510DataReady(
    unsigned char bufferNum );
```

Arguments: ***bufferNum***
Receive buffer to check for waiting message. One of the following values:

`CAN2510_RXB0`	Check Receive Buffer 0
`CAN2510_RXB1`	Check Receive Buffer 1
`CAN2510_RXBX`	Check Receive Buffer 0 and Receive Buffer 1

Remarks: This function tests the appropriate `RXnIF` bit in the CANINTF register.

Return Value: Returns zero if no message detected or a non-zero value if a message was detected.
1 = buffer0
2 = buffer1
3 = both

File Name: `canready.c`

CAN2510Disable

Function: Drives the selected PIC18CXXX I/O pin high to disable the Chip Select of the MCP2510.

Required CAN Mode(s): All

Include: `canenabl.h`

Note: This include file will need to be modified if the chip select signal is not associated with the RC2 pin of the PICmicro MCU.

Prototype:
```
void CAN2510Disable( void );
```

Arguments: None

Remarks: This function requires that the user modifies the file to specify the PIC18CXXX I/O pin (and Port) that will be used to connect to the MCP2510 $\overline{CS}$ pin. The default pin is RC2.

Note: The source file that contains this function (and the `CAN2510Enable` function) must have the definitions modified to correctly specify the Port (A, B, C, ...) and Pin number (1, 2, 3, ...) that is used to control the MCP2510 $\overline{CS}$ pin. After the modification, the processor-specific library must be rebuilt. See 1.5.3 “Rebuilding” for information on rebuilding.

File Name: `canenabl.c`

CAN2510Enable

Function: Drives the selected PIC18CXXX I/O pin low to Chip Select the MCP2510.

Required CAN Mode(s): All

Include: `canenabl.h`

Note: This include file will need to be modified if the chip select signal is not associated with the RC2 pin of the PICmicro MCU.

Prototype:
```
void CAN2510Enable( void );
```

Remarks: This function requires that the user modifies the file to specify the PIC18CXXX I/O pin (and Port) that will be used to connect to the MCP2510 $\overline{CS}$ pin. The default pin is RC2.

Note: The source file that contains this function (and the `CAN2510Disable` function) must have the definitions modified to correctly specify the Port (A, B, C, ...) and Pin number (1, 2, 3, ...) that is used to control the MCP2510 $\overline{CS}$ pin. After the modification, the processor-specific library must be rebuilt. See 1.5.3 “Rebuilding” for information on rebuilding.

File Name: `canenabl.c`

CAN2510ErrorState

Function: Reads the current Error State of the CAN bus.

Required CAN Mode(s): Normal mode, Loopback mode, Listen Only mode (Error counters are reset in Configuration mode)

Include: `can2510.h`

Prototype:
```
unsigned char CAN2510ErrorState( void );
```

Remarks: This function returns the Error State of the CAN bus. The Error State is dependent on the values in the TEC and REC registers.

Return Value: Function returns one of the following values:

`CAN2510_BUS_OFF`	TEC > 255
`CAN2510_ERROR_PASSIVE_TX`	TEC > 127
`CAN2510_ERROR_PASSIVE_RX`	REC > 127
`CAN2510_ERROR_ACTIVE_WITH_TXWARN`	TEC > 95
`CAN2510_ERROR_ACTIVE_WITH_RXWARN`	REC > 95
`CAN2510_ERROR_ACTIVE`	TEC ≤ 95 and REC ≤ 95

File Name: `canerrst.c`

CAN2510Init

Function: Initialize the PIC18CXXX SPI port for communications to the MCP2510 and then configures the MCP2510 registers to interface with the CAN bus.

Required CAN Mode(s): Configuration mode

Include: can2510.h

Prototype:

```
unsigned char CAN2510Init(
    unsigned short long BufferConfig,
    unsigned short long BitTimeConfig,
    unsigned char interruptEnables,
    unsigned char SPI_syncMode,
    unsigned char SPI_busMode,
    unsigned char SPI_smpPhase );
```

Arguments: The values of the following parameters are defined in the include file can2510.h.

BufferConfig

The value of BufferConfig is constructed through the bitwise AND (&) operation of the following options. Only one option per group function may be selected. The option in the **bold font** is the default value.

Reset MCP2510 Device

Specifies if the MCP2510 RESET command is to be sent. This does not correspond to a bit in the MCP2510 registers.

CAN2510_NORESET	**Don't reset the MCP2510**
CAN2510_RESET	Reset the MCP2510

Buffer 0 Filtering

Controlled by the RXB0M1:RXB0M0 bits (RXB0CTRL register)

CAN2510_RXB0_USEFILT	**Receive all messages, Use filters**
CAN2510_RXB0_STDMSG	Receive only Standard messages
CAN2510_RXB0_XTDMSG	Receive only Extended messages
CAN2510_RXB0_NOFILT	Receive all messages, NO filters

Buffer 1 Filtering

Controlled by the RXB1M1:RXB1M0 bits (RXB1CTRL register)

CAN2510_RXB1_USEFILT	**Receive all messages, Use filters**
CAN2510_RXB1_STDMSG	Receive only Standard messages
CAN2510_RXB1_XTDMSG	Receive only Extended messages
CAN2510_RXB1_NOFILT	Receive all messages, NO filters

Receive Buffer 0 to Receive Buffer 1 Rollover

Controlled by the BUKT bit (RXB0CTRL register)

CAN2510_RXB0_ROLL	**If receive buffer 0 is full, message goes to receive buffer**
CAN2510_RXB0_NOROLL	Rollover Disabled

CAN2510Init (Continued)

RX1BF Pin Setting

Controlled by the B1BFS:B1BFE:B1BFM bits (BFPCTRL register)

CAN2510_RX1BF_OFF	**RX1BF pin is Hi-impedance**
CAN2510_RX1BF_INT	RX1BF pin is an output which indicates Receive Buffer 1 was loaded. Can be used as an interrupt signal.
CAN2510_RX1BF_GPOUTH	RX1BF pin is a general purpose digital output, Output High
CAN2510_RX1BF_GPOUTL	RX1BF pin is a general purpose digital output, Output Low

RX0BF Pin Setting

Controlled by the B0BFS:B0BFE:B0BFM bits (BFPCTRL register)

CAN2510_RX0BF_OFF	**RX0BF pin is Hi-impedance**
CAN2510_RX0BF_INT	RX0BF pin is an output which indicates Receive Buffer 0 was loaded. Can be used as an interrupt signal.
CAN2510_RX0BF_GPOUTH	RX0BF pin is a general purpose digital output, Output High
CAN2510_RX0BF_GPOUTL	RX0BF pin is a general purpose digital output, Output Low

TX2 Pin Setting

Controlled by the B2RTSM bit (TXRTSCTRL register)

CAN2510_TX2_GPIN	**TX2RTS pin is a digital input**
CAN2510_TX2_RTS	TX2RTS pin is an input used to initiate a Request To Send frame from TXBUF2

TX1 Pin Setting

Controlled by the B1RTSM bit (TXRTSCTRL register)

CAN2510_TX1_GPIN	**TX1RTS pin is a digital input**
CAN2510_TX1_RTS	TX1RTS pin is an input used to initiate a Request To Send frame from TXBUF1

TX0 Pin Setting

Controlled by the B0RTSM bit (TXRTSCTRL register)

CAN2510_TX0_GPIN	**TX0RTS pin is a digital input**
CAN2510_TX0_RTS	TX0RTS pin is an input used to initiate a Request To Send frame from TXBUF0

Request Mode of Operation

Controlled by the REQOP2:REQOP0 bits (CANCTRL register)

CAN2510_REQ_CONFIG	**Configuration Mode**
CAN2510_REQ_NORMAL	Normal Operation Mode
CAN2510_REQ_SLEEP	SLEEP Mode
CAN2510_REQ_LOOPBACK	Loop Back Mode
CAN2510_REQ_LISTEN	Listen Only Mode

CAN2510Init (Continued)

CLKOUT Pin Setting

Controlled by the CLKEN:CLKPRE1:CLKPRE0 bits (CANCTRL register)

CAN2510_CLKOUT_8	**CLKOUT = FOSC / 8**
CAN2510_CLKOUT_4	CLKOUT = FOSC / 4
CAN2510_CLKOUT_2	CLKOUT = FOSC / 2
CAN2510_CLKOUT_1	CLKOUT = FOSC
CAN2510_CLKOUT_OFF	CLKOUT is Disabled

BitTimeConfig

The value of BitTimeConfig is constructed through the bitwise AND (&) operation of the following options. Only one option per group function may be selected. The option in the **bold font** is the default value.

Baud Rate Prescaler (BRP)

Controlled by the BRP5:BRP0 bits (CNF1 register)

CAN2510_BRG_1X	**TQ = 1 x (2TOSC)**
:	**:**
CAN2510_BRG_64X	TQ = 64 x (2TOSC)

Synchronization Jump Width

Controlled by the SJW1:SJW0 bits (CNF1 register)

CAN2510_SJW_1TQ	**SJW length = 1 TQ**
CAN2510_SJW_2TQ	SJW length = 2 TQ
CAN2510_SJW_3TQ	SJW length = 3 TQ
CAN2510_SJW_4TQ	SJW length = 4 TQ

Phase 2 Segment Width

Controlled by the PH2SEG2:PH2SEG0 bits (CNF3 register)

CAN2510_PH2SEG_2TQ	**Length = 2 TQ**
CAN2510_PH2SEG_3TQ	Length = 3 TQ
CAN2510_PH2SEG_4TQ	Length = 4 TQ
CAN2510_PH2SEG_5TQ	Length = 5 TQ
CAN2510_PH2SEG_6TQ	Length = 6 TQ
CAN2510_PH2SEG_7TQ	Length = 7 TQ
CAN2510_PH2SEG_8TQ	Length = 8 TQ

Phase 1 Segment Width

Controlled by the PH1SEG2:PH1SEG0 bits (CNF2 register)

CAN2510_PH1SEG_1TQ	**Length = 1 TQ**
CAN2510_PH1SEG_2TQ	Length = 2 TQ
CAN2510_PH1SEG_3TQ	Length = 3 TQ
CAN2510_PH1SEG_4TQ	Length = 4 TQ
CAN2510_PH1SEG_5TQ	Length = 5 TQ
CAN2510_PH1SEG_6TQ	Length = 6 TQ
CAN2510_PH1SEG_7TQ	Length = 7 TQ
CAN2510_PH1SEG_8TQ	Length = 8 TQ

CAN2510Init (Continued)

Propagation Segment Width

Controlled by the PRSEG2:PRSEG0 bits (CNF2 register)

CAN2510_PROPSEG_1TQ	**Length = 1 TQ**
CAN2510_PROPSEG_2TQ	Length = 2 TQ
CAN2510_PROPSEG_3TQ	Length = 3 TQ
CAN2510_PROPSEG_4TQ	Length = 4 TQ
CAN2510_PROPSEG_5TQ	Length = 5 TQ
CAN2510_PROPSEG_6TQ	Length = 6 TQ
CAN2510_PROPSEG_7TQ	Length = 7 TQ
CAN2510_PROPSEG_8TQ	Length = 8 TQ

Phase 2 Source

Controlled by the BTLMODE bit (CNF2 register). This determines if the Phase 2 length is determined by the PH2SEG2:PH2SEG0 bits or the greater length of PH1SEG2:PH1SEG0 bits and (2TQ).

CAN2510_PH2SOURCE_PH2	**Length = PH2SEG2:PH2SEG0**
CAN2510_PH2SOURCE_PH1	Length = greater of PH1SEG2:PH1SEG0 and 2TQ

Bit Sample Point Frequency

Controlled by the SAM bit (CNF2 register). This determines if the bit is sampled 1 or 3 times at the sample point.

CAN2510_SAMPLE_1x	**Bit is sampled once**
CAN2510_SAMPLE_3x	Bit is sampled three times

RX pin Noise Filter in SLEEP Mode

Controlled by the WAKFIL bit (CNF3 register). This determines if the RX pin will use a filter to reject noise when the device is in SLEEP mode.

CAN2510_RX_FILTER	**Filtering on RX pin when in SLEEP mode**
CAN2510_RX_NOFILTER	No filtering on RX pin when in SLEEP mode

interruptEnables

The value of interruptEnables can be a combination of the following values, combined using a bitwise AND (&) operation. The option in the **bold font** is the default value. Controlled by all bits in the CANINTE register.

CAN2510_NONE_EN	**No interrupts enabled**
CAN2510_MSGERR_EN	Interrupt on error during message reception or transmission
CAN2510_WAKEUP_EN	Interrupt on CAN bus activity
CAN2510_ERROR_EN	Interrupt on EFLG error condition change
CAN2510_TXB2_EN	Interrupt on transmission buffer 2 becoming empty
CAN2510_TXB1_EN	Interrupt on transmission buffer 1 becoming empty
CAN2510_TXB0_EN	Interrupt on transmission buffer 0 becoming empty
CAN2510_RXB1_EN	Interrupt when message received in receive buffer 1
CAN2510_RXB0_EN	Interrupt when message received in receive buffer 0

CAN2510Init (Continued)

SPI_syncMode
Specifies the PIC18CXXX SPI synchronization frequency:

CAN2510_SPI_FOSC4	Communicates at FOSC/4
CAN2510_SPI_FOSC16	Communicates at FOSC/16
CAN2510_SPI_FOSC64	Communicates at FOSC/64
CAN2510_SPI_FOSCTMR2	Communicates at TMR2/2

SPI_busMode
Specifies the PIC18CXXX SPI bus mode:

CAN2510_SPI_MODE00	Communicate using SPI mode 00
CAN2510_SPI_MODE01	Communicate using SPI mode 01

SPI_smpPhase
Specifies the PIC18CXXX SPI sample point:

CAN2510_SPI_SMPMID	Samples in middle of SPI bit
CAN2510_SPI_SMPEND	Samples at end of SPI bit

Remarks: This function initializes the PIC18CXXX SPI module, resets the MCP2510 device (if requested) and then configures the MCP2510 registers.

Note: When this function is completed, the MCP2510 is left in the Configuration mode.

Return Value: Indicates if the MCP2510 could be initialized.
0 if initialization completed
-1 if initialization did not complete

File Name: caninit.c

CAN2510InterruptEnable

Function: Modifies the CAN2510 interrupt enable bits (CANINTE register) to the new values.

Required CAN Mode(s): All

Include: can2510.h,
spi_can.h

Prototype:
```
void CAN2510InterruptEnable(
    unsigned char interruptEnables );
```

Arguments: ***interruptEnables***
The value of ***interruptEnables*** can be a combination of the following values, combined using a bitwise AND (&) operation. The option in the **bold font** is the default value. Controlled by all bits in the CANINTE register.

CAN2510_NONE_EN	**No interrupts enabled (00000000)**
CAN2510_MSGERR_EN	Interrupt on error during message reception or transmission (10000000)
CAN2510_WAKEUP_EN	Interrupt on CAN bus activity (01000000)
CAN2510_ERROR_EN	Interrupt on EFLG error condition change (00100000)

CAN2510InterruptEnable (Continued)

CAN2510_TXB2_EN	Interrupt on transmission buffer 2 becoming empty (00010000)
CAN2510_TXB1_EN	Interrupt on transmission buffer 1 becoming empty (00001000)
CAN2510_TXB0_EN	Interrupt on transmission buffer 0 becoming empty (00000100)
CAN2510_RXB1_EN	Interrupt when message received in receive buffer 1 (00000010)
CAN2510_RXB0_EN	Interrupt when message received in receive buffer 0 (00000001)

Remarks: This function updates the CANINTE register with the value that is determined by ANDing the desired interrupt sources.

File Name: caninte.c

CAN2510InterruptStatus

Function: Indicates the source of the CAN2510 interrupt.

Required CAN Mode(s): All

Include: can2510.h, spi_can.h

Prototype:
```
unsigned char CAN2510InterruptStatus(
   void );
```

Remarks: This function reads the CANSTAT register and specifies a code depending on the state of the ICODE2:ICODE0 bits.

Return Value: Function returns one of the following values:

CAN2510_NO_INTS	No interrupts occurred
CAN2510_WAKEUP_INT	Interrupt on CAN bus activity
CAN2510_ERROR_INT	Interrupt on EFLG error condition change
CAN2510_TXB2_INT	Interrupt on transmission buffer 2 becoming empty
CAN2510_TXB1_INT	Interrupt on transmission buffer 1 becoming empty
CAN2510_TXB0_INT	Interrupt on transmission buffer 0 becoming empty
CAN2510_RXB1_INT	Interrupt when message received in receive buffer 1
CAN2510_RXB0_INT	Interrupt when message received in receive buffer 0

File Name: canints.c

CAN2510LoadBufferStd

Function: Loads a Standard data frame into the specified transfer buffer.

Required CAN Mode(s): All

Include: can2510.h

CAN2510LoadBufferStd (Continued)

Prototype:

```
void CAN2510LoadBufferStd(
    unsigned char bufferNum,
    unsigned int  msgId,
    unsigned char numBytes,
    unsigned char *data );
```

Arguments:

bufferNum
Specifies the buffer to load the message into. One of the following values:

CAN2510_TXB0	Transmit buffer 0
CAN2510_TXB1	Transmit buffer 1
CAN2510_TXB2	Transmit buffer 2

msgId
CAN message identifier, up to 11 bits for a standard message.

numBytes
Number of bytes of data to transmit, from 0 to 8. If value is greater than 8, only the first 8 bytes of data will be stored.

data
Array of data values to be loaded. The array must be at least as large as the value specified in ***numBytes***.

Remarks: This function loads the message information, but does not transmit the message. Use the CAN2510WriteBuffer() function to write the message onto the CAN bus.
This function does not set the priority of the buffer. Use the CAN2510SetBufferPriority() function to set buffer priority.

File Name: canloads.c

CAN2510LoadBufferXtd

Function: Loads an Extended data frame into the specified transfer buffer.

Required CAN Mode(s): All

Include: can2510.h

Prototype:

```
void CAN2510LoadBufferXtd(
    unsigned char bufferNum,
    unsigned int  msgId,
    unsigned char numBytes,
    unsigned char *data );
```

Arguments:

bufferNum
Specifies the buffer to load the message into. One of the following values:

CAN2510_TXB0	Transmit buffer 0
CAN2510_TXB1	Transmit buffer 1
CAN2510_TXB2	Transmit buffer 2

msgId
CAN message identifier, up to 29 bits for a extended message.

numBytes
Number of bytes of data to transmit, from 0 to 8. If value is greater than 8, only the first 8 bytes of data will be stored.

data
Array of data values to be loaded. The array must be at least as large as the value specified in ***numBytes***.

CAN2510LoadBufferXtd (Continued)

Remarks: This function loads the message information, but does not transmit the message. Use the `CAN2510WriteBuffer()` function to write the message onto the CAN bus.
This function does not set the priority of the buffer. Use the `CAN2510SetBufferPriority()` function to set buffer priority.

File Name: `canloadx.c`

CAN2510LoadRTRStd

Function: Loads a Standard remote frame into the specified transfer buffer.

Required CAN Mode(s): All

Include: `can2510.h`

Prototype:

```
void CAN2510LoadBufferStd(
    unsigned char bufferNum,
    unsigned int  msgId,
    unsigned char numBytes,
    unsigned char *data );
```

Arguments:

bufferNum
Specifies the buffer to load the message into. One of the following values:

`CAN2510_TXB0`	Transmit buffer 0
`CAN2510_TXB1`	Transmit buffer 1
`CAN2510_TXB2`	Transmit buffer 2

msgId
CAN message identifier, up to 11 bits for a standard message.

numBytes
Number of bytes of data to transmit, from 0 to 8. If value is greater than 8, only the first 8 bytes of data will be stored.

data
Array of data values to be loaded. The array must be at least as large as the value specified in ***numBytes***.

Remarks: This function loads the message information, but does not transmit the message. Use the `CAN2510WriteBuffer()` function to write the message onto the CAN bus.
This function does not set the priority of the buffer. Use the `CAN2510SetBufferPriority()` function to set buffer priority.

File Name: `canlrtrs.c`

CAN2510LoadRTRXtd

Function: Loads an Extended remote frame into the specified transfer buffer.

Required CAN Mode(s): All

Include: `can2510.h`

CAN2510LoadRTRXtd (Continued)

Prototype:
```
void CAN2510LoadBufferXtd(
    unsigned char bufferNum,
    unsigned long msgId,
    unsigned char numBytes,
    unsigned char *data );
```

Arguments:

bufferNum
Specifies the buffer to load the message into. One of the following values:

CAN2510_TXB0	Transmit buffer 0
CAN2510_TXB1	Transmit buffer 1
CAN2510_TXB2	Transmit buffer 2

msgId
CAN message identifier, up to 29 bits for a extended message.

numBytes
Number of bytes of data to transmit, from 0 to 8. If value is greater than 8, only the first 8 bytes of data will be stored.

data
Array of data values to be loaded. The array must be at least as large as the value specified in ***numBytes***.

Remarks: This function loads the message information, but does not transmit the message. Use the CAN2510WriteBuffer() function to write the message onto the CAN bus.
This function does not set the priority of the buffer. Use the CAN2510SetBufferPriority() function to set buffer priority.

File Name: canlrtrx.c

CAN2510ReadMode

Function: Reads the MCP2510 current mode of operation.

Required CAN Mode(s): All

Include: can2510.h

Prototype:
```
unsigned char CAN2510ReadMode( void );
```

Remarks: This function reads the current Operating mode. The mode may have a pending request for a new mode.

Return Value:

mode
The value of ***mode*** can be one of the following values (defined in can2510.h). Specified by the OPMODE2:OPMODE0 bits (CANSTAT register). One of the following values:

CAN2510_MODE_CONFIG	Configuration registers can be modified
CAN2510_MODE_NORMAL	Normal (send and receive messages)
CAN2510_MODE_SLEEP	Wait for interrupt
CAN2510_MODE_LISTEN	Listen only, don't send
CAN2510_MODE_LOOPBACK	Used for testing, messages stay internal

File Name: canmoder.c

CAN2510ReadStatus

Function: Reads the status of the MCP2510 Transmit and Receive Buffers.

Required CAN Mode(s): All

Include: `can2510.h`

Prototype:
```
unsigned char CAN2510ReadStatus( void );
```

Remarks: This function reads the current status of the transmit and receive buffers.

Return Value: ***status***

The value of ***status*** (an unsigned byte) has the following format:

bit 7	TXB2IF
bit 6	TXB2REQ
bit 5	TXB1IF
bit 4	TXB1REQ
bit 3	TXB0IF
bit 2	TXB0REQ
bit 1	RXB1IF
bit 0	RXB0IF

File Name: `canstats.c`

CAN2510Reset

Function: Resets the MCP2510.

Required CAN Mode(s): All

Include:
```
can2510.h
spi_can.h
spi.h
```

Prototype:
```
void CAN2510Reset( void );
```

Remarks: This function resets the MCP2510.

File Name: `canreset.c`

CAN2510SendBuffer

Function: Requests message transmission for the specified transmit buffer(s).

Required CAN Mode(s): Normal mode

Include: `can2510.h`

Prototype:
```
void CAN2510WriteBuffer
    ( unsigned char bufferNum );
```

CAN2510SendBuffer (Continued)

Arguments: ***bufferNum***
Specifies the buffer to request transmission of. One of the following values:

CAN2510_TXB0	Transmit buffer 0
CAN2510_TXB1	Transmit buffer 1
CAN2510_TXB2	Transmit buffer 2
CAN2510_TXB0_B1	Transmit buffer 0 and buffer 1
CAN2510_TXB0_B2	Transmit buffer 0 and buffer 2
CAN2510_TXB1_B2	Transmit buffer 1 and buffer 2
CAN2510_TXB0_B1_B2	Transmit buffer 0, buffer 1, and buffer 2

Remarks: This function requests transmission of a previously loaded message stored in the specified buffer(s). To load a message, use the CAN2510LoadBufferStd() or CAN2510LoadBufferXtd() routines.

File Name: cansend.c

CAN2510SequentialRead

Function: Reads the number of specified bytes in the MCP2510, starting at the specified address. These values will be stored in DataArray.

Required CAN Mode(s): All

Include: can2510.h

Prototype:
```
void CAN2510SequentialRead(
    unsigned char *DataArray
    unsigned char CAN2510addr
    unsigned char numbytes );
```

Arguments: ***DataArray***
The start address of the data array that stores the sequential read data.

CAN2510addr
The address of the MCP2510 where the sequential reads start from.

numbytes
The number of bytes to sequentially read.

Remarks: This function reads sequential bytes from the MCP2510 starting at the specified address. These values are loaded starting at the first address of the array that is specified.

File Name: readseq.c

CAN2510SequentialWrite

Function: Writes the number of specified bytes in the MCP2510, starting at the specified address. These values will be written from ***DataArray***.

Required CAN Mode(s): All

Include: can2510.h

CAN2510SequentialWrite (Continued)

Prototype:
```
void CAN2510SequentialWrite(
    unsigned char *DataArray
    unsigned char CAN2510addr
    unsigned char numbytes );
```

Arguments:

DataArray
The start address of the data array that contains the sequential write data.

CAN2510addr
The address of the MCP2510 where the sequential writes start from.

numbytes
The number of bytes to sequentially write.

Remarks: This function writes sequential bytes to the MCP2510 starting at the specified address. These values are contained starting at the first address of the array that is specified.

File Name: wrtseq.c

CAN2510SetBufferPriority

Function: Loads the specified priority for the specified transmit buffer.

Required CAN Mode(s): All

Include: can2510.h

Prototype:
```
void CAN2510SetBufferPriority(
    unsigned char bufferNum,
    unsigned char bufferPriority );
```

Arguments:

bufferNum
Specifies the buffer to configure the priority of. One of the following values:

CAN2510_TXB0	Transmit buffer 0
CAN2510_TXB1	Transmit buffer 1
CAN2510_TXB2	Transmit buffer 2

bufferPriority
Priority of buffer. One of the following values:

CAN2510_PRI_HIGHEST	Highest message priority
CAN2510_PRI_HIGH	High message priority
CAN2510_PRI_LOW	Low message priority
CAN2510_PRI_LOWEST	Lowest message priority

Remarks: This function loads the specified priority of an individual buffer.

File Name: cansetpr.c

CAN2510SetMode

Function: Configures the MCP2510 mode of operation.

Required CAN Mode(s): All

Include: can2510.h

Prototype:
```
void CAN2510SetMode( unsigned char mode );
```

Arguments:

mode

The value of ***mode*** can be one of the following values (defined in can2510.h). Controlled by the REQOP2:REQOP0 bits (CANCTRL register). One of the following values:

CAN2510_MODE_CONFIG	Configuration registers can be modified
CAN2510_MODE_NORMAL	Normal (send and receive messages)
CAN2510_MODE_SLEEP	Wait for interrupt
CAN2510_MODE_LISTEN	Listen only, don't send
CAN2510_MODE_LOOPBACK	Used for testing, messages stay internal

Remarks: This function configures the specified mode. The mode will not change until all pending message transmissions are complete.

File Name: canmodes.c

CAN2510SetMsgFilterStd

Function: Configures ALL of the filter and mask values of the specific receive buffer for a standard message.

Required CAN Mode(s): Configuration mode

Include: can2510.h

Prototype:
```
unsigned char CAN2510SetMsgFilteringStd(
    unsigned char bufferNum,
    unsigned int  mask,
    unsigned int  *filters );
```

Arguments:

bufferNum

Specifies the receive buffer to configure the mask and filters for. One of the following values:

CAN2510_RXB0	Configure RXM0, RXF0 and RXF1
CAN2510_RXB1	Configure RXM1, RXF2, RXF3, RXF4 and RXF5

mask

Value to store in the corresponding mask

filters

Array of filter values.

For Buffer 0

Standard-length messages: Array of 2 unsigned integers

For Buffer 1

Standard-length messages: Array of 4 unsigned integers

CAN2510SetMsgFilterStd (Continued)

Remarks: This function configures the MCP2510 into Configuration mode, then writes the mask and filter values out to the appropriate registers. Before returning, it configures the MCP2510 to the original mode.

Return Value: Indicates if the MCP2510 modes could be modified properly.
0 if initialization and restoration of Operating mode completed
-1 if initialization and restoration of Operating mode did not complete

File Name: `canfms.c`

CAN2510SetMsgFilterXtd

Function: Configures ALL of the filter and mask values of the specific receive buffer for a extended message.

Required CAN Mode(s): Configuration mode

Include: `can2510.h`

Prototype:

