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

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

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

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contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates.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

Trademarks

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

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

Analog-for-the-Digital Age, Application Maestro, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, FuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, Pickit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, rFLAB, rFICDEM, Select Mode, Smart Serial, SmartTel and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

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

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

© 2004, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

Printed on recycled paper.
Table of Contents

Preface ............................................................................................................................ 1

Chapter 1. Overview

1.1 Introduction ............................................................................................................. 7
1.2 MPLAB C18 Libraries Overview ......................................................................... 7
1.3 Start-up Code .......................................................................................................... 7
1.4 Processor-independent Library ............................................................................. 8
1.5 Processor-specific Libraries ................................................................................. 9

Chapter 2. Hardware Peripheral Functions

2.1 Introduction ............................................................................................................. 11
2.2 A/D Converter Functions ...................................................................................... 11
2.3 Input Capture Functions ....................................................................................... 19
2.4 I²C™ Functions .................................................................................................... 23
2.5 I/O Port Functions .................................................................................................. 32
2.6 Microwire® Functions .......................................................................................... 34
2.7 Pulse-Width Modulation Functions ...................................................................... 39
2.8 SPI™ Functions ..................................................................................................... 42
2.9 Timer Functions ..................................................................................................... 48
2.10 USART Functions ............................................................................................... 56

Chapter 3. Software Peripheral Library

3.1 Introduction ............................................................................................................. 65
3.2 External LCD Functions ....................................................................................... 65
3.3 External CAN2510 Functions ............................................................................... 72
3.4 Software I²C Functions ....................................................................................... 94
3.5 Software SPI® Functions .................................................................................... 100
3.6 Software UART Functions .................................................................................. 103

Chapter 4. General Software Library

4.1 Introduction ............................................................................................................. 107
4.2 Character Classification Functions ......................................................................... 107
4.3 Data Conversion Functions ................................................................................... 112
4.4 Memory and String Manipulation Functions ........................................................... 117
4.5 Delay Functions .................................................................................................... 129
4.6 Reset Functions .................................................................................................... 131

Chapter 5. Math Libraries

5.1 Introduction ............................................................................................................. 135
5.2 32-bit Integer and 32-bit Floating Point Math Libraries ........................................... 135
5.3 Decimal/Floating Point and Floating Point/Decimal Conversions ....................... 136
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.

ABOUT THIS GUIDE

Document Layout

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.
Conventions Used in this Guide

This guide uses the following documentation conventions:

**DOCUMENTATION CONVENTION**

<table>
<thead>
<tr>
<th>Description</th>
<th>Represents</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code (Courier font):</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Courier font</td>
<td>Sample source code</td>
<td><code>distance -= time * speed;</code></td>
</tr>
<tr>
<td>Filenames and paths</td>
<td><code>\mcc18\h</code></td>
<td></td>
</tr>
<tr>
<td>Keywords</td>
<td><code>_asm, _endasm, static</code></td>
<td></td>
</tr>
<tr>
<td>Command-line options</td>
<td><code>-Opa+, -Opa-</code></td>
<td></td>
</tr>
<tr>
<td>Italic Courier font</td>
<td>Variable name argument</td>
<td><code>file.o, where file can be any valid file name</code></td>
</tr>
<tr>
<td>Square brackets [ ]</td>
<td>Optional arguments</td>
<td><code>mcc18 [options] file [options]</code></td>
</tr>
<tr>
<td>Ellipses...</td>
<td>Replaces repeated instances of text</td>
<td><code>var_name [, var_name...]</code></td>
</tr>
<tr>
<td></td>
<td>Represents code supplied by user.</td>
<td><code>void main (void) { ... }</code></td>
</tr>
<tr>
<td>A hexadecimal number</td>
<td>A hexadecimal number where <code>n</code> is a hexadecimal digit</td>
<td><code>0xFFFF, 0x007A</code></td>
</tr>
<tr>
<td>Documents (Arial font):</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Italic characters</td>
<td>Referenced books</td>
<td><em>MPLAB User's Guide</em></td>
</tr>
</tbody>
</table>

**Documentation Updates**

All documentation becomes dated, and this guide is no exception. Since MPLAB C18 is constantly evolving to meet customer needs, some 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 DSXXXXXXA, where:

- `XXXXX` = The document number.
- `A` = The revision level of the document.
RECOMMENDED READING

For more information on included libraries and precompiled object files for the compilers, 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 software.

MPLAB C18 C Compiler Getting Started Guide (DS51295)
Describes how to install the MPLAB C18 compiler, how to write simple programs and how to use the MPLAB IDE with the compiler.

MPLAB C18 C Compiler User’s Guide (DS51288)
Comprehensive guide that describes the operation and features of Microchip’s MPLAB C18 C compiler for PIC18 devices.

MPLAB IDE V6.XX Quick Start Guide (DS51281)
Describes how to set up the MPLAB IDE software and use it to create projects and program devices.

MPASM™ User’s Guide with MPLINK™ Linker and MPLIB™ Librarian (DS33014)
Describes how to use the Microchip PICmicro MCU assembler (MPASM), linker (MPLINK) and librarian (MPLIB).

PICmicro 18C MCU Family Reference Manual (DS39500)
Focuses on the Enhanced MCU family of devices. The operation of the Enhanced MCU family architecture and peripheral modules is explained but does not cover the specifics of each device.

PIC18 Device Data Sheets and Application Notes
Data sheets describe the operation and electrical specifications of PIC18 devices. Application notes describe how to use PIC18 devices.

To obtain any of the above listed documents, visit the Microchip web site (www.microchip.com) to retrieve these documents in Adobe Acrobat (.pdf) format.
THE MICROCHIP WEB SITE

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

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

http://www.microchip.com

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

Technical Support

• Frequently Asked Questions (FAQ)
• On-line Discussion Groups – conferences for products, development systems, technical information and more
• Microchip Consultant Program Member Listing
• Links to other useful web sites related to Microchip products

Engineer's Toolbox

• Design Tips
• Device Errata

Other Available Information

• Latest Microchip Press Releases
• Listing of seminars and events
• Job Postings
DEVELOPMENT SYSTEMS CUSTOMER NOTIFICATION SERVICE

Microchip started the customer notification service to help our customers keep current on Microchip products with the least amount of effort. Once you subscribe, you will receive e-mail 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 at (http://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 object linkers; MPLIB™ and MPLAB LIB30 object librarians.

**Emulators** – The latest information on Microchip in-circuit emulators. This includes the MPLAB ICE 2000 and MPLAB ICE 4000.

**In-Circuit Debuggers** – The latest information on the Microchip in-circuit debugger, MPLAB ICD 2.

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

**Programmers** – The latest information on Microchip device programmers. These include the MPLAB PM3 and PRO MATE® II device programmers and PICSTART® Plus development programmer.
CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Corporate Applications Engineer (CAE)
- Hotline

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

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

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

The Hotline Numbers are:
1-800-755-2345 for U.S. and most of Canada.
1-480-792-7302 for the rest of the world.
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. The directory src\traditional contains the files for Non-extended mode and src\extended contains the files for Extended mode. 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, src\traditional and src\extended directories.