```
unsigned char CAN2510SetMsgFilteringXtd(
    unsigned char bufferNum,
    unsigned long mask,
    unsigned long *filters );
```

Arguments:

bufferNum
Specifies the receive buffer to configure the mask and filters for one of the following values:

`CAN2510_RXB0`	Configure RXM0, RXF0 and RXF1
`CAN2510_RXB1`	Configure RXM1, RXF2, RXF3, RXF4 and RXF5

mask
Value to store in the corresponding mask

filters
Array of filter values.
For Buffer 0
Extened-length messages: Array of 4 unsigned integers
For Buffer 1
Extened-length messages: Array of 8 unsigned integers

Remarks: This function configures the MCP2510 into Configuration mode, then writes the mask and filter values out to the appropriate registers. Before returning, it configures the MCP2510 to the original mode.

Return Value: Indicates if the MCP2510 modes could be modified properly:
0 if Initialization and restoration of Operating mode completed
-1 if initialization and restoration of Operating mode did not complete

File Name: `canfmx.c`

CAN2510SetSingleFilterStd

Function: Configures the specified Receive filter with a filter value for a Standard (Std) message.

Required CAN Mode(s): Configuration mode

Include: can2510.h

Prototype:
```
void CAN2510SetSingleFilterStd(
    unsigned char filterNum,
    unsigned long filter );
```

Arguments:

filterNum
Specifies the acceptance filter to configure. One of the following values:

CAN2510_RXF0	Configure RXF0 (for RXB0)
CAN2510_RXF1	Configure RXF1 (for RXB0)
CAN2510_RXF2	Configure RXF2 (for RXB1)
CAN2510_RXF3	Configure RXF3 (for RXB1)
CAN2510_RXF4	Configure RXF4 (for RXB1)
CAN2510_RXF5	Configure RXF5 (for RXB1)

filter
Value to store in the corresponding filter

Remarks: This function writes the filter value to the appropriate registers. The MCP2510 must be in Configuration mode before executing this function.

File Name: canfilts.c

CAN2510SetSingleFilterXtd

Function: Configures the specified Receive filter with a filter value for a Extended (Xtd) message.

Required CAN Mode(s): Configuration mode

Include: can2510.h

Prototype:
```
void CAN2510SetSingleFilterXtd(
    unsigned char filterNum,
    unsigned int  filter );
```

Arguments:

filterNum
Specifies the acceptance filter to configure. One of the following values:

CAN2510_RXF0	Configure RXF0	(for RXB0)
CAN2510_RXF1	Configure RXF1	(for RXB0)
CAN2510_RXF2	Configure RXF2	(for RXB1)
CAN2510_RXF3	Configure RXF3	(for RXB1)
CAN2510_RXF4	Configure RXF4	(for RXB1)
CAN2510_RXF5	Configure RXF5	(for RXB1)

filter
Value to store in the corresponding filter

Remarks: This function writes the filter value to the appropriate registers. The MCP2510 must be in Configuration mode before executing this function.

File Name: canfiltx.c

CAN2510SetSingleMaskStd

Function: Configures the specified Receive buffer mask with a mask value for a Standard (Std) format message.

Required CAN Mode(s): Configuration mode

Include: `can2510.h`

Prototype:
```
unsigned char CAN2510SetSingleMaskStd(
    unsigned char maskNum,
    unsigned int  mask );
```

Arguments:

maskNum
Specifies the acceptance mask to configure. One of the following values:

`CAN2510_RXM0`	Configure RXM0 (for RXB0)
`CAN2510_RXM1`	Configure RXM1 (for RXB1)

mask
Value to store in the corresponding mask

Remarks: This function writes the mask value to the appropriate registers. The MCP2510 must be in Configuration mode before executing this function.

File Name: `canmasks.c`

CAN2510SetSingleMaskXtd

Function: Configures the specified Receive buffer mask with a mask value for an Extended (Xtd) message.

Required CAN Mode(s): Configuration mode

Include: `can2510.h`

Prototype:
```
unsigned char CAN2510SetSingleMaskXtd(
    unsigned char maskNum,
    unsigned long mask );
```

Arguments:

maskNum
Specifies the acceptance mask to configure. One of the following values:

`CAN2510_RXM0`	Configure RXM0 (for RXB0)
`CAN2510_RXM1`	Configure RXM1 (for RXB1)

mask
Value to store in the corresponding mask

Remarks: This function writes the mask value to the appropriate registers. The MCP2510 must be in Configuration mode before executing this function.

File Name: `canmaskx.c`

CAN2510WriteStd

Function: Writes a Standard format message out to the CAN bus using the first available transmit buffer.

Required CAN Mode(s): Normal mode

Include: can2510.h

Prototype:
```
unsigned char CAN2510WriteStd(
    unsigned int  msgId,
    unsigned char msgPriority,
    unsigned char numBytes,
    unsigned char *data );
```

Arguments:

msgId
CAN message identifier, 11 bits for a standard message. This 11-bit identifier is stored in the lower 11 bits of msgId (an unsigned integer).

msgPriority
Priority of buffer. One of the following values:

CAN2510_PRI_HIGHEST	Highest message priority
CAN2510_PRI_HIGH	High intermediate message priority
CAN2510_PRI_LOW	Low intermediate message priority
CAN2510_PRI_LOWEST	Lowest message priority

numBytes
Number of bytes of data to transmit, from 0 to 8. If value is greater than 8, only the first 8 bytes of data will be sent.

data
Array of data values to be written. Must be at least as large as the value specified in ***numBytes***.

Remarks: This function will query each transmit buffer for a pending message, and will post the specified message into the first available buffer.

Return Value: Value indicates which buffer was used to transmit the message (0, 1 or 2).
-1 indicates that no message was sent.

File Name: canwrits.c

CAN2510WriteXtd

Function: Writes an Extended format message out to the CAN bus using the first available transmit buffer.

Required CAN Mode(s): Normal mode

Include: can2510.h

Prototype:
```
unsigned char CAN2510WriteXtd(
    unsigned long msgId,
    unsigned char msgPriority,
    unsigned char numBytes,
    unsigned char *data );
```

Arguments:

msgId
CAN message identifier, 29 bits for an extended message. This 29-bit identifier is stored in the lower 29 bits of msgId (an unsigned long).

CAN2510WriteXtd (Continued)

msgPriority
Priority of buffer. One of the following values:

CAN2510_PRI_HIGHEST	Highest message priority
CAN2510_PRI_HIGH	High intermediate message priority
CAN2510_PRI_LOW	Low intermediate message priority
CAN2510_PRI_LOWEST	Lowest message priority

numBytes
Number of bytes of data to transmit, from 0 to 8. If value is greater than 8, only the first 8 bytes of data will be sent.

data
Array of data values to be written. Must be at least as large as the value specified in numBytes.

Remarks: This function will query each transmit buffer for a pending message, and will post the specified message into the first available buffer.

Return Value: Value indicates which buffer was used to transmit the message (0, 1 or 2).
-1 indicates that no message was sent.

File Name: canwritx.c

3.4 SOFTWARE I²C FUNCTIONS

These functions are designed to allow the implementation of an I^2C bus using I/O pins from a PIC18 microcontroller. The following functions are provided:

Function	Description
`Clock_test`	Generate a delay for slave clock stretching.
`SWAckI2C`	Generate an I^2C bus *Acknowledge* condition.
`SWGetcI2C`	Read a byte from the I^2C bus.
`SWGetsI2C`	Read a data string.
`SWNotAckI2C`	Generate an I^2C bus *Acknowledge* condition.
`SWPutI2C`	Write a single byte to the I^2C bus.
`SWPutsI2C`	Write a string to the I^2C bus.
`SWReadI2C`	Read a byte from the I^2C bus.
`SWRestartI2C`	Generate an I^2C bus *Restart* condition.
`SWStartI2C`	Generate an I^2C bus *START* condition.
`SWStopI2C`	Generate an I^2C bus *STOP* condition.
`SWWriteI2C`	Write a single byte to the I^2C bus.

The precompiled versions of these functions use default pin assignments that can be changed by redefining the macro assignments in the file `sw_i2c.h`, found in the `h` subdirectory of the compiler installation:

I^2C Line	Macros	Default Value	Use
DATA Pin	`DATA_PIN`	`PORTBbits.RB4`	Pin used for the DATA line.
	`DATA_LAT`	`LATBbits.RB4`	Latch associated with DATA pin.
	`DATA_LOW`	`TRISBbits.TRISB4 = 0;`	Statement to configure the DATA pin as an output.
	`DATA_HI`	`TRISBbits.TRISB4 = 1;`	Statement to configure the DATA pin as an input.
CLOCK Pin	`SCLK_PIN`	`PORTBbits.RB3`	Pin used for the CLOCK line.
	`SCLK_LAT`	`LATBbits.LATB3`	Latch associated with the CLOCK pin.
	`CLOCK_LOW`	`TRISBbits.TRISB3 = 0;`	Satement to configure the CLOCK pin as an output.
	`CLOCK_HI`	`TRISBbits.TRISB3 = 1;`	Statement to configure the CLOCK pin as an input.

After these definitions have been made, the user must recompile the I^2C routines and then use the updated files in the project. This can be accomplished by adding the library source files into the project or by recompiling the library files using the provided batch files.

3.4.1 Function Descriptions

Clock_test

Function: Generate a delay for slave clock stretching.

Include: `sw_i2c.h`

Prototype:
```
unsigned char Clock_test( void );
```

Remarks: This function is called to allow for slave clock stretching. The delay time may need to be adjusted per application requirements. If at the end of the delay period the clock line is low, a value is returned indicating clock error.

Return Value: 0 is returned if no clock error occurred
-2 is returned if a clock error occurred

File Name: `swckti2c.c`

SWAckI2C
SWNotAckI2C

Function: Generate an I^2C bus *Acknowledge* condition.

Include: `sw_i2c.h`

Prototype:
```
unsigned char SWAckI2C( void );
unsigned char SWNotAckI2C( void );
```

Remarks: This function is called to generate an I^2C bus Acknowledge sequence.

Return Value: 0 if the slave Acknowledges
-1 if the slave does not Acknowledge

File Name: `swacki2c.c`

SWGetcI2C

See **SWReadI2C.**

SWGetsI2C

Function: Read a string from the I^2C bus.

Include: `sw_i2c.h`

Prototype:
```
unsigned char SWGetsI2C(
      unsigned char *rdptr,
      unsigned char length );
```

Arguments:
rdptr
Location to store the data read from the I^2C bus.
length
Number of bytes to read.

Remarks: This function reads in a string of predetermined length.

Return Value: -1 if the master generated a *NOT ACK* bus condition before all bytes have been received
0 otherwise

File Name: `swgtsi2c.c`

SWGetsI2C (Continued)

Code Example:
```
char x[10];
SWGetsI2C( x,5 );
```

SWNotAckI2C

See **SWAckI2C**.

SWPutcI2C

See **SWWriteI2C**.

SWPutsI2C

Function: Write a string to the I^2C bus.

Include: sw_i2c.h

Prototype:
```
unsigned char SWPutsI2C(
      unsigned char *wrdptr );
```

Arguments:
wrdptr
Pointer to data to be written to the I^2C bus.

Remarks: This function writes out a data string up to (but not including) a null character.

Return Value: -1 if there was an error writing to the I^2C bus
0 otherwise

File Name: swptsi2c.c

Code Example:
```
char mybuff [20];
SWPutsI2C(mybuff);
```

SWReadI2C
SWGetcI2C

Function: Read a byte from the I^2C bus.

Include: sw_i2c.h

Prototype:
```
unsigned char SWReadI2C( void );
```

Remarks: This function reads in a single data byte by generating the appropriate signals on the predefined I^2C clock line.

Return Value: This function returns the acquired I^2C data byte.
-1 if there was an error in this function.

File Name: swgtci2c.c

SWRestartI2C

Function: Generate an I^2C *Restart* bus condition.

Include: `sw_i2c.h`

Prototype:
```
void SWRestartI2C( void );
```

Remarks: This function is called to generate an I^2C bus restart condition.

File Name: `swrsti2c.c`

SWStartI2C

Function: Generate an I^2C bus *START* condition.

Include: `sw_i2c.h`

Prototype:
```
void SWStartI2C( void );
```

Remarks: This function is called to generate an I^2C bus START condition.

File Name: `swstri2c.c`

SWStopI2C

Function: Generate an I^2C bus *STOP* condition.

Include: `sw_i2c.h`

Prototype:
```
void SWStopI2C( void );
```

Remarks: This function is called to generate an I^2C bus STOP condition.

File Name: `swstpi2c.c`

SWWriteI2C
SWPutcI2C

Function: Write a byte to the I^2C bus.

Include: `sw_i2c.h`

Prototype:
```
unsigned char SWWriteI2C(
      unsigned char data_out );
```

Arguments:

data_out
Single data byte to be written to the I^2C device.

Remarks: This function writes out a single data byte to the predefined data pin.

Return Value: 0 if write is successful
-1 if there was an error condition

File Name: `swptci2c.c`

Code Example
```
if(SWWriteI2C(0x80))
  {
    errorHandler();
  }
```

3.4.2 Example of Use

The following is a simple code example illustrating a software I^2C implementation communicating with a Microchip 24LC01B I^2C EE memory device.

```
#include <p18cxxx.h>
#include <sw_i2c.h>
#include <delays.h>

// FUNCTION Prototype
void main(void);
void byte_write(void);
void page_write(void);
void current_address(void);
void random_read(void);
void sequential_read(void);
void ack_poll(void);
unsigned char warr[] = {8,7,6,5,4,3,2,1,0};
unsigned char rarr[15];
unsigned char far *rdptr = rarr;
unsigned char far *wrptr = warr;
unsigned char var;

#define W_CS  PORTA.2

//*************************************************
void main( void )
{
  byte_write();
  ack_poll();
  page_write();
  ack_poll();
  Nop();
  sequential_read();
  Nop();
  while (1);  // Loop indefinitely
}

void byte_write( void )
{
  SWStartI2C();
  var = SWPutcI2C(0xA0); // control byte
  SWAckI2C();
  var = SWPutcI2C(0x10); // word address
  SWAckI2C();
  var = SWPutcI2C(0x66); // data
  SWAckI2C();
  SWStopI2C();
}

void page_write( void )
{
  SWStartI2C();
  var = SWPutcI2C(0xA0); // control byte
  SWAckI2C();
  var = SWPutcI2C(0x20); // word address
  SWAckI2C();
  var = SWPutsI2C(wrptr); // data
  SWStopI2C();
}
```

```
void sequential_read( void )
{
  SWStartI2C();
  var = SWPutcI2C( 0xA0 ); // control byte
  SWAckI2C();
  var = SWPutcI2C( 0x00 ); // address to read from
  SWAckI2C();
  SWRestartI2C();
  var = SWPutcI2C( 0xA1 );
  SWAckI2C();
  var = SWGetsI2C( rdptr, 9 );
  SWStopI2C();
}

void current_address( void )
{
  SWStartI2C();
  SWPutcI2C( 0xA1 ); // control byte
  SWAckI2C();
  SWGetcI2C();     // word address
  SWNotAckI2C();
  SWStopI2C();
}

void ack_poll( void )
{
  SWStartI2C();
  var = SWPutcI2C( 0xA0 );  // control byte
  while( SWAckI2C() )
  {
    SWRestartI2C();
    var = SWPutcI2C(0xA0); // data
  }
  SWStopI2C();
}
```

3.5 SOFTWARE SPI FUNCTIONS

These functions are designed to allow the implementation of an SPI using I/O pins from a PIC18 microcontroller. The following functions are provided:

Function	Description
ClearSWCSSPI	Clear the chip select ($\overline{CS}$) pin.
OpenSWSPI	Configure the I/O pins for use as an SPI.
putcSWSPI	Write a byte of data to the software SPI.
SetSWCSSPI	Set the chip select ($\overline{CS}$) pin.
WriteSWSPI	Write a byte of data to the software SPI bus.

The precompiled versions of these functions use default pin assignments that can be changed by redefining the macro assignments in the file sw_spi.h, found in the h subdirectory of the compiler installation:

LCD Controller Line	Macros	Default Value	Use
$\overline{CS}$ Pin	SW_CS_PIN TRIS_SW_CS_PIN	PORTBbits.RB2 TRISBbits.TRISB2	Pin used for the chip select ($\overline{CS}$) line. Bit that controls the direction of the pin associated with the $\overline{CS}$ line.
DIN Pin	SW_DIN_PIN TRIS_SW_DIN_PIN	PORTBbits.RB3 TRISBbits.TRISB3	Pin used for the DIN line. Bit that controls the direction of the pin associated with the DIN line.
DOUT Pin	SW_DOUT_PIN TRIS_SW_DOUT_PIN	PORTBbits.RB7 TRISBbits.TRISB7	Pin used for the DOUT line. Bit that controls the direction of the pin associated with the DOUT line.
SCK Pin	SW_SCK_PIN TRIS_SW_SCK_PIN	PORTBbits.RB6 TRISBbits.TRISB6	Pin used for the SCK line. Bit that controls the direction of the pin associated with the SCK line.

The libraries that are provided can operate in one of four modes. The table below lists the macros used for selecting between these modes. Exactly one of these must be defined when rebuilding the software SPI libraries.

Macro	Default Value	Meaning
MODE0	defined	CKP = 0 CKE = 0
MODE1	not defined	CKP = 1 CKE = 0
MODE2	not defined	CKP = 0 CKE = 1
MODE3	not defined	CKP = 1 CKE = 1

After these definitions have been made, the user must recompile the software SPI routines and then include the updated files in the project. This can be accomplished by adding the software SPI source files into the project or by recompiling the library files using the provided batch files.

3.5.1 Function Descriptions

ClearSWCSSPI

Function:	Clear the chip select ($\overline{CS}$) pin that is specified in the `sw_spi.h` header file.
Include:	`sw_spi.h`
Prototype:	`void ClearSWCSSPI( void );`
Remarks:	This function clears the I/O pin that is specified in `sw_spi.h` to be the chip select ($\overline{CS}$) pin for the software SPI.
File Name:	`clrcsspi.c`

OpenSWSPI

Function:	Configure the I/O pins for the software SPI.
Include:	`sw_spi.h`
Prototype:	`void OpenSWSPI( void );`
Remarks:	This function configures the I/O pins used for the software SPI to the correct input or ouput state and logic level.
File Name:	`opensspi.c`

putcSWSPI

See **WriteSWSPI.**

SetSWCSSPI

Function:	Set the chip select ($\overline{CS}$) pin that is specified in the `sw_spi.h` header file.
Include:	`sw_spi.h`
Prototype:	`void SetSWCSSPI( void );`
Remarks:	This function sets the I/O pin that is specified in `sw_spi.h` to be the chip select ($\overline{CS}$) pin for the software SPI.
File Name:	`setcsspi.c`

WriteSWSPI
putcSWSPI

Function:	Write a byte to the software SPI.
Include:	sw_spi.h
Prototype:	char WriteSWSPI(char ***data***);
Arguments:	***data*** Data to be written to the software SPI.
Remarks:	This function writes the specified byte of data out the software SPI and returns the byte of data that was read. This function does not provide any control of the chip select pin ($\overline{CS}$).
Return Value:	This function returns the byte of data that was read from the data in (DIN) pin of the software SPI.
File Name:	wrtsspi.c

Code Example:	char addr = 0x10;
	char result;
	result = WriteSWSPI(addr);

3.5.2 Example of Use

```
#include <p18C452.h>
#include <sw_spi.h>
#include <delays.h>

void main( void )
{
  char address;

  // configure software SPI
  OpenSWSPI();

  for( address=0; address<0x10; address++ )
  {
    ClearCSSWSPI();        //clear CS pin
    WriteSWSPI( 0x02 );    //send write cmd
    WriteSWSPI( address ); //send address hi
    WriteSWSPI( address ); //send address low
    SetCSSWSPI();          //set CS pin
    Delay10KTCYx( 50 );    //wait 5000,000TCY
  }
}
```

3.6 SOFTWARE UART FUNCTIONS

These functions are designed to allow the implementation of a UART using I/O pins from a PIC18 microcontroller. The following functions are provided:

Function	Description
getcUART	Read a byte from the software UART.
getsUART	Read a string from the software UART.
OpenUART	Configure I/O pins for use as a UART.
putcUART	Write a byte to the software UART.
putsUART	Write a string to the software UART.
ReadUART	Read a byte from the software UART.
WriteUART	Write a byte to the software UART.

The precompiled versions of these functions use default pin assignments that can be changed by redefining the equate (equ) statements in the files `writuart.asm`, `readuart.asm` and `openuart.asm`, found in the `src/pmc/sw_uart/18Cxx` subdirectory of the compiler installation:

LCD Controller Line	Definition	Default Value	Use
TX Pin	`SWTXD`	`PORTB`	Port used for the transmit line.
	`SWTXDpin`	`4`	Bit in the `SWTXD` port used for the TX line.
	`TRIS_SWTXD`	`TRISB`	Data direction register associated with the port used for the TX line.
RX Pin	`SWRXD`	`PORTB`	Port used for the receive line.
	`SWRXDpin`	`5`	Bit in the `SWRXD` port used for the RX line.
	`TRIS_SWRXD`	`TRISB`	Data direction register associated with the port used for the RX line.

If changes to these definitions are made, the user must recompile the software UART routines and then include the updated files in the project. This can be accomplished by adding the software UART source files into the project or by recompiling the library files using the batch files provided with the MPLAB C18 compiler installation.

The XLCD libraries also require that the following functions be defined by the user to provide the appropriate delays:

Function	Behavior
`DelayTXBitUART`	Delay for: ((((2*FOSC) / (4*baud)) + 1) / 2) - 12 cycles
`DelayRXHalfBitUART`	Delay for: ((((2*FOSC) / (8*baud)) + 1) / 2) - 9 cycles
`DelayRXBitUART`	Delay for: ((((2*FOSC) / (4*baud)) + 1) / 2) - 14 cycles

3.6.1 Function Descriptions

getcUART

See **ReadUART.**

getsUART

Function: Read a string from the software UART.

Include: sw_uart.h

Prototype:
```
void getsUART( char * buffer,
               unsigned char len);
```

Arguments:
buffer
Pointer to the string of characters read from the software UART.
len
Number of characters to be read from the software UART.

Remarks: This function reads ***len*** characters from the software UART and places them in ***buffer***.

File Name: getsuart.c

Code Example:
```
char x[10];
getsUART( x, 5 );
```

OpenUART

Function: Configure the I/O pins for the software UART.

Include: sw_uart.h

Prototype:
```
void OpenUART( void );
```

Remarks: This function configures the I/O pins used for the software UART to the correct input or ouput state and logic level.

File Name: openuart.asm

Code Example:
```
OpenUART();
```

putcUART

See **WriteUART.**

putsUART

Function: Write a string to the software UART.

Include: sw_uart.h

Prototype:
```
void putsUART( char * buffer );
```

Arguments:
buffer
String to be written to the software UART.

Remarks: This function writes a string of characters to the software UART. The entire string including the null is sent to the UART.

File Name: putsuart.c

Code Example:
```
char mybuff [20];
putsUART( mybuff );
```

ReadUART
getcUART

Function: Read a byte from the software UART.

Include: `sw_uart.h`

Prototype: `char ReadUART( void );`

Remarks: This function reads a byte of data out the software UART.

Return Value: Returns the byte of data that was read from the receive data (RXD) pin of the software UART.

File Name: `readuart.asm`

Code Example:
```
char x;
x = ReadUART();
```

WriteUART
putcUART

Function: Write a byte to the software UART.

Include: `sw_uart.h`

Prototype: `void WriteUART( char data );`

Arguments: ***data***
Byte of data to be written to software UART.

Remarks: This function writes the specified byte of data out the software UART.

File Name: `writuart.asm`

Code Example:
```
char x = 'H';
WriteUART( x );
```

3.6.2 Example of Use