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. The c018*.o object files are for use with the compiler operating in the Non-extended mode. The c018*_e.o object files are for use with the compiler when operating in Extended mode. In increasing order of complexity, they are:

- c018.o/c018_e.o initializes the C software stack and jumps to the start of the application function, main().
- c018i.o/c018i_e.o performs all of the same tasks as c018.o/c018_e.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/c018iz_e.o performs all of the same tasks as c018i.o/c018i_e.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\traditional\startup` and `src\extended\startup` subdirectories of the compiler installation.

1.3.3 Rebuilding
Use the batch file `makestartup.bat` to rebuild the start-up code and copy the generated object files to the `lib` directory.
Before rebuilding the start-up code with `makestartup.bat`, verify that MPLAB C18 (`mcc18.exe`) is in your path.

1.4 PROCESSOR-INDEPENDENT LIBRARY

1.4.1 Overview
The standard C library (`clib.lib` or `clib_e.lib`) 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 the standard C library may be found in the following subdirectories of the compiler installation:
- `src\traditional\math`
- `src\extended\math`
- `src\traditional\delays`
- `src\extended\delays`
- `src\traditional\stdclib`
- `src\extended\stdclib`

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 the standard C library, 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:

\texttt{p\ processor.lib} - Non-extended mode processor-specific library
\texttt{p\ processor\_e.lib} - Extended mode processor-specific library

For example, the library file for the PIC18F4620 is named \texttt{p18f4620.lib} for the
Non-extended version of the library and \texttt{p18f4620\_e.lib} for the Extended version
of the library.

1.5.2 Source Code

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

• \texttt{src\traditional\pmc}
• \texttt{src\extended\pmc}
• \texttt{src\traditional\proc}
• \texttt{src\extended\proc}

1.5.3 Rebuilding

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

• MPLAB C18 (\texttt{mcc18.exe})
• MPASM assembler (\texttt{mpasm.exe})
• MPLIB librarian (\texttt{mplib.exe})

Also prior to invoking \texttt{makeplib.bat}, be sure that the environment variable
\texttt{MCC\_INCLUDE} is set to the path of the MPLAB C18 include files (e.g., \texttt{c:\mcc18\h}).
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\traditional\pmc and src\extended\pmc subdirectories 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 (Section 2.2 “A/D Converter Functions”)
- Input Capture (Section 2.3 “Input Capture Functions”)
- I²C™ (Section 2.4 “I²C™ Functions”)
- I/O Ports (Section 2.5 “I/O Port Functions”)
- Microwire® (Section 2.6 “Microwire® Functions”)
- Pulse-Width Modulation (PWM) (Section 2.7 “Pulse-Width Modulation Functions”)
- SPI™ (Section 2.8 “SPI™ Functions”)
- Timer (Section 2.9 “Timer Functions”)
- USART (Section 2.10 “USART Functions”)

2.2 A/D CONVERTER FUNCTIONS

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusyADC</td>
<td>Is A/D converter currently performing a conversion?</td>
</tr>
<tr>
<td>CloseADC</td>
<td>Disable the A/D converter.</td>
</tr>
<tr>
<td>ConvertADC</td>
<td>Start an A/D conversion.</td>
</tr>
<tr>
<td>OpenADC</td>
<td>Configure the A/D converter.</td>
</tr>
<tr>
<td>ReadADC</td>
<td>Read the results of an A/D conversion.</td>
</tr>
<tr>
<td>SetChanADC</td>
<td>Select A/D channel to be used.</td>
</tr>
</tbody>
</table>
## 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 converter 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

**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

**File Name:** `adcopen.c`
Hardware Peripheral Functions

OpenADC
PIC18CXX2, PIC18FXX2, PIC18FXX8, PIC18FXX39 (Continued)

<table>
<thead>
<tr>
<th>A/D voltage reference source:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_8ANA_0REF VREF+=VDD, VREF-=VSS,</td>
</tr>
<tr>
<td>ADC_8ANA_1REF AN3=VREF+, All analog channels</td>
</tr>
<tr>
<td>ADC_7ANA_1REF AN3=VREF+, All analog channels except AN3</td>
</tr>
<tr>
<td>ADC_6ANA_2REF AN3=VREF+, AN2=VREF</td>
</tr>
<tr>
<td>ADC_6ANA_0REF VREF+=VDD, VREF-=VSS</td>
</tr>
<tr>
<td>ADC_5ANA_1REF AN3=VREF+, VREF-=VSS</td>
</tr>
<tr>
<td>ADC_5ANA_0REF VREF+=VDD, VREF-=VSS</td>
</tr>
<tr>
<td>ADC_4ANA_2REF AN3=VREF+, AN2=VREF-</td>
</tr>
<tr>
<td>ADC_4ANA_1REF AN3=VREF+</td>
</tr>
<tr>
<td>ADC_3ANA_2REF AN3=VREF+, AN2=VREF-</td>
</tr>
<tr>
<td>ADC_3ANA_0REF VREF+=VDD, VREF-=VSS</td>
</tr>
<tr>
<td>ADC_2ANA_2REF AN3=VREF+, AN2=VREF-</td>
</tr>
<tr>
<td>ADC_2ANA_1REF AN3=VREF+</td>
</tr>
<tr>
<td>ADC_1ANA_2REF AN3=VREF+, AN2=VREF-, AN0=A</td>
</tr>
<tr>
<td>ADC_1ANA_0REF AN0 is analog input</td>
</tr>
<tr>
<td>ADC_0ANA_0REF All digital I/O</td>
</tr>
</tbody>
</table>

`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`.

<table>
<thead>
<tr>
<th>Channel:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_CH0                      Channel 0</td>
</tr>
<tr>
<td>ADC_CH1                      Channel 1</td>
</tr>
<tr>
<td>ADC_CH2                      Channel 2</td>
</tr>
<tr>
<td>ADC_CH3                      Channel 3</td>
</tr>
<tr>
<td>ADC_CH4                      Channel 4</td>
</tr>
<tr>
<td>ADC_CH5                      Channel 5</td>
</tr>
<tr>
<td>ADC_CH6                      Channel 6</td>
</tr>
<tr>
<td>ADC_CH7                      Channel 7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A/D Interrupts:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_INT_ON       Interrupts enabled</td>
</tr>
<tr>
<td>ADC_INT_OFF      Interrupts disabled</td>
</tr>
</tbody>
</table>