```
#include <p18C452.h>
#include <sw_uart.h>

void main( void )
{
  char data

  // configure software UART
  OpenUART();

  while( 1 )
  {
    data = ReadUART();   //read a byte
    WriteUART( data );   //bounce it back
  }
}
```

NOTES:

Chapter 4. General Software Library

4.1 INTRODUCTION

This chapter documents general software library functions found in the precompiled clib.lib file. The source code for all of these functions is included with MPLAB C18 in the following subdirectories of the compiler installation:

- src\string
- src\stdlib
- src\delays
- src\ctype

The following categories of routines are supported by the MPLAB C18 library:

- Character Classification Functions
- Data Conversion Functions
- Delay Functions
- Memory and String Manipulation Functions

4.2 CHARACTER CLASSIFICATION FUNCTIONS

These functions are consistent with the ANSI 1989 standard C library functions of the same name. The following functions are provided:

Function	Description
isalnum	Determine if a character is alphanumeric.
isalpha	Determine if a character is alphabetic.
iscntrl	Determine if a character is a control character.
isdigit	Determine if a character is a decimal digit.
isgraph	Determine if a character is a graphical character.
islower	Determine if a character is a lower case alphabetic character.
isprint	Determine if a character is a printable character.
ispunct	Determine if a character is a punctuation character.
isspace	Determine if a character is a white space character.
isupper	Determine if a character is an upper case alphabetic character.
isxdigit	Determine if a character is a hexadecimal digit.

4.2.1 Function Descriptions

isalnum

Function: Determine if a character is alphanumeric.

Include: `ctype.h`

Prototype: `unsigned char isalnum( unsigned char ch );`

Arguments: ***ch***
Character to be checked.

Remarks: A character is considered to be alphanumeric if it is in the range of ‘A’ to ‘Z’, ‘a’ to ‘z’ or ‘0’ to ‘9’.

Return Value: Non-zero if the character is alphanumeric
Zero otherwise

File Name: `isalnum.c`

isalpha

Function: Determine if a character is alphabetic.

Include: `ctype.h`

Prototype: `unsigned char isalpha( unsigned char ch );`

Arguments: ***ch***
Character to be checked.

Remarks: A character is considered to be alphabetic if it is in the range of ‘A’ to ‘Z’ or ‘a’ to ‘z’.

Return Value: Non-zero if the character is alphabetic
Zero otherwise

File Name: `isalpha.c`

iscntrl

Function: Determine if a character is a control character.

Include: `ctype.h`

Prototype: `unsigned char iscntrl( unsigned char ch );`

Arguments: ***ch***
Character to be checked.

Remarks: A character is considered to be a control character if it is not a printable character as defined by `isprint()`.

Return Value: Non-zero if the character is a control character
Zero otherwise

File Name: `iscntrl.c`

isdigit

Function: Determine if a character is a decimal digit.

Include: `ctype.h`

Prototype: `unsigned char isdigit( unsigned char ch );`

Arguments: ***ch***
Character to be checked.

Remarks: A character is considered to be a digit character if it is in the range of '0' to '9'.

Return Value: Non-zero if the character is a digit character
Zero otherwise

File Name: `isdigit.c`

isgraph

Function: Determine if a character is a graphical character.

Include: `ctype.h`

Prototype: `unsigned char isgraph( unsigned char ch );`

Arguments: ***ch***
Character to be checked.

Remarks: A character is considered to be a graphical case alphabetic character if it is any printable character except space.

Return Value: Non-zero if the character is a graphical character
Zero otherwise

File Name: `isgraph.c`

islower

Function: Determine if a character is a lower case alphabetic character.

Include: `ctype.h`

Prototype: `unsigned char islower( unsigned char ch );`

Arguments: ***ch***
Character to be checked.

Remarks: A character is considered to be a lower case alphabetic character if it is in the range of 'a' to 'z'.

Return Value: Non-zero if the character is a lower case alphabetic character
Zero otherwise

File Name: `islower.c`

isprint

Function: Determine if a character is a printable character.

Include: `ctype.h`

Prototype: `unsigned char isprint( unsigned char ch );`

Arguments: ***ch***
Character to be checked.

Remarks: A character is considered to be a printable character if it is in the range 0x20 to 0x7e, inclusive.

Return Value: Non-zero if the character is a printable character
Zero otherwise

File Name: `isprint.c`

ispunct

Function: Determine if a character is a punctuation character.

Include: `ctype.h`

Prototype: `unsigned char ispunct( unsigned char ch );`

Arguments: ***ch***
Character to be checked.

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.

Return Value: Non-zero if the character is a punctuation character
Zero otherwise

File Name: `ispunct.c`

isspace

Function: Determine if a character is a white space character.

Include: `ctype.h`

Prototype: `unsigned char isspace (unsigned char ch);`

Arguments: ***ch***
Character to be checked.

Remarks: A character is considered to be a white space character if it is one of the following: space (‘ ’), tab(‘\t’), carriage return (‘\r’), new line (‘\n’), form feed (‘\f’) or vertical tab (‘\v’).

Return Value: Non-zero if the character is a white space character
Zero otherwise

File Name: `isspace.c`

isupper

Function: Determine if a character is an upper case alphabetic character.

Include: `ctype.h`

Prototype: `unsigned char isupper (unsigned char ch);`

Arguments: ***ch***
Character to be checked.

Remarks: A character is considered to be an upper case alphabetic character if it is in the range of ‘A’ to ‘Z’.

Return Value: Non-zero if the character is an upper case alphabetic character
Zero otherwise

File Name: `isupper.c`

isxdigit

Function: Determine if a character is a hexadecimal digit.

Include: `ctype.h`

Prototype: `unsigned char isxdigit( unsigned char ch );`

Arguments: ***ch***
Character to be checked.

Remarks: A character is considered to be a HEX digit character if it is in the range of ‘0’ to ‘9’, ‘a’ to ‘f’ or ‘A’ to ‘F’.

Return Value: Non-zero if the character is a HEX digit character
Zero otherwise

File Name: `isxdig.c`

4.3 DATA CONVERSION FUNCTIONS

Except as noted in the function descriptions, these functions are consistent with the ANSI 1989 standard C library functions of the same name. The following functions are provided:

Function	Description
atob	Convert a string to an 8-bit signed byte.
atof	Convert a string into a floating point value.
atoi	Convert a string to a 16-bit signed integer.
atol	Convert a string into a long integer representation.
btoa	Convert an 8-bit signed byte to a string.
itoa	Convert a 16-bit signed integer to a string.
ltoa	Convert a signed long integer to a string.
rand	Generate a pseudo-random integer.
srand	Set the starting seed for the pseudo-random number generator.
tolower	Convert a character to a lower case alphabetical ASCII character.
toupper	Convert a character to an upper case alphabetical ASCII character.
ultoa	Convert an unsigned long integer to a string.

4.3.1 Function Descriptions

atob

Function: Convert a string to an 8-bit signed byte.

Include: `stdlib.h`

Prototype:
```
signed char atob( const char * s );
```

Arguments: *s*
Pointer to ASCII string to be converted.

Remarks: This function converts the ASCII string *s* into an 8-bit signed byte (-128 to 127). The input string must be in base 10 (decimal radix) and can begin with a character indicating sign ('+' or '-'). Overflow results are undefined. This function is an MPLAB C18 extension to the ANSI standard libraries.

Return Value: 8-bit signed byte for all strings in the range (-128 to 127).

File Name: `atob.asm`

atof

Function: Convert a string into a floating point value.

Include: `stdlib.h`

Prototype:
```
double atof ( const char * s );
```

Arguments: ***s***
Pointer to ASCII string to be converted.

Remarks: This function converts the ASCII string ***s*** into a floating point value. Examples of floating point strings that are recognized are:
```
-3.1415
1.0E2
1.0E+2
1.0E-2
```

Return Value: The function returns the converted value.

File Name: `atof.c`

atoi

Function: Convert a string to a 16-bit signed integer.

Include: `stdlib.h`

Prototype:
```
int atoi( const char * s );
```

Arguments: ***s***
Pointer to ASCII string to be converted.

Remarks: This function converts the ASCII string ***s*** into an 16-bit signed integer (-32768 to 32767). The input string must be in base 10 (decimal radix) and can begin with a character indicating sign ('+' or '-'). Overflow results are undefined. This function is an MPLAB C18 extension to the ANSI standard libraries.

Return Value: 16-bit signed integer for all strings in the range (-32768 to 32767).

File Name: `atoi.asm`

atol

Function: Convert a string into a long integer representation.

Include: `stdlib.h`

Prototype:
```
long atol( const char * s );
```

Arguments: ***s***
Pointer to ASCII string to be converted.

Remarks: This function converts the ASCII string ***s*** into a long value. The input string must be in base 10 (decimal radix) and can begin with a character indicating sign ('+' or '-'). Overflow results are undefined. This function is an MPLAB C18 extension to the ANSI standard libraries.

Return Value: The function returns the converted value.

File Name: `atol.asm`

btoa

Function: Convert an 8-bit signed byte to a string.

Include: stdlib.h

Prototype:
```
char * btoa( signed char value,
             char * string );
```

Arguments:

value
An 8-bit signed byte.

string
Pointer to ASCII string that will hold the result. ***string*** must be long enough to hold the ASCII representation, including the sign character for negative values and a trailing null character.

Remarks: This function converts the 8-bit signed byte in the argument ***value*** to a ASCII string representation.

This function is an MPLAB C18 extension of the ANSI required libraries.

Return Value: Pointer to the result ***string***.

File Name: btoa.asm

itoa

Function: Convert a 16-bit signed integer to a string.

Include: stdlib.h

Prototype:
```
char * itoa( int value,
             char * string );
```

Arguments:

value
An 8-bit signed byte.

string
Pointer to ASCII string that will hold the result. ***string*** must be long enough to hold the ASCII representation, including the sign character for negative values and a trailing null character.

Remarks: This function converts the 16-bit signed integer in the argument ***value*** to a ASCII string representation.

This function is an MPLAB C18 extension of the ANSI required libraries.

Return Value: Pointer to the result ***string***.

File Name: itoa.asm

ltoa

Function: Convert a signed long integer to a string.

Include: `stdlib.h`

Prototype:
```
char * ltoa( long value,
             char * string );
```

Arguments:

value
A signed long integer to be converted.

string
Pointer to ASCII string that will hold the result.

Remarks: This function converts the signed long integer in the argument ***value*** to a ASCII string representation. ***string*** must be long enough to hold the ASCII representation, including the sign character for negative values and a trailing null character. This function is an MPLAB C18 extension to the ANSI required libraries.

Return Value: Pointer to the result ***string***.

File Name: `ltoa.asm`

rand

Function: Generate a pseudo-random integer.

Include: `stdlib.h`

Prototype:
```
int rand( void );
```

Remarks: Calls to this function return pseudo-random integer values in the range [0,32767]. 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 identical seed values are used.

Return Value: A psuedo-random integer value.

File Name: `rand.asm`

srand

Function: Set the starting seed for the pseudo-random number sequence.

Include: `stdlib.h`

Prototype:
```
void rand( unsigned int seed );
```

Arguments:

seed
The starting value for the pseudo-random number sequence.

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 without `srand()` having first been called, the sequence of numbers generated will be the same as if `srand()` had been called with a seed value of 1.

File Name: `rand.asm`

tolower

Function: Convert a character to a lower case alphabetical ASCII character.

Include: `ctype.h`

Prototype:
```
char tolower( char ch );
```

Arguments:
ch
Character to be converted.

Remarks: This function converts ***ch*** to a lower case alphabetical ASCII character provided that the argument is a valid upper case alphabetical character.

Return Value: This function returns a lower case character if the argument was upper case to begin with; otherwise the original character is returned.

File Name: `tolower.c`

toupper

Function: Convert a character to an upper case alphabetical ASCII character.

Include: `ctype.h`

Prototype:
```
char toupper( char ch );
```

Arguments:
ch
Character to be converted.

Remarks: This function converts ***ch*** to a upper case alphabetical ASCII character provided that the argument is a valid lower case alphabetical character.

Return Value: This function returns a lower case character if the argument was upper case to begin with; otherwise the original character is returned.

File Name: `toupper.c`

ultoa

Function: Convert an unsigned long integer to a string.

Include: `stdlib.h`

Prototype:
```
char * ultoa( unsigned long value,
              char * string);
```

Arguments:
value
An unsigned long integer to be converted.
string
Pointer to ASCII string that will hold the result.

Remarks: This function converts the unsigned long integer in the argument ***value*** to a ASCII string representation. ***string*** must be long enough to hold the ASCII representation, including a trailing null character. This function is an MPLAB C18 extension to the ANSI required libraries.

Return Value: Pointer to the result ***string***.

File Name: `ultoa.asm`

4.4 MEMORY AND STRING MANIPULATION FUNCTIONS

Except as noted in the function descriptions, these functions are consistent with the ANSI (1989) standard C library functions of the same name. The following functions are provided:

Function	Description
memchr	Search for a value in a specified memory region.
memcmp memcmppgm memcmppgm2ram memcmpram2pgm	Compare the contents of two arrays.
memcpy memcpypgm2ram	Copy a buffer from data or program memory into data memory.
memmove memmovepgm2ram	Copy a buffer from data or program memory into data memory.
memset	Initialize an array with a single repeated value.
strcat strcatpgm2ram	Append a copy of the source string to the end of the destination string.
strchr	Locate the first occurrence of a value in a string.
strcmp strcmppgm2ram	Compare two strings.
strcpy strcpypgm2ram	Copy a string from data or program memory into data memory.
strcspn	Calculate the number of consecutive characters at the beginning of a string that are not contained in a set of characters.
strlen	Determine the length of a string.
strlwr	Convert all upper case characters in a string to lower case.
strncat strncatpgm2ram	Append a specified number of characters from the source string to the end of the destination string.
strncmp	Compare two strings, up to a specified number of characters.
strncpy strncpypgm2ram	Copy characters from the source string into the destination string, up to the specified number of characters.
strpbrk	Search a string for the first occurrence of a character from a set of characters.
strrchr	Locate the last occurrence of a specified character in a string.
strspn	Calculate the number of consecutive characters at the beginning of a string that are contained in a set of characters.
strstr	Locate the first occurrence of a string inside another string.
strtok	Break a string into substrings, or tokens, by inserting null characters in place of specified delimiters.
strupr	Convert all lower case characters in a string to upper case.

4.4.1 Function Descriptions

memchr

Function: Locate the first occurrence of a byte value in a specified memory region.

Include: `string.h`

Prototype:
```
void * memchr( const void *mem,
               unsigned char c,
               size_t n);
```

Arguments:
mem
Pointer to a memory region.
c
Byte value to find.
n
Maximum number of bytes to search.

Remarks: This function searches up to ***n*** bytes of the region ***mem*** to find the first occurrence of ***c***.
This function differs from the ANSI specified function in that ***c*** is defined as an `unsigned char` parameter rather than an `int` parameter.

Return Value: If ***c*** appears in the first ***n*** bytes of ***mem***, this function returns a pointer to the character in ***mem***. Otherwise, it returns a null pointer.

File Names: `memchr.asm`

memcmp
memcmppgm
memcmppgm2ram
memcmpram2pgm

Function: Compare the contents of two arrays of bytes.

Include: `string.h`

Prototype:
```
signed char memcmp(
       const void * buf1,
       const void * buf2,
       size_t memsize );
signed char memcmppgm(
       const rom void * buf1,
       const rom void * buf2,
       sizerom_t memsize );
signed char memcmppgm2ram(
       const void * buf1,
       const rom void * buf2,
       sizeram_t memsize );
signed char memcmpram2pgm(
       const rom void * buf1,
       const void * buf2,
       sizeram_t memsize );
```

memcmp
memcmppgm
memcmppgm2ram
memcmpram2pgm (Continued)

Arguments: ***buf1***
Pointer to first array.
buf2
Pointer to second array.
memsize
Number of elements to be compared in arrays.

Remarks: This function compares the first ***memsize*** number of bytes in ***buf1*** to the first ***memsize*** number of bytes in ***buf2*** and returns a value indicating whether the buffers are less than, equal to or greater than each other.

Return Value: `memcmp` returns a value that is:
<0 if ***buf1*** is less than ***buf2***
==0 if ***buf1*** is the same as ***buf2***
>0 if ***buf1*** is greater than ***buf2***

File Names:
```
memcmp.asm
memcmpp2p.asm
memcmpp2r.asm
memcmpr2p.asm
```

memcpy
memcpypgm2ram

Function: Copy the contents of the source buffer into the destination buffer.

Include: `string.h`

Prototype:
```
void * memcpy(
            void * dest,
            const void * src,
            size_t memsize );
void * memcpypgm2ram(
            void * dest,
            const rom void * src,
            sizeram_t memsize );
```

Arguments: ***dest***
Pointer to destination array.
src
Pointer to source array.
memsize
Number of bytes of ***src*** array to copy into ***dest***.

Remarks: This function copies the first ***memsize*** number of bytes in ***src*** to the array ***dest***. If ***src*** and ***dest*** overlap, the behavior is undefined.

Return Value: This function returns the value of ***dest***.

File Names:
```
memcpy.asm
memcpyp2r.asm
```

memmove
memmovepgm2ram

Function: Copy the contents of the source buffer into the destination buffer, even if the regions overlap.

Include: `string.h`

Prototype:
```
void * memmove( void * dest,
                const void * src,
                size_t memsize );
void * memmovepgm2ram(
                void * dest,
                const rom void * src,
                sizeram_t memsize );
```

Arguments:
dest
Pointer to destination array.
src
Pointer to source array.
memsize
Number of bytes of ***src*** array to copy into ***dest***.

Remarks: This function copies the first ***memsize*** number of bytes in ***src*** to the array ***dest***. This function performs correctly even if ***src*** and ***dest*** overlap.

Return Value: This function returns the value of ***dest***.

File Names: `memmove.asm`
`memmovp2r.asm`

memset

Function: Copy the specified character into the destination array.

Include: `string.h`

Prototype:
```
void * memset( void * dest,
               unsigned char value,
               size_t memsize );
```

Arguments:
dest
Pointer to destination array.
value
Character value to be copied.
memsize
Number of bytes of ***dest*** into which ***value*** is copied.

Remarks: This function copies the character ***value*** into the first ***memsize*** bytes of the array ***dest***. This functions differs from the ANSI specified function in that ***value*** is defined as an `unsigned char` rather than as an `int` parameter.

Return Value: This function returns the value of ***dest***.

File Name: `memset.asm`

strcat
strcatpgm2ram

Function: Append a copy of the source string to the end of the destination string.

Include: `string.h`

Prototype:
```
char * strcat( char * dest,
               const char * src );
char * strcatpgm2ram(
               char * dest,
               const rom char * src );
```

Arguments:
dest
Pointer to destination array.
src
Pointer to source array.

Remarks: This function copies the string in ***src*** to the end of the string in ***dest***. The ***src*** string starts at the null in ***dest***. A null character is added to the end of the resulting string in ***dest***. If ***src*** and ***dest*** overlap, the behavior is undefined.

Return Value: This function returns the value of ***dest***.

File Names: `strcat.asm`
`scatp2r.asm`

strchr

Function: Locate the first occurrence of a specified character in a string.

Include: `string.h`

Prototype:
```
char * strchr( const char * str,
               const char c );
```

Arguments:
str
Pointer to a string to be searched.
c
Character to find.

Remarks: This function searches the string ***str*** to find the first occurrence of character ***c***.
This function differs from the ANSI specified function in that ***c*** is defined as an `unsigned char` parameter rather than an `int` parameter.

Return Value: If *c* appears in ***str***, this function returns a pointer to the character in ***str***. Otherwise, it returns a null pointer.

File Names: `strchr.asm`

strcmp
strcmppgm2ram

Function: Compare two strings.

Include: string.h

Prototype:
```
signed char strcmp(
      const char * str1,
      const char * str2 );
signed char strcmppgm2ram(
      const char * str1,
      const rom char * str2 );
```

Arguments:
str1
Pointer to first string.
str2
Pointer to second string.

Remarks: This function compares the string in ***str1*** to the string in ***str2*** and returns a value indicating if ***str1*** is less than, equal to or greater than ***str2***.

Return Value: strcmp returns a value that is:
<0 if ***str1*** is less than ***str2***
==0 if ***str1*** is the same as ***str2***
>0 if ***str1*** is greater than ***str2***

File Name: strcmp.asm
scmpp2r.asm

strcpy
strcpypgm2ram

Function: Copy the source string into the destination string.

Include: string.h

Prototype:
```
char * strcpy( char * dest,
               const char * src );
char * strcpypgm2ram(
               char * dest,
               const rom char *src );
```

Arguments:
dest
Pointer to destination string.
src
Pointer to source string.

Remarks: This function copies the string in ***src*** to ***dest***. Characters in ***src*** are copied up to, and including, the terminating null character in ***src***. If ***src*** and ***dest*** overlap, the behavior is undefined.

Return Value: This function returns the value of ***dest***.

File Name: strcpy.asm
scpyp2r.asm

strcspn

Function: 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 * str1,
                  const char * str2 );
```

Arguments:
str1
Pointer to a string to be searched.
str2
Pointer to a string that is treated as a set of characters.

Remarks: This function will determine the number of consecutive characters from the beginning of ***str1*** that are not contained in ***str2***. For example:

str1	***str2***	**result**
"hello"	"aeiou"	1
"antelope"	"aeiou"	0
"antelope"	"xyz"	8

Return Value: This function returns the number of consecutive characters from the beginning of ***str1*** that are not contained in ***str2***, as shown in the examples above.

File Names: `strcspn.asm`

strlen

Function: Return the length of the string.

Include: `string.h`

Prototype:
```
size_t strlen( const char * str );
```

Arguments:
str
Pointer to string.

Remarks: This function determines the length of the string, not including the terminating null character.

Return Value: This function returns the length of the string.

File Name: `strlen.asm`

strlwr

Function: Convert all upper case characters in a string to lower case.

Include: `string.h`

Prototype:
```
char * strlwr( char * str );
```

Arguments:
str
Pointer to string.

Remarks: This function converts all upper case characters in ***str*** to lower case characters. All characters that are not upper case (A to Z) are not affected.

Return Value: This function returns the value of ***str***.

File Name: `strlwr.asm`

strncat
strncatpgm2ram

Function: Append a specified number of characters from the source string to the destination string.

Include: string.h

Prototype:
```
char * strncat( char * dest,
                const char * src,
                size_t n );
char * strncatpgm2ram(
                char * dest,
                const rom char * src,
                sizeram_t n );
```

Arguments:
dest
Pointer to destination array.
src
Pointer to source array.
n
Number of characters to append.

Remarks: This function appends exactly ***n*** characters from the string in ***src*** to the end of the string in ***dest***. If a null character is copied before ***n*** characters have been copied, null characters will be appended to ***dest*** until exactly ***n*** characters have been appended.
If ***src*** and ***dest*** overlap, the behavior is undefined.
If a null character is not encountered, then a null character is not appended.

Return Value: This function returns the value of ***dest***.

File Names: strncat.asm
sncatp2r.asm

strncmp

Function: Compare two strings, up to a specified number of characters.

Include: string.h

Prototype:
```
signed char strncmp( const char * str1,
                     const char * str2,
                     size_t n );
```

Arguments:
str1
Pointer to first string.
str2
Pointer to second string.
n
Maximum number of characters to compare.

Remarks: This function compares the string in ***str1*** to the string in ***str2*** and returns a value indicating if ***str1*** is less than, equal to or greater than ***str2***. If ***n*** characters are compared and no differences are found, this function will return a value indicating that the strings are equivalent.

strncmp (Continued)

Return Value: `strncmp` returns a value based on the first character that differs between ***str1*** and ***str2***. It returns:
<0 if ***str1*** is less than ***str2***
==0 if ***str1*** is the same as ***str2***
>0 if ***str1*** is greater than ***str2***

File Name: `strncmp.asm`

strncpy
strncpypgm2ram

Function: Copy characters from the source string into the destination string, up to the specified number of characters.

Include: `string.h`

Prototype:
```
char * strncpy( char * dest,
                const char * src,
                size_t n );
char *strncpypgm2ram(
                char * dest,
                const rom char * src,
                sizeram_t n );
```

Arguments:
dest
Pointer to destination string.
src
Pointer to source string.
n
Maximum number of characters to copy.

Remarks: This function copies the string in ***src*** to ***dest***. Characters in ***src*** are copied into ***dest*** until the terminating null character or ***n*** characters have been copied. If ***n*** characters were copied and no null character was found then ***dest*** will not be null-terminated.
If copying takes place between objects that overlap, the behavior is undefined.

Return Value: This function returns the value of ***dest***.

File Name: `strncpy.asm`
`sncpyp2r.asm`

strpbrk

Function: Search a string for the first occurrence of a character from a specified set of characters.

Include: `string.h`

Prototype:
```
char * strpbrk( const char * str1,
                const char * str2 );
```

Arguments:
str1
Pointer to a string to be searched.
str2
Pointer to a string that is treated as a set of characters.

Remarks: This function will search ***str1*** for the first occurrence of a character contained in ***str2***.

Return Value: If a character in ***str2*** is found, a pointer to that character in ***str1*** is returned. If no character from ***str2*** is found in ***str1***, a null pointer is returned.

File Names: `strpbrk.asm`

strrchr

Function: Locate the last occurrence of a specified character in a string.

Include: `string.h`

Prototype:
```
char * strrchr( const char * str,
                const char c );
```

Arguments:
str
Pointer to a string to be searched.
c
Character to find.

Remarks: This function searches the string ***str***, including the terminating null character, to find the last occurrence of character ***c***.
This function differs from the ANSI specified function in that ***c*** is defined as an `unsigned char` parameter rather than an `int` parameter.

Return Value: If *c* appears in ***str***, this function returns a pointer to the character in ***str***. Otherwise, it returns a null pointer.