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 converter.
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 port configuration:
  ADC_0ANA        Analog:AN0    Digital:AN1-AN15
  ADC_1ANA        Analog:AN0-AN1 Digital:AN2-AN15
  ADC_2ANA        Analog:AN0-AN2 Digital:AN3-AN15
  ADC_3ANA        Analog:AN0-AN3 Digital:AN4-AN15
  ADC_4ANA        Analog:AN0-AN4 Digital:AN5-AN15
  ADC_5ANA        Analog:AN0-AN5 Digital:AN6-AN15
  ADC_6ANA        Analog:AN0-AN6 Digital:AN7-AN15
  ADC_7ANA        Analog:AN0-AN7 Digital:AN8-AN15
  ADC_8ANA        Analog:AN0-AN8 Digital:AN9-AN15
  ADC_9ANA        Analog:AN0-AN9 Digital:AN10-AN15
  ADC_10ANA       Analog:AN0-AN10 Digital:AN11-AN15
  ADC_11ANA       Analog:AN0-AN11 Digital:AN12-AN15
  ADC_12ANA       Analog:AN0-AN12 Digital:AN13-AN15
  ADC_13ANA       Analog:AN0-AN13 Digital:AN14-AN15
  ADC_14ANA       Analog:AN0-AN14 Digital:AN15-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.
OpenADC
PIC18C658/858, PIC18C601/801,
PIC18F6X20, PIC18F8X20 (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

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:
```c
OpenADC( ADC_FOSC_32    &
         ADC_RIGHT_JUST &
         ADC_14ANA,     
         ADC_CH0        &
         ADC_INT_OFF    );
```
OpenADC
All Other Processors

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.

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
Hardware Peripheral Functions

OpenADC
All Other Processors (Continued)

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
### 2.2.2 Example Use of the A/D Converter Routines

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

int result;

void main( void )
{
    // configure A/D converter
    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
}
```

---

**SetChanADC**

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

**Include:** adc.h

**Prototype:**

```c
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:**

```c
SetChanADC( ADC_CH0 );
```
2.3 INPUT CAPTURE FUNCTIONS

The capture peripheral is supported with the following functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloseCapture(x)</td>
<td>Disable capture peripheral (x).</td>
</tr>
<tr>
<td>OpenCapture(x)</td>
<td>Configure capture peripheral (x).</td>
</tr>
<tr>
<td>ReadCapture(x)</td>
<td>Read a value from capture peripheral (x).</td>
</tr>
<tr>
<td>CloseECapture(x)</td>
<td>Disable enhanced capture peripheral (x).</td>
</tr>
<tr>
<td>OpenECapture(x)</td>
<td>Configure enhanced capture peripheral (x).</td>
</tr>
<tr>
<td>ReadECapture(x)</td>
<td>Read a value from enhanced capture peripheral (x).</td>
</tr>
</tbody>
</table>

Note 1: The enhanced capture functions are only available on those devices with an ECCP\(x\)CON register.

2.3.1 Function Descriptions

- **CloseCapture1**
  - Function: Disable input capture \(x\).
  - Include: capture.h
  - Prototype: `void CloseCapture1( void );`
  - Remarks: This function disables the interrupt corresponding to the specified input capture.
  - File Name: `cp1close.c`

- **CloseCapture2**
  - Include: capture.h
  - Prototype: `void CloseCapture2( void );`

- **CloseCapture3**
  - Include: capture.h
  - Prototype: `void CloseCapture3( void );`

- **CloseCapture4**
  - Include: capture.h
  - Prototype: `void CloseCapture4( void );`

- **CloseCapture5**
  - Include: capture.h
  - Prototype: `void CloseCapture5( void );`

- **CloseECapture1**
  - Include: capture.h
  - Prototype: `void CloseECapture1( void );`

- **CloseECapture2**
  - Include: capture.h
  - Prototype: `void CloseECapture2( void );`

- **CloseECapture3**
  - Include: capture.h
  - Prototype: `void CloseECapture3( void );`

- **CloseECapture4**
  - Include: capture.h
  - Prototype: `void CloseECapture4( void );`

- **CloseECapture5**
  - Include: capture.h
  - Prototype: `void CloseECapture5( void );`
OpenCapture1
OpenCapture2
OpenCapture3
OpenCapture4
OpenCapture5
OpenECapture1

**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 );
void OpenECapture1( 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:

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
- EC1_EVERY_FALL_EDGE: Interrupt on every falling edge (enhanced)
- EC1_EVERY_RISE_EDGE: Interrupt on every rising edge (enhanced)
- EC1_EVERY_4_RISE_EDGE: Interrupt on every 4th rising edge (enhanced)
- EC1_EVERY_16_RISE_EDGE: Interrupt on every 16th rising edge (enhanced)

**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
- ECap1OVF

In addition to opening the capture, the appropriate timer module must
be enabled before any of the captures will operate. See Section 2.9
"Timer Functions" for information on using the Timer runtime library
functions for this.
OpenCapture1
OpenCapture2
OpenCapture3
OpenCapture4
OpenCapture5
OpenECapture1 (Continued)

File Name: 
  cp1open.c
  cp2open.c
  cp3open.c
  cp4open.c
  cp5open.c
  eplopen.c

Code Example:
OpenCapture1( CAPTURE_INT_ON &
               C1_EVERY_4_RISE_EDGE );

ReadCapture1
ReadCapture2
ReadCapture3
ReadCapture4
ReadCapture5
ReadECapture1

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 );
  unsigned int ReadECapture1( 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
  ep1read.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.

```c
#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:

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

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEAckPolling</td>
<td>Generate the Acknowledge polling sequence.</td>
</tr>
<tr>
<td>EEBByteWrite</td>
<td>Write a single byte.</td>
</tr>
<tr>
<td>EECurrentAddRead</td>
<td>Read a single byte from the next location.</td>
</tr>
<tr>
<td>EEPPageWrite</td>
<td>Write a string of data.</td>
</tr>
<tr>
<td>EERandomRead</td>
<td>Read a single byte from an arbitrary address.</td>
</tr>
<tr>
<td>EESequentialRead</td>
<td>Read a string of data.</td>
</tr>
</tbody>
</table>

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 I2C 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 I2C bus operating in master I2C 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 I2C device.
- **length**: Number of bytes to read from I2C device.

Remarks: This routine reads a predefined data string length from the I2C 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 I²C bus is Idle.

**Include:** i2c.h

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

**Remarks:**
This function checks the state of the I²C peripheral and waits for the bus to become available. The IdleI2C function is required since the hardware I²C peripheral does not allow for spooling of bus sequences. The I²C peripheral must be in an Idle state before an I²C operation can be initiated or a write collision will be generated.

**File Name:** idlei2c.c

## NotAckI2C

**Function:** Generate I²C bus Not Acknowledge condition.

**Include:** i2c.h

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

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

**File Name:** noacki2c.c

## OpenI2C

**Function:** Configure the SSP module.

**Include:** i2c.h

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

**Arguments:**
- **sync_mode**
  One of the following values, defined in i2c.h:
  - SLAVE_7 I²C Slave mode, 7-bit address
  - SLAVE_10 I²C Slave mode, 10-bit address
  - MASTER I²C 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:**
```c
OpenI2C(MASTER, SLEW_ON);
```

## putcI2C

*See WriteI2C.*
### putsI2C

**Function:** Write a data string to the I²C bus operating in either Master or Slave mode.

**Include:** i2c.h  
**Prototype:**
```c
unsigned char putsI2C(
    unsigned char *wrptr );
```

**Arguments:**  
- `wrptr` Pointer to data that will be written to the I²C bus.

**Remarks:** This routine writes a data string to the I²C 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 I²C mode:
  - 0 if the null character was reached in the data string
  - -2 if the slave I²C device responded with a NOT ACK
  - -3 if a write collision occurred
- Slave I²C mode:
  - 0 if the null character was reached in the data string
  - -2 if the master I²C device responded with a NOT ACK which terminated the data transfer

**File Name:** putsi2c.c

**Code Example:**
```c
unsigned char string[] = "data to send";
putsi2c(string);
```

### ReadI2C
getI2C

**Function:** Read a single byte from the I²C bus.

**Include:** i2c.h  
**Prototype:**
```c
unsigned char ReadI2C (void );
```

**Remarks:** This function reads in a single byte from the I²C bus.

**Return Value:** The data byte read from the I²C bus.

**File Name:** readi2c.c

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

### RestartI2C

**Function:** Generate an I²C bus Restart condition.

**Include:** i2c.h  
**Prototype:**
```c
void RestartI2C( void );
```

**Remarks:** This function generates an I²C 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 I²C 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 I²C 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: i2ceap.c

Code Example: temp = EEAckPolling(0xA0);

EEByteWrite

Function: Write a single byte to the I²C 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 I²C bus. This routine can be used for any Microchip I²C 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 I²C bus.

**Include:** i2c.h

**Prototype:**

```c
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 I²C bus. The address location of the data to read is that of the current pointer within the I²C 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:**

```c
temp = EECURRENTADDREAD(0xA1);
```

## EEPageWrite

**Function:** Write a string of data to the EE device from the I²C bus.

**Include:** i2c.h

**Prototype:**

```c
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 I²C 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:**

```c
temp = EEPageWrite(0xA0, 0x70, wrptr);
```
### EERandomRead

**Function:** Read a single byte from the I²C bus.

**Include:** i2c.h

**Prototype:**
```c
#include "i2c.h"

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 I²C bus. The routine can be used for Microchip I²C 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:**
```c
unsigned int temp;
temp = EERandomRead(0xA0,0x30);
```

### EESequentialRead

**Function:** Read a string of data from the I²C bus.

**Include:** i2c.h

**Prototype:**
```c
#include "i2c.h"

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 I²C bus. The routine can be used for Microchip I²C 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

**File Name:** i2ceesr.c

**Code Example:**
```c
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 I2C master communication. The routine illustrates I2C communications with a Microchip 24LC01B I2C 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);
        EESuccessiveRead(0xA0, 0x70, arrayrd, 20);
        EERandomRead(0xA0,0x30);
    }
}
```
2.5 I/O PORT FUNCTIONS

PORTB is supported with the following functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClosePORTB</td>
<td>Disable the interrupts and internal pull-up resistors for PORTB.</td>
</tr>
<tr>
<td>CloseRB0INT</td>
<td>Disable interrupts for PORTB pin 0.</td>
</tr>
<tr>
<td>DisablePullups</td>
<td>Disable the internal pull-up resistors on PORTB.</td>
</tr>
<tr>
<td>EnablePullups</td>
<td>Enable the internal pull-up resistors on PORTB.</td>
</tr>
<tr>
<td>OpenPORTB</td>
<td>Configure the interrupts and internal pull-up resistors on PORTB.</td>
</tr>
<tr>
<td>OpenRB0INT</td>
<td>Enable interrupts for PORTB pin 0.</td>
</tr>
<tr>
<td>OpenRB1INT</td>
<td>Enable interrupts for PORTB pin 1.</td>
</tr>
<tr>
<td>OpenRB2INT</td>
<td>Enable interrupts for PORTB pin 2.</td>
</tr>
</tbody>
</table>

2.5.1 Function Descriptions

**ClosePORTB**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClosePORTB</td>
<td>Disable the interrupts and internal pull-up resistors for PORTB.</td>
</tr>
<tr>
<td>Include:</td>
<td>portb.h</td>
</tr>
<tr>
<td>Prototype:</td>
<td>void ClosePORTB( void );</td>
</tr>
<tr>
<td>Remarks:</td>
<td>This function disables the PORTB interrupt-on-change and the internal pull-up resistors.</td>
</tr>
<tr>
<td>File Name:</td>
<td>pbclose.c</td>
</tr>
</tbody>
</table>

**CloseRB0INT**

**CloseRB1INT**

**CloseRB2INT**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloseRB0INT</td>
<td>Disable the interrupts for the specified PORTB pin.</td>
</tr>
<tr>
<td>CloseRB1INT</td>
<td>Disable the interrupts for the specified PORTB pin.</td>
</tr>
<tr>
<td>CloseRB2INT</td>
<td>Disable the interrupts for the specified PORTB pin.</td>
</tr>
<tr>
<td>Include:</td>
<td>portb.h</td>
</tr>
<tr>
<td>Prototype:</td>
<td>void CloseRB0INT( void );</td>
</tr>
<tr>
<td></td>
<td>void CloseRB1INT( void );</td>
</tr>
<tr>
<td></td>
<td>void CloseRB2INT( void );</td>
</tr>
<tr>
<td>Remarks:</td>
<td>This function disables the PORTB interrupt-on-change.</td>
</tr>
<tr>
<td>File Name:</td>
<td>rb0close.c</td>
</tr>
<tr>
<td></td>
<td>rb1close.c</td>
</tr>
<tr>
<td></td>
<td>rb2close.c</td>
</tr>
</tbody>
</table>

**DisablePullups**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisablePullups</td>
<td>Disable the internal pull-up resistors on PORTB.</td>
</tr>
<tr>
<td>Include:</td>
<td>portb.h</td>
</tr>
<tr>
<td>Prototype:</td>
<td>void DisablePullups( void );</td>
</tr>
<tr>
<td>Remarks:</td>
<td>This function disables the internal pull-up resistors on PORTB.</td>
</tr>
<tr>
<td>File Name:</td>
<td>pulldis.c</td>
</tr>
</tbody>
</table>
## EnablePullups

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

**Include:** portb.h

**Prototype:**
```c
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:**
```c
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:**
```c
OpenPORTB( PORTB_CHANGE_INT_ON & PORTB_PULLUPS_ON);
```

## OpenRB0INT

**Function:** Enable interrupts for the specified PORTB pin.

**Include:** portb.h

**Prototype:**
```c
void OpenRB0INT( unsigned char config );
void OpenRB1INT( unsigned char config );
void OpenRB2INT( 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
  - **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.
Microwire communication is supported with the following functions:

### TABLE 2-6: MICROWIRE FUNCTIONS

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloseMwire</td>
<td>Disable the SSP module used for Microwire communication.</td>
</tr>
<tr>
<td>DataRdyMwire</td>
<td>Indicate completion the internal write cycle.</td>
</tr>
<tr>
<td>getcMwire</td>
<td>Read a byte from the Microwire device.</td>
</tr>
<tr>
<td>getsMwire</td>
<td>Read a string from the Microwire device.</td>
</tr>
<tr>
<td>OpenMwire</td>
<td>Configure the SSP module for Microwire use.</td>
</tr>
<tr>
<td>putcMwire</td>
<td>Write a byte to the Microwire device.</td>
</tr>
<tr>
<td>ReadMwire</td>
<td>Read a byte from the Microwire device.</td>
</tr>
<tr>
<td>WriteMwire</td>
<td>Write a byte to the Microwire device.</td>
</tr>
</tbody>
</table>

#### 2.6.1 Function Descriptions

**CloseMwire**

**Function:** Disable the SSP module.

**Include:** mwire.h

**Prototype:**

```c
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:**

```c
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:**

```c
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:
                 MWIRE_FOSC_4    clock = FOSC/4
                 MWIRE_FOSC_16   clock = FOSC/16
                 MWIRE_FOSC_64   clock = FOSC/64
                 MWIRE_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(MWIRE_FOSC_16);

putcMwire

See WriteMwire.
# ReadMwire

**getcMwire**

**Function:** Read a byte from a Microwire device.

**Include:** mwire.h

**Prototype:**

```c
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:**

```c
ReadMwire(0x03, 0x00);
```

# WriteMwire

**putcMwire**

**Function:**

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

**Include:** mwire.h

**Prototype:**

```c
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:**

```c
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.

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

#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;   //ensure CS is negated
    W_CS = 0;               //ensure CS is negated
    OpenMwire(MWIRE_FOSC_16);  //enable SSP peripheral
    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:

### TABLE 2-7: PWM FUNCTIONS

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClosePWMx</td>
<td>Disable PWM channel x.</td>
</tr>
<tr>
<td>OpenPWMx</td>
<td>Configure PWM channel x.</td>
</tr>
<tr>
<td>SetDCPWMx</td>
<td>Write a new duty cycle value to PWM channel x.</td>
</tr>
<tr>
<td>SetOutputPWMx</td>
<td>Sets the PWM output configuration bits for ECCP x.</td>
</tr>
<tr>
<td>CloseEPWMx(1)</td>
<td>Disable enhanced PWM channel x.</td>
</tr>
<tr>
<td>OpenEPWMx(1)</td>
<td>Configure enhanced PWM channel x.</td>
</tr>
<tr>
<td>SetDCEPWMx(1)</td>
<td>Write a new duty cycle value to enhanced PWM channel x.</td>
</tr>
<tr>
<td>SetOutputEPWMx(1)</td>
<td>Sets the enhanced PWM output configuration bits for ECCP x.</td>
</tr>
</tbody>
</table>

**Note 1:** The enhanced PWM functions are only available on those devices with an ECCPxCON register.

### 2.7.1 Function Descriptions

**ClosePWM1**
**ClosePWM2**
**CloseEPWM1**

**Function:** Disable PWM channel.

**Include:** `pwm.h`

**Prototype:**
- `void ClosePWM1( void );`
- `void ClosePWM2( void );`
- `void CloseEPWM1( void );`

**Remarks:** This function disables the specified PWM channel.

**File Name:**
- `pw1close.c`
- `pw2close.c`
- `ew1close.c`

**OpenPWM1**
**OpenPWM2**
**OpenEPWM1**

**Function:** Configure PWM channel.

**Include:** `pwm.h`

**Prototype:**
- `void OpenPWM1( char period );`
- `void OpenPWM2( char period );`
- `void OpenEPWM1( char period );`

**Arguments:**
- `period`
  Can be any value from 0x00 to 0xff. This value determines the PWM frequency by using the following formula:
  
  \[
  \text{PWM period} = [(\text{period } + 1) \times 4 \times \text{ToSC} \times \text{TMR2 prescaler}]
  \]
OpenPWM1
OpenPWM2
OpenEPWM1 (Continued)

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
ewlopen.c

Code Example: OpenPWM1(0xff);

SetDCPWM1
SetDCPWM2
SetDCEPWM1

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 );
void SetDCEPWM1( 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
ewlsetdc.c

Code Example: SetDCPWM1(0);
### SetOutputPWM1
### SetOutputEPWM1

**Function:**
Sets the PWM output configuration bits for ECCP.

**Include:**
`pwm.h`

**Prototype:**
```c
#include <pwm.h>

void SetOutputPWM1 (unsigned char outputconfig, unsigned char outputmode);
void SetOutputEPWM1 (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 forward
  - `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

**Remarks:**
This is only applicable to those devices with Extended or Enhanced CCP (ECCP).

**File Name:**
- `pw1setoc.c`
- `ew1setoc.c`

**Code Example:**
```c
SetOutputPWM1 (SINGLE_OUT, PWM_MODE_1);
```
2.8 SPI™ FUNCTIONS

SPI communication is supported with the following functions:

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

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:**
```c
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:**
```c
unsigned char wrptr(10);
getsSPI(wrptr, 10);
```

### OpenSPI

**Function:** Initialize the SSP module.

**Include:** spi.h

**Prototype:**
```c
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:
  - SPI_FOSC_4 SPI Master mode, clock = Fosc/4
  - SPI_FOSC_16 SPI Master mode, clock = Fosc/16
  - SPI_FOSC_64 SPI Master mode, clock = Fosc/64
  - SPI_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:**
```c
OpenSPI(SPI_FOSC_16, MODE_00, SMPEND);
```

### putcSPI

See WriteSPI.
### putsSPI

**Function:**
Write a string to the SPI bus.

**Include:**
spi.h

**Prototype:**
```c
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:**
```c
unsigned char wrptr[] = "Hello!";
putsSPI(wrptr);
```

### ReadSPI

**Function:**
Read a byte from the SPI bus.

**Include:**
spi.h

**Prototype:**
```c
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:**
reads.spi.c

**Code Example:**
```c
char x;

x = ReadSPI();
```

### WriteSPI

**Function:**
Write a byte to the SPI bus.

**Include:**
spi.h

**Prototype:**
```c
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:**
```c
WriteSPI('a');
```
2.8.2 Example of Use

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

```c
#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(SPI_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);

    busy_polling();
    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 = getcSPI();         //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:

### TABLE 2-9: TIMER FUNCTIONS

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloseTimer x</td>
<td>Disable timer x.</td>
</tr>
<tr>
<td>OpenTimer x</td>
<td>Configure timer x.</td>
</tr>
<tr>
<td>ReadTimer x</td>
<td>Read the value of timer x.</td>
</tr>
<tr>
<td>WriteTimer x</td>
<td>Write a value into timer x.</td>
</tr>
</tbody>
</table>

#### 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

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
OpenTimer0 (Continued)

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

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.
### OpenTimer1 (Continued)

**File Name:** t1open.c

**Code Example:**
```
OpenTimer1( TIMER_INT_ON &
    T1_8BIT_RW &
    T1_SOURCE_EXT &
    T1_PS_1_1 &
    T1_OSCIEN_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

**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_PS_1_16 &
    T2_POST_1_8 )
```
## OpenTimer3

<table>
<thead>
<tr>
<th><strong>Function:</strong></th>
<th>Configure timer3.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Include:</strong></td>
<td>timers.h</td>
</tr>
<tr>
<td><strong>Prototype:</strong></td>
<td><code>void OpenTimer3( unsigned char config );</code></td>
</tr>
<tr>
<td><strong>Arguments:</strong></td>
<td><code>config</code></td>
</tr>
</tbody>
</table>

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`.

<table>
<thead>
<tr>
<th><strong>Enable Timer3 Interrupt:</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TIMER_INT_ON</code></td>
<td>Interrupt enabled</td>
</tr>
<tr>
<td><code>TIMER_INT_OFF</code></td>
<td>Interrupt disabled</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Timer Width:</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>T3_8BIT_RW</code></td>
<td>8-bit mode</td>
</tr>
<tr>
<td><code>T3_16BIT_RW</code></td>
<td>16-bit mode</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Clock Source:</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>T3_SOURCE_EXT</code></td>
<td>External clock source (I/O pin)</td>
</tr>
<tr>
<td><code>T3_SOURCE_INT</code></td>
<td>Internal clock source (ToSc)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Prescale Value:</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>T3_PS_1_1</code></td>
<td>1:1 prescale</td>
</tr>
<tr>
<td><code>T3_PS_1_2</code></td>
<td>1:2 prescale</td>
</tr>
<tr>
<td><code>T3_PS_1_4</code></td>
<td>1:4 prescale</td>
</tr>
<tr>
<td><code>T3_PS_1_8</code></td>
<td>1:8 prescale</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Synchronize Clock Input:</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>T3_SYNC_EXT_ON</code></td>
<td>Sync external clock input</td>
</tr>
<tr>
<td><code>T3_SYNC_EXT_OFF</code></td>
<td>Don’t sync external clock input</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Use With CCP:</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>T1_SOURCE_CCP</code></td>
<td>Timer1 source for both CCP’s</td>
</tr>
<tr>
<td><code>T3_SOURCE_CCP</code></td>
<td>Timer3 source for both CCP’s</td>
</tr>
<tr>
<td><code>T1_CCP1_T3_CCP2</code></td>
<td>Timer1 source for CCP1 and Timer3 source for CCP2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Remarks:</strong></th>
<th>This function configures timer3 according to the options specified.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File Name:</strong></td>
<td>t3open.c</td>
</tr>
</tbody>
</table>
| **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 );
Hardware Peripheral Functions

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 );

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

**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

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

**Code Example:**
```
WriteTimer0( 10000 );
```
2.9.2 Example of Use

```c
#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();             // close modules
    }
```
2.10 USART FUNCTIONS

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

**TABLE 2-10: SINGLE USART PERIPHERAL FUNCTIONS**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusyUSART</td>
<td>Is the USART transmitting?</td>
</tr>
<tr>
<td>CloseUSART</td>
<td>Disable the USART.</td>
</tr>
<tr>
<td>DataRdyUSART</td>
<td>Is data available in the USART read buffer?</td>
</tr>
<tr>
<td>getUSART</td>
<td>Read a byte from the USART.</td>
</tr>
<tr>
<td>getsUSART</td>
<td>Read a string from the USART.</td>
</tr>
<tr>
<td>OpenUSART</td>
<td>Configure the USART.</td>
</tr>
<tr>
<td>putcUSART</td>
<td>Write a byte to the USART.</td>
</tr>
<tr>
<td>putsUSART</td>
<td>Write a string from data memory to the USART.</td>
</tr>
<tr>
<td>putrsUSART</td>
<td>Write a string from program memory to the USART.</td>
</tr>
<tr>
<td>ReadUSART</td>
<td>Read a byte from the USART.</td>
</tr>
<tr>
<td>WriteUSART</td>
<td>Write a byte to the USART.</td>
</tr>
<tr>
<td>baudUSART</td>
<td>Set the baud rate configuration bits for enhanced USART.</td>
</tr>
</tbody>
</table>

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

**TABLE 2-11: MULTIPLE USART PERIPHERAL FUNCTIONS**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusyUSART x</td>
<td>Is USART x transmitting?</td>
</tr>
<tr>
<td>CloseUSART x</td>
<td>Disable USART x.</td>
</tr>
<tr>
<td>DataRdyUSART x</td>
<td>Is data available in the read buffer of USART x?</td>
</tr>
<tr>
<td>getUSART x</td>
<td>Read a byte from USART x.</td>
</tr>
<tr>
<td>getsUSART x</td>
<td>Read a string from USART x.</td>
</tr>
<tr>
<td>OpenUSART x</td>
<td>Configure USART x.</td>
</tr>
<tr>
<td>putcUSART x</td>
<td>Write a byte to USART x.</td>
</tr>
<tr>
<td>putsUSART x</td>
<td>Write a string from data memory to USART x.</td>
</tr>
<tr>
<td>putrsUSART x</td>
<td>Write a string from program memory to USART x.</td>
</tr>
<tr>
<td>ReadUSART x</td>
<td>Read a byte from USART x.</td>
</tr>
<tr>
<td>WriteUSART x</td>
<td>Write a byte to USART x.</td>
</tr>
<tr>
<td>baudUSART x</td>
<td>Set the baud rate configuration bits for enhanced USART x.</td>
</tr>
</tbody>
</table>
2.10.1 Function Descriptions

**BusyUSART**

*Function:* Is the USART transmitting?

*Include:* `usart.h`

*Prototype:*

```c
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**

*Function:* Disable the specified USART.

*Include:* `usart.h`

*Prototype:*

```c
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.

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

Code Example:
char inputstr[10];
getsUSART( inputstr, 5 );
Hardware Peripheral Functions

OpenUSART
Open1USART
Open2USART

Function: Configure the specified USART module.
Include: 
Prototype: 
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

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:
\[ \frac{F_{OSC}}{16 \times (spbrg + 1)} \]
Asynchronous mode, low speed:
\[ \frac{F_{OSC}}{64 \times (spbrg + 1)} \]
Synchronous mode:
\[ \frac{F_{OSC}}{4 \times (spbrg + 1)} \]

Where \( F_{OSC} \) 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:  

```c
OpenUSART1( USART_TX_INT_OFF &
             USART_RX_INT_OFF &
             USARTASYNCH_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:**  
```c
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:**  
```c
putrsUSART( "Hello World!" );
```
Hardware Peripheral Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Read a byte (one character) out of the USART receive buffer, including the 9th bit if enabled.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td>usart.h</td>
</tr>
<tr>
<td>Prototype</td>
<td>char getcUSART( void ); char getc1USART( void ); char getc2USART( void ); char ReadUSART( void ); char Read1USART( void ); char Read2USART( void );</td>
</tr>
</tbody>
</table>

**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:

```c
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 `getcUSARTx` and `Read USARTx` 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:**

```c
int result;
result = ReadUSART();
result |= (unsigned int) USART_Status.RX_NINE << 8;
```
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WriteUSART</td>
<td>Write a byte (one character) to the USART transmit buffer, including the</td>
</tr>
<tr>
<td>Write1USART</td>
<td>9th bit if enabled.</td>
</tr>
<tr>
<td>Write2USART</td>
<td></td>
</tr>
<tr>
<td>putcUSART</td>
<td></td>
</tr>
<tr>
<td>putc1USART</td>
<td></td>
</tr>
<tr>
<td>putc2USART</td>
<td></td>
</tr>
<tr>
<td>WriteUSART</td>
<td></td>
</tr>
<tr>
<td>Write1USART</td>
<td></td>
</tr>
<tr>
<td>Write2USART</td>
<td></td>
</tr>
</tbody>
</table>

**Include:**
```
#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 `putcUSART` and `WriteUSART` functions should be used and the Status register is named `USARTx_Status` which is of the type `USART` described above.

**File Name:**
```
uwrite.c
ulwrite.c
u2write.c
```

**Code Example:**
```
unsigned int outval;
USART1_Status.TX_NINE = (outval & 0x0100) >> 8;
WriteUSART( (char) outval );
```
**baudUSART**  
**baud1USART**  
**baud2USART**

<table>
<thead>
<tr>
<th>Function:</th>
<th>Set the baud rate configuration bits for enhanced USART operation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td><code>usart.h</code></td>
</tr>
</tbody>
</table>
| Prototype:| `void baudUSART( unsigned char baudconfig );`  
`void baud1USART( unsigned char baudconfig );`  
`void baud2USART( unsigned char baudconfig );` |
| Arguments:| `baudconfig`  
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`:  
**Clock Idle State:**  
- **BAUD_IDLE_CLK_HIGH**: Clock idle state is a high level  
- **BAUD_IDLE_CLK_LOW**: Clock idle state is a low level  
**Baud Rate Generation:**  
- **BAUD_16_BIT_RATE**: 16-bit baud generation rate  
- **BAUD_8_BIT_RATE**: 8-bit baud generation rate  
**RX Pin Monitoring:**  
- **BAUD_WAKEUP_ON**: RX pin monitored  
- **BAUD_WAKEUP_OFF**: RX pin not monitored  
**Baud Rate Measurement:**  
- **BAUD_AUTO_ON**: Auto baud rate measurement enabled  
- **BAUD_AUTO_OFF**: Auto baud rate measurement disabled  |
| Remarks:  | These functions are only available for processors with enhanced  
USART capability. |
| File Name:| `ubaud.c`  
`u1baud.c`  
`u2baud.c` |
| Code Example:| `baudUSART (BAUD_IDLE_CLK_HIGH &  
BAUD_16_BIT_RATE &  
BAUD_WAKEUP_ON &  
BAUD_AUTO_ON);` |
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();
}
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\traditional\pmc and src\extended\pmc subdirectories 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 (Section 3.2 “External LCD Functions”)
- External CAN2510 Functions (Section 3.3 “External CAN2510 Functions”)
- Software I²C™ Functions (Section 3.4 “Software I²C™ Functions”)
- Software SPI Functions (Section 3.5 “Software SPI® Functions”)
- Software UART Functions (Section 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:

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

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:
TABLE 3-2: MACROS FOR SELECTING LCD PIN ASSIGNMENTS

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

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.

TABLE 3-3: MACROS FOR SELECTING 4- OR 8-BIT MODE

<table>
<thead>
<tr>
<th>Macro</th>
<th>Default Value</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT8</td>
<td>not defined</td>
<td>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.</td>
</tr>
<tr>
<td>UPPER</td>
<td>not defined</td>
<td>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.</td>
</tr>
</tbody>
</table>

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:

TABLE 3-4: XLCD DELAY FUNCTIONS

<table>
<thead>
<tr>
<th>Function</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>DelayFor18TCY</td>
<td>Delay for 18 cycles.</td>
</tr>
<tr>
<td>DelayPORXLCD</td>
<td>Delay for 15 ms.</td>
</tr>
<tr>
<td>DelayXLCD</td>
<td>Delay for 5 ms.</td>
</tr>
</tbody>
</table>
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.
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 SetRamAddr 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.
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:**
```c
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`:
  - `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:**
```c
while( BusyXLCD() );
WriteCmdXLCD( EIGHT_BIT & LINES_5X7 );
WriteCmdXLCD( BLINK_ON );
WriteCmdXLCD( SHIFT_DISP_LEFT );
```

---

### WriteDataXLCD

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

**Include:** `xlcd.h`

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

**Arguments:**
- `data` The value of `data` can be any 8-bit value, but should correspond to the character RAM table of the 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:** `writedata.c`
3.2.2 Example of Use

```c
#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();
    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:

TABLE 3-5: EXTERNAL CAN2510 FUNCTIONS

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

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

**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:**
```c
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`: Number of bytes in the message.
- `data`: Pointer to the buffer where the message data is stored.

**Remarks:** This function reads a message from the specified receive buffer.
**CAN2510DataRead (Continued)**

```
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:**
```c
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 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 CS pin. After the modification, the processor-specific library must be rebuilt. See Section 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:**
```c
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 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 CS pin. After the modification, the processor-specific library must be rebuilt. See Section 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:**
```c
unsigned char CAN2510ErrorState( void );
```

**Remarks:**
This function returns the Error State of the CAN bus. The Error State depends 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 \(\leq\) 95 and REC \(\leq\) 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:**
```c
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_RESET**
  Reset the MCP2510
- **CAN2510_NORESET**
  Don't 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
CAN2510Init (Continued)

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

RX1BF Pin Setting
Controlled by the B1BFS:B1BFE:B1BFM bits (BFPCTRL register)
CAN2510_RX1BF_OFF RX1BF pin is high-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 high-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

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
Software Peripheral Library

CAN2510Init (Continued)

**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 BRPS:BRP0 bits (CNF1 register)
- CAN2510_BRG_1X \( T_q = 1 \times (2T_{osc}) \)
- CAN2510_BRG_64X \( T_q = 64 \times (2T_{osc}) \)

**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

**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 BTIMODE 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.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN2510_RX_FILTER</td>
<td>Filtering on RX pin when in Sleep mode</td>
</tr>
<tr>
<td>CAN2510_RX_NOPFILTER</td>
<td>No filtering on RX pin when in Sleep mode</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN2510_NONE_EN</td>
<td>No interrupts enabled</td>
</tr>
<tr>
<td>CAN2510_MSGERR_EN</td>
<td>Interrupt on error during message reception or transmission</td>
</tr>
<tr>
<td>CAN2510_WAKEUP_EN</td>
<td>Interrupt on CAN bus activity</td>
</tr>
<tr>
<td>CAN2510_ERROR_EN</td>
<td>Interrupt on EFLG error condition change</td>
</tr>
<tr>
<td>CAN2510_TXB2_EN</td>
<td>Interrupt on transmission buffer 2 becoming empty</td>
</tr>
<tr>
<td>CAN2510_TXB1_EN</td>
<td>Interrupt on transmission buffer 1 becoming empty</td>
</tr>
<tr>
<td>CAN2510_TXB0_EN</td>
<td>Interrupt on transmission buffer 0 becoming empty</td>
</tr>
<tr>
<td>CAN2510_RXB1_EN</td>
<td>Interrupt when message received in receive buffer 1</td>
</tr>
<tr>
<td>CAN2510_RXB0_EN</td>
<td>Interrupt when message received in receive buffer 0</td>
</tr>
</tbody>
</table>

**SPI_syncMode**
Specifies the PIC18CXXX SPI synchronization frequency:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN2510_SPI_FOSC4</td>
<td>Communicates at Fosc/4</td>
</tr>
<tr>
<td>CAN2510_SPI_FOSC16</td>
<td>Communicates at Fosc/16</td>
</tr>
<tr>
<td>CAN2510_SPI_FOSC64</td>
<td>Communicates at Fosc/64</td>
</tr>
<tr>
<td>CAN2510_SPI_FOSCTMR2</td>
<td>Communicates at TMR2/2</td>
</tr>
</tbody>
</table>

**SPI_busMode**
Specifies the PIC18CXXX SPI bus mode:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN2510_SPI_MODE00</td>
<td>Communicate using SPI mode 00</td>
</tr>
<tr>
<td>CAN2510_SPI_MODE01</td>
<td>Communicate using SPI mode 01</td>
</tr>
</tbody>
</table>

**SPI_smpPhase**
Specifies the PIC18CXXX SPI sample point:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN2510_SPI_SMPMID</td>
<td>Samples in middle of SPI bit</td>
</tr>
<tr>
<td>CAN2510_SPI_SMPEND</td>
<td>Samples at end of SPI bit</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>CAN2510_NONE_EN</th>
<th>No interrupts enabled (00000000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN2510_MSGERR_EN</td>
<td>Interrupt on error during message reception or transmission (10000000)</td>
</tr>
<tr>
<td>CAN2510_WAKEUP_EN</td>
<td>Interrupt on CAN bus activity (01000000)</td>
</tr>
<tr>
<td>CAN2510_ERROR_EN</td>
<td>Interrupt on EFLG error condition change (00100000)</td>
</tr>
<tr>
<td>CAN2510_TXB2_EN</td>
<td>Interrupt on transmission buffer 2 becoming empty (00010000)</td>
</tr>
<tr>
<td>CAN2510_TXB1_EN</td>
<td>Interrupt on transmission buffer 1 becoming empty (00001000)</td>
</tr>
<tr>
<td>CAN2510_TXB0_EN</td>
<td>Interrupt on transmission buffer 0 becoming empty (00000100)</td>
</tr>
<tr>
<td>CAN2510_RXB1_EN</td>
<td>Interrupt when message received in receive buffer 1 (00000010)</td>
</tr>
<tr>
<td>CAN2510_RXB0_EN</td>
<td>Interrupt when message received in receive buffer 0 (00000001)</td>
</tr>
</tbody>
</table>

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

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:**
```c
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`.

**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:** canloadsx.c
**CAN2510LoadRTRStd**

**Function:**

Loads a Standard remote frame into the specified transfer buffer.

**Required CAN Mode(s):**

All

**Include:**

`can2510.h`

**Prototype:**

```c
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`

**Prototype:**

```c
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 an 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.
CAN2510LoadRTRXtd (Continued)

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:**
```c
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:**
```c
void CAN2510WriteBuffer( unsigned char bufferNum );
```

**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:**
```c
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.
CAN2510SequentialRead (Continued)

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

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:**

```c
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:**

```c
unsigned char CAN2510SetMsgFilterStd( 
    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

**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 CAN2510SetMsgFilterXtd(
    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
  - Extented-length messages: Array of 4 unsigned integers
  For Buffer 1
  - Extented-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:**
```c
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:** `canfilt.c`

### CAN2510SetSingleFilterXtd

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

**Required CAN Mode(s):** Configuration mode

**Include:** `can2510.h`

**Prototype:**
```c
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:

```c
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:

```c
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:**
```c
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`
  - `CAN2510_PRI_HIGH`
  - `CAN2510_PRI_LOW`
  - `CAN2510_PRI_LOWEST`
- `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).
- **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 I²C bus using I/O pins from a PIC18 microcontroller. The following functions are provided:

**TABLE 3-6: I²C SOFTWARE FUNCTIONS**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock_test</td>
<td>Generate a delay for slave clock stretching.</td>
</tr>
<tr>
<td>SWAckI2C</td>
<td>Generate an I²C bus Acknowledge condition.</td>
</tr>
<tr>
<td>SWGetcI2C</td>
<td>Read a byte from the I²C bus.</td>
</tr>
<tr>
<td>SWGetsI2C</td>
<td>Read a data string.</td>
</tr>
<tr>
<td>SWNotAckI2C</td>
<td>Generate an I²C bus Acknowledge condition.</td>
</tr>
<tr>
<td>SWPutI2C</td>
<td>Write a single byte to the I²C bus.</td>
</tr>
<tr>
<td>SWPutsI2C</td>
<td>Write a string to the I²C bus.</td>
</tr>
<tr>
<td>SWReadI2C</td>
<td>Read a byte from the I²C bus.</td>
</tr>
<tr>
<td>SWRestartI2C</td>
<td>Generate an I²C bus Restart condition.</td>
</tr>
<tr>
<td>SWStartI2C</td>
<td>Generate an I²C bus Start condition.</td>
</tr>
<tr>
<td>SWStopI2C</td>
<td>Generate an I²C bus Stop condition.</td>
</tr>
<tr>
<td>SWWriteI2C</td>
<td>Write a single byte to the I²C bus.</td>
</tr>
</tbody>
</table>

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:

**TABLE 3-7: MACROS FOR SELECTING I²C PIN ASSIGNMENTS**

<table>
<thead>
<tr>
<th>I²C Line</th>
<th>Macros</th>
<th>Default Value</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA Pin</td>
<td>DATA_PIN</td>
<td>PORTBbits.RB4</td>
<td>Pin used for the DATA line.</td>
</tr>
<tr>
<td></td>
<td>DATA_LAT</td>
<td>LATBbits.RB4</td>
<td>Latch associated with DATA pin.</td>
</tr>
<tr>
<td></td>
<td>DATA_LOW</td>
<td>TRISBbits.TRISB4 = 0;</td>
<td>Statement to configure the DATA pin as an output.</td>
</tr>
<tr>
<td></td>
<td>DATA_HI</td>
<td>TRISBbits.TRISB4 = 1;</td>
<td>Statement to configure the DATA pin as an input.</td>
</tr>
<tr>
<td>CLOCK Pin</td>
<td>SCLK_PIN</td>
<td>PORTBbits.RB3</td>
<td>Pin used for the CLOCK line.</td>
</tr>
<tr>
<td></td>
<td>SCLK_LAT</td>
<td>LATBbits.LATB3</td>
<td>Latch associated with the CLOCK pin.</td>
</tr>
<tr>
<td></td>
<td>CLOCK_LOW</td>
<td>TRISBbits.TRISB