File Names: `strrchr.asm`

strspn

Function: 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 * str1,
                 const char * str2 );
```

Arguments:
str1
Pointer to a string to be searched.
str2
Pointer to a string that is treated as a set of characters.

strspn (Continued)

Remarks: This function will determine the number of consecutive characters from the beginning of ***str1*** that are contained in ***str2***. For example:

str1	***str2***	**result**
"banana"	"ab"	2
"banana"	"abn"	6
"banana"	"an"	0

Return Value: This function returns the number of consecutive characters from the beginning of ***str1*** that are contained in ***str2***, as shown in the examples above.

File Names: `strspn.asm`

strstr

Function: Locate the first occurrence of a string inside another string.

Include: `string.h`

Prototype:
```
char * strstr( const char * str,
               const char * substr );
```

Arguments:

str
Pointer to a string to be searched.

substr
Pointer to a string pattern for which to search.

Remarks: This function will find the first occurrence of the string ***substr*** (excluding the null terminator) within string ***str***.

Return Value: If the string is located, a pointer to that string in ***str*** will be returned. Otherwise a null pointer is returned.

File Names: `strstr.asm`

strtok

Function: Break a string into substrings, or tokens, by inserting null characters in place of specified delimiters.

Include: `string.h`

Prototype:
```
char * strtok( char * str,
               const char * delim );
```

Arguments:

str
Pointer to a string to be searched.

delim
Pointer to a set of characters that indicate the end of a token.

strtok (Continued)

Remarks: This function can be used to split up a string into substrings by replacing specified characters with null characters. The first time this function is invoked on a particular string, that string should be passed in ***str***. After the first time, this function can continue parsing the string from the last delimiter by invoking it with a null value passed in ***str***.
When `strtok` is invoked with a non-null parameter for ***str***, it starts searching ***str*** from the beginning. It skips all leading characters that appear in the string ***delim***, then skips all characters not appearing in ***delim***, then sets the next character to null.
When `strtok` is invoked with a null parameter for ***str***, it searches the string that was most recently examined, beginning with the character after the one that was set to null during the previous call. It skips all characters not appearing in ***delim***, then sets the next character to null.
If `strtok` finds the end of the string before it finds a delimiter, it does not modify the string.
The set of characters that is passed in ***delim*** need not be the same for each call to `strtok`.

Return Value: If a delimiter was found, this function returns a pointer into ***str*** to the first character that was searched that did not appear in the set of characters ***delim***. This character represents the first character of a token that was created by the call.
If no delimiter was found prior to the terminating null character, a null pointer is returned from the function.

File Names: `strtok.asm`

strupr

Function: Convert all lower case characters in a string to upper case.

Include: `string.h`

Prototype:
```
char * strupr( char * str );
```

Arguments: ***str***
Pointer to string.

Remarks: This function converts all lower case characters in ***str*** to upper case characters. All characters that are not lower case (a to z) are not affected.

Return Value: This function returns the value of ***str***.

File Name: `strupr.asm`

4.5 DELAY FUNCTIONS

The delay functions execute code for a specific number of processor instruction cycles. For time-based delays, the processor operating frequency must be taken into account. The following routines are provided:

Function	Description
Delay1TCY	Delay one instruction cycle.
Delay10TCYx	Delay in multiples of 10 instruction cycles.
Delay100TCYx	Delay in multiples of 100 instruction cycles.
Delay1KTCYx	Delay in multiples of 1,000 instruction cycles.
Delay10KTCYx	Delay in multiples of 10,000 instruction cycles.

4.5.1 Function Descriptions

Delay1TCY

Function: Delay 1 instruction cycle (Tcy).

Include: `delays.h`

Prototype: `void Delay1TCY( void );`

Remarks: This function is actually a `#define` for the `NOP()` instruction. When encountered in the source code, the compiler simply inserts a `NOP()`.

File Name: `#define` in `delays.h`

Delay10TCYx

Function: Delay in multiples of 10 instruction cycles (Tcy).

Include: `delays.h`

Prototype: `void Delay10TCYx( unsigned char unit );`

Arguments: ***unit***
The value of ***unit*** can be any 8-bit value. A value in the range [1,255] will delay (***unit*** * 10) cycles. A value of 0 causes a delay of 2,560 cycles.

Remarks: This function creates a delay in multiples of 10 instruction cycles.

File Name: `d10tcyx.asm`

Delay100TCYx

Function: Delay in multiples of 100 instruction cycles (Tcy).

Include: `delays.h`

Prototype: `void Delay100TCYx( unsigned char unit );`

Arguments: ***unit***
The value of ***unit*** can be any 8-bit value. A value in the range [1,255] will delay (***unit*** * 100) cycles. A value of 0 causes a delay of 25,600 cycles.

Remarks: This function creates a delay in multiples of 100 instruction cycles.

File Name: `d100tcyx.asm`

Delay1KTCYx

Function: Delay in multiples of 1,000 instruction cycles (Tcy).

Include: `delays.h`

Prototype: `void Delay1KTCYx( unsigned char unit );`

Arguments: ***unit***
The value of ***unit*** can be any 8-bit value. A value in the range [1,255] will delay (***unit*** * 1000) cycles. A value of 0 causes a delay of 256,000 cycles.

Remarks: This function creates a delay in multiples of 1,000 instruction cycles.

File Name: `d1ktcyx.asm`

Delay10KTCYx

Function: Delay in multiples of 10,000 instruction cycles (Tcy).

Include: `delays.h`

Prototype: `void Delay10KTCYx( unsigned char unit );`

Arguments: ***unit***
The value of ***unit*** can be any 8-bit value. A value in the range [1,255] will delay (***unit*** * 10000) cycles. A value of 0 causes a delay of 2,560,000 cycles.

Remarks: This function creates a delay in multiples of 10,000 instruction cycles.

File Name: `d10ktcyx.asm`

4.6 RESET FUNCTIONS

The RESET functions may be used to help determine the source of a RESET or wake-up event and for reconfiguring the processor status following a RESET. The following routines are provided:

Function	Description
isBOR	Determine if the cause of a RESET was the Brown-Out Reset circuit.
isLVD	Determine if the cause of a RESET was a low voltage detect condition.
isMCLR	Determine if the cause of a RESET was the $\overline{\text{MCLR}}$ pin.
isPOR	Detect a Power-on RESET condition.
isWDTTO	Determine if the cause of a RESET was a watchdog timer time out.
isWDTWU	Determine if the cause of a wake-up was the watchdog timer.
isWU	Detects if the microcontroller was just waken up from SLEEP from the $\overline{\text{MCLR}}$ pin or an interrupt.
StatusReset	Set the $\overline{\text{POR}}$ and $\overline{\text{BOR}}$ bits.

Note: If you are using Brown-out Reset (BOR) or the Watchdog Timer (WDT), you must define the enable macros (`#define BOR_ENABLED` and `#define WDT_ENABLED`, respectively) in the header file `reset.h` and recompile the source code.

4.6.1 Function Descriptions

isBOR

Function: Determine if the cause of a RESET was the Brown-out Reset circuit.

Include: `reset.h`

Prototype:
```
char isBOR( void );
```

Remarks: This function detects if the microcontroller was reset due to the Brown-out Reset circuit. This condition is indicated by the following status bits:
$\overline{\text{POR}}$ = 1
$\overline{\text{BOR}}$ = 0

Return Value: 1 if the RESET was due to the Brown-out Reset circuit
0 otherwise

File Name: `isbor.c`

isLVD

Function:	Determine if the cause of a RESET was a low voltage detect condition.
Include:	`reset.h`
Prototype:	`char isLVD( void );`
Remarks:	This function detects if the voltage of the device has become lower than the value specified in the LVDCON register (LVDL3:LVDL0 bits.)
Return Value:	1 if a RESET was due to LVD during normal operation 0 otherwise
File Name:	`islvd.c`

isMCLR

Function:	Determine if the cause of a RESET was the $\overline{\text{MCLR}}$ pin.
Include:	`reset.h`
Prototype:	`char isMCLR( void );`
Remarks:	This function detects if the microcontroller was reset via the $\overline{\text{MCLR}}$ pin while in normal operation. This situation is indicated by the following status bits: $\overline{\text{POR}}$ = 1 If Brown-out is enabled, $\overline{\text{BOR}}$ = 1 If WDT is enabled, $\overline{\text{TO}}$ = 1 $\overline{\text{PD}}$ = 1
Return Value:	1 if the RESET was due to $\overline{\text{MCLR}}$ during normal operation 0 otherwise
File Name:	`ismclr.c`

isPOR

Function:	Detect a Power-on Reset condition.
Include:	`reset.h`
Prototype:	`char isPOR( void );`
Remarks:	This function detects if the microcontroller just left a Power-on Reset. This condition is indicated by the following status bits: $\overline{\text{POR}}$ = 0 $\overline{\text{BOR}}$ = 0 $\overline{\text{TO}}$ = 1 $\overline{\text{PD}}$ = 1 This condition also can occur for $\overline{\text{MCLR}}$ during normal operation and when the `CLRWDT` instruction is executed. After `isPOR` is called, `StatusReset` should be called to set the $\overline{\text{POR}}$ and $\overline{\text{BOR}}$ bits.
Return Value:	1 if the device just left a Power-on Reset 0 otherwise
File Name:	`ispor.c`

isWDTTO

Function: Determine if the cause of a RESET was a watchdog timer (WDT) time out.

Include: `reset.h`

Prototype: `char isWDTTO( void );`

Remarks: This function detects if the microcontroller was reset due to the WDT during normal operation. This condition is indicated by the following status bits:
$\overline{POR}$ = 1
$\overline{BOR}$ = 1
$\overline{TO}$ = 0
$\overline{PD}$ = 1

Return Value: 1 if the RESET was due to the WDT during normal operation
0 otherwise

File Name: `iswdtto.c`

isWDTWU

Function: Determine if the cause of a wake-up was the watchdog timer (WDT).

Include: `reset.h`

Prototype: `char isWDTWU( void );`

Remarks: This function detects if the microcontroller was brought out of SLEEP by the WDT. This condition is indicated by the following status bits:
$\overline{POR}$ = 1
$\overline{BOR}$ = 1
$\overline{TO}$ = 0
$\overline{PD}$ = 0

Return Value: 1 if device was brought out of SLEEP by the WDT
0 otherwise

File Name: `iswdtwu.c`

isWU

Function: Detects if the microcontroller was just waken up from SLEEP via the $\overline{MCLR}$ pin or interrupt.

Include: `reset.h`

Prototype: `char isWU( void );`

Remarks: This function detects if the microcontroller was brought out of SLEEP by the $\overline{MCLR}$ pin or an interrupt. This condition is indicated by the following status bits:
$\overline{POR}$ = 1
$\overline{BOR}$ = 1
$\overline{TO}$ = 1
$\overline{PD}$ = 0

Return Value: 1 if the device was brought out of SLEEP by the $\overline{MCLR}$ pin or an interrupt
0 otherwise

File Name: `iswu.c`

StatusReset

Function:	Set the $\overline{\text{POR}}$ and $\overline{\text{BOR}}$ bits in the CPUSTA register.
Include:	reset.h
Prototype:	void StatusReset(void);
Remarks:	This function sets the $\overline{\text{POR}}$ and $\overline{\text{BOR}}$ bits in the CPUSTA register. These bits must be set in software after a Power-on Reset has occurred.
File Name:	statrst.c

Chapter 5. Math Libraries

5.1 INTRODUCTION

This chapter documents math library functions. For more information on math libraries, see the *Embedded Control Handbook, Volume 2* (DS00167). See the *MPASM User's Guide with MPLINK and MPLIB* for more information on creating and using libraries in general.

This chapter includes the following sections:

- 32-Bit Integer and 32-Bit Floating Point Math Libraries
- Decimal/Floating Point and Floating Point/Decimal Conversions

5.2 32-BIT INTEGER AND 32-BIT FLOATING POINT MATH LIBRARIES

The math routines used by MPLAB C18 are based on the Microchip Application Note AN575. Source code for the routines may be found in the `src\math` subdirectory of the compiler installation. These source files have been compiled into object code and added to the `clib.lib` standard library, which may be found in the `lib` subdirectory. The `clib.lib` file is included when using the linker script files provided with MPLAB C18.

The mathematical functions performed by the floating point library routines are: 32-bit signed integer multiplication and division, 32-bit unsigned integer multiplication and division and 32-bit floating-point multiplication and division. The routines also contain functions that convert from 8-, 16-, 24- and 32-bit signed and unsigned integers to 32-bit floating point, as well as a 32-bit floating point conversion to 32-bit integer.

5.2.1 Floating Point Representation

Floating point numbers are represented in a modified IEEE-754 format. This format allows the floating-point routines to take advantage of the processor architecture and reduce the amount of overhead required in the calculations. The representation is shown below compared to the IEEE-754 format:

Format	Exponent	Mantissa 0	Mantissa 1	Mantissa 2
IEEE-754	`sxxx xxxx`	`yxxx xxxx`	`xxxx xxxx`	`xxxx xxxx`
Microchip	`xxxx xxxy`	`sxxx xxxx`	`xxxx xxxx`	`xxxx xxxx`

where `s` is the sign bit, `y` is the LSb of the exponent and `x` is a placeholder for the mantissa and exponent bits.

The two formats may be easily converted from one to the other by manipulation of the Exponent and Mantissa 0 bytes. The following assembly code shows an example of this operation.

EXAMPLE 5-1: IEEE-754 TO MICROCHIP

```
Rlcf MANTISSA0
Rlcf EXPONENT
Rrcf MANTISSA0
```

EXAMPLE 5-2: MICROCHIP TO IEEE-754

```
Rlcf MANTISSA0
Rrcf EXPONENT
Rrcf MANTISSA0
```

5.2.2 Variables Used by the Floating Point Libraries

Several 8-bit RAM registers are used by the math routines to hold the operands for and results of floating point and integer operations. Since there may be two operands required for a floating point operation (such as multiplication or division), there are two sets of exponent and mantissa registers reserved (A and B). For argument A, AEXP holds the exponent and AARGB0, AARGB1 and AARGB2 hold the mantissa. For argument B, BEXP holds the exponent and BARGB0, BARGB1 and BARGB2 hold the mantissa.

Note: The MSB of the mantissa is stored in the AARGB0 or BARGB0 byte. Results of the floating point routines are placed in the AEXP and AARGB0:2 registers.

For 32-bit integers, AARGB0, AARGB1, AARGB2 and AARGB3 or BARGB0, BARGB1, BARGB2 and BARGB3 are used to hold the operands. Results of integer operations will be placed in AARGB0, AARGB1, AARGB2 and AARGB3. In the case of 32-bit division, the remainder is placed in an additional set of registers, REMB0, REMB1, REMB2 and REMB3. The MSB of the 32-bit integer is contained in AARGB0, BARGB0 or REMB0.

5.3 DECIMAL/FLOATING POINT AND FLOATING POINT/DECIMAL CONVERSIONS

The details of how decimal numbers are converted to floating point numbers and how floating point numbers are converted to decimal numbers are discussed in the following sections.

5.3.1 Converting Decimal to Microchip Floating Point

There are several methods that will allow the conversion of decimal (base 10) numbers to Microchip floating point format. Microchip provides a PC utility called `FPREP.EXE`, which will convert decimal numbers to floating point for use in the math library routines. This utility may be downloaded from the Microchip web site along with the AN575 source code.

Alternatively, the floating point equivalent to decimal numbers may be calculated longhand. To calculate the floating point via a longhand method, both the exponent and mantissa must be found.

To find the exponent, the following formulae are used:

EQUATION 5-1:

$$2^Z = A_{10}$$

EQUATION 5-2:

$$Exp = int(Z)$$

where Z is the fractional exponent, A_{10} is the original decimal number, and Exp is the integer portion of Z.

To solve for the exponent, first begin by rearranging Equation 5-1 to solve for Z.

$$Z = \frac{\ln(A_{10})}{\ln(2)}$$

If Z is positive, then it is rounded to the next larger integer value. If Z is negative, then it is rounded to the next smaller integer value. The resulting value is Exp.

Finally, a bias value of `0x7F` is added to convert Exp to Microchip floating point format (Exp_{MFP}).

$$Exp_{MFP} = Exp + 0x7F$$

To find the mantissa, the exponent value just determined must be removed from the original decimal number, using division.

EQUATION 5-3:

$$x = \frac{A_{10}}{2^Z}$$

where x is the fractional portion of the mantissa, and A_{10} and Z are values as described above.

Note: x will always be a value greater than 1.

To determine the binary representation of the mantissa, x is compared in turn to decreasing powers of 2, starting with 2^0 and decreasing to 2^{-23}. If x is greater than or equal to the power of 2 currently being compared, a '1' is placed in the corresponding bit position of the binary representation and the power of 2 value is subtracted from x. The new x is then used for the next decreasing power of 2 comparison. If x is less than the power of 2 currently being compared, a '0' is placed in the bit position and no subtraction occurs. The same value of x is used to compare to the next power of 2 value.

This process repeats until all 24 bits have been determined or until subtraction yields an x value of 0. Finally, to convert this 24-bit value to Microchip floating point format, the MSb is substituted with the sign of the original decimal number, i.e., '1' for negative or '0' for positive.

To demonstrate the method of conversion, the same example as in AN575 will be used, where A_{10} = 0.15625.

First, find the exponent:

$$2^Z = 0.15625$$

$$Z = \frac{\ln(0.15625)}{\ln(2)} = -2.6780719$$

$$Exp = int(Z) = -3$$

Next calculate the fractional portion of the mantissa:

$$x = \frac{0.15625}{2^{-3}} = 1.25$$

And then the binary representation:

$x = 1.25 \geq 2^0$?	Yes	bit = 1; x = 1.25 - 1 = 0.25
$x = 0.25 \geq 2^{-1}$?	No	bit = 0; x = 0.25
$x = 0.25 \geq 2^{-2}$?	Yes	bit = 1; x = 0.25 - 0.25 = 0
$x = 0$	Process complete	

Therefore, the binary representation is:

A_2=1.01000000000000000000000.

Finally, convert to Microchip floating point format by placing the proper sign bit in the MSb of the mantissa and add `0x7F` to the calculated exponent. The Microchip floating point representation of 0.156256 is then `0x7C200000`. For more details on the floating point conversion, please consult AN575.

5.3.2 Converting Microchip Floating-Point to Decimal

The process of converting floating-point number to decimal is relatively simple and can be done by hand (or using a calculator) to check your results. To convert from floating point to decimal, the following formula is used:

EQUATION 5-4:

$$A_{10} = 2^{Exp} \cdot A_2$$

where Exp is the unbiased exponent and A is the binary expansion of the mantissa.

Some processing of the values stored in AEXP and AARGB0:2 must be performed in order to use the above formula. The exponent is stored in a biased format, which simply means that `0x7F` has been added to the true exponent that of the number. To extract the exponent to be used in the above calculation, subtract `0x7F` from the value stored in AEXP.

The sign bit is stored in the MSB of the mantissa. To allow the full 24-bit precision of the mantissa, the MSB is assumed to be 1 explicitly, once the sign bit is stripped out. To calculate A_2, a simple binary expansion is used, as shown in the formula below. Since the MSB is explicitly 1, the expansion will always contain the term 2^0.

EQUATION 5-5:

$$A_2 = 2^0 + (Bit22) \bullet 2^{-1} + (Bit21) \bullet 2^{-2} + ... + (Bit0) \bullet 2^{-23}$$

As in AN575, we will use the example of the decimal number 50.2654824574. which has a floating point representation of `0x84490FDB`, with the biased exponent being `0x84` and the mantissa (including sign bit) being `0x490FDB`. The unbiased exponent is calculated to be Exp = `0x84` - `0x7F` = `0x05`. To process the mantissa, it is first translated to binary format and the MSB is set to prepare for the expansion.

0x490FDB =

$0100\ 1001\ 0000\ 1111\ 1101\ 1011_2 \rightarrow$

$1100\ 1001\ 0000\ 1111\ 1101\ 1011_2$

The expansion is then performed according to Equation 5-5.

$$A_2 = 2^0 + 2^{-1} + 2^{-4} + 2^{-7} + 2^{-12} + 2^{-13} + 2^{-14} + 2^{-15} + 2^{-16} + 2^{-17} + 2^{-19} + 2^{-20} + 2^{-22} + 2^{-23}$$

$$A_2 = 1.570796371$$

Finally, to calculate the actual floating point number, the exponent and expanded mantissa are plugged into the conversion formula (Equation 5-4).

$$A_{10} = 2^5 \bullet 1.570796371$$

$$A_{10} = 50.26548387$$

The result of these calculations are accurate out to about 5 decimal places, with rounding and calculation errors creating some degree of uncertainty for the remaining decimal places. For more details on the sources of error, please consult AN575.

NOTES:

Glossary

A

absolute section

A section with a fixed address that cannot be changed by the linker.

access memory

Special general purpose registers on the PIC18 PICmicro microcontrollers that allow access regardless of the setting of the bank select register (`BSR`).

address

The code that identifies where a piece of information is stored in memory.

anonymous structure

An unnamed object.

ANSI

American National Standards Institute

assembler

A language tool that translates assembly source code into machine code.

assembly

A symbolic language that describes the binary machine code in a readable form.

assigned section

A section that has been assigned to a target memory block in the linker command file.

asynchronously

Multiple events that do not occur at the same time. This is generally used to refer to interrupts that may occur at any time during processor execution.

B

binary

The base two numbering system that uses the digits 0-1. The right-most digit counts ones, the next counts multiples of 2, then $2^2 = 4$, etc.

C

central processing unit

The part of a device that is responsible for fetching the correct instruction for execution, decoding that instruction, and then executing that instruction. When necessary, it works in conjunction with the arithmetic logic unit (ALU) to complete the execution of the instruction. It controls the program memory address bus, the data memory address bus, and accesses to the stack.

compiler

A program that translates a source file written in a high-level language into machine code.

conditional compilation

The act of compiling a program fragment only if a certain constant expression, specified by a preprocessor directive, is true.

CPU

Central Processing Unit

E

endianness

The ordering of bytes in a multi-byte object.

error file

A file containing the diagnostics generated by the MPLAB C18

F

fatal error

An error that will halt compilation immediately. No further messages will be produced.

frame pointer

A pointer that references the location on the stack that separates the stack-based arguments from the stack-based local variables.

free-standing

An implementation that accepts any strictly conforming program that does not use complex types and in which the use of the features specified in the library clause (ANSI '89 standard clause 7) is confined to the contents of the standard headers `<float.h>`, `<iso646.h>`, `<limits.h>`, `<stdarg.h>`, `<stdbool.h>`, `<stddef.h>`, and `<stdint.h>`.

H

hexadecimal

The base 16 numbering system that uses the digits 0-9 plus the letters A-F (or a-f). The digits A-F represent decimal values of 10 to 15. The right-most digit counts ones, the next counts multiples of 16, then $16^2 = 256$, etc.

high-level language

A language for writing programs that is further removed from the processor than assembly.

I

ICD

In-Circuit Debugger

ICE

In-Circuit Emulator

IDE

Integrated Development Environment

IEEE

Institute of Electrical and Electronics Engineers

interrupt

A signal to the CPU that suspends the execution of a running application and transfers control to an ISR so that the event may be processed. Upon completion of the ISR, normal execution of the application resumes.

interrupt service routine

A function that handles an interrupt.

ISO

International Organization for Standardization

ISR

Interrupt Service Routine

L

latency

The time between when an event occurs and the response to it.

librarian

A program that creates and manipulates libraries.

library

A collection of relocatable object modules.

linker

A program that combines object files and libraries to create executable code.

little endian

Within a given object, the Least Significant byte is stored at lower addresses.

M

memory model

A description that specifies the size of pointers that point to program memory.

microcontroller

A highly integrated chip that contains a CPU, RAM, some form of ROM, I/O ports, and timers.

MPASM assembler

Microchip Technology's relocatable macro assembler for PICmicro microcontroller families.

MPLIB object librarian

Microchip Technology's librarian for PICmicro microcontroller families.

MPLINK object linker

Microchip Technology's linker for PICmicro microcontroller families.

O

object file

A file containing object code. It may be immediately executable or it may require linking with other object code files, e.g. libraries, to produce a complete executable program.

object code

The machine code generated by an assembler or compiler.

octal

The base 8 number system that only uses the digits 0-7. The right-most digit counts ones, the next digit counts multiples of 8, then 8^2 = 64, etc.

P

pragma

A directive that has meaning to a specific compiler.

R

RAM

Random Access Memory

random access memory

A memory device in which information can be accessed in any order.

read only memory

Memory hardware that allows fast access to permanently stored data but prevents addition to or modification of the data.

ROM

Read Only Memory

recursive

Self-referential (e.g., a function that calls itself). *See recursive*.

reentrant

A function that may have multiple, simultaneously active instances. This may happen due to either direct or indirect recursion or through execution during interrupt processing.

relocatable

An object whose address has not been assigned to a fixed memory location.

runtime model

Set of assumptions under which the compiler operates.

S

section

A portion of an application located at a specific address of memory.

section attribute

A characteristic ascribed to a section (e.g., an `access` section).

special function register

Registers that control I/O processor functions, I/O status, timers or other modes or peripherals.

storage class

Determines the lifetime of the memory associated with the identified object.

storage qualifier

Indicates special properties of the objects being declared (e.g., `const`).

V

vector

The memory locations that an application will jump to when either a RESET or interrupt occurs.

Index

Index

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: 480-792-7627
Web Address: http://www.microchip.com

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

Atlanta
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

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

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

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

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

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

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

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

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

ASIA/PACIFIC

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

China - Beijing
Microchip Technology Consulting (Shanghai) Co., Ltd., Beijing Liaison Office
Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Chengdu
Microchip Technology Consulting (Shanghai) Co., Ltd., Chengdu Liaison Office
Rm. 2401, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-86766200 Fax: 86-28-86766599

China - Fuzhou
Microchip Technology Consulting (Shanghai) Co., Ltd., Fuzhou Liaison Office
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506 Fax: 86-591-7503521

China - Shanghai
Microchip Technology Consulting (Shanghai) Co., Ltd.
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

China - Shenzhen
Microchip Technology Consulting (Shanghai) Co., Ltd., Shenzhen Liaison Office
Rm. 1315, 13/F, Shenzhen Kerry Centre,
Renminnan Lu
Shenzhen 518001, China
Tel: 86-755-82350361 Fax: 86-755-82366086

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

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

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

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

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-6334-8870 Fax: 65-6334-8850

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

EUROPE

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

Denmark
Microchip Technology Nordic ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

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

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

Italy
Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom
Microchip Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

10/18/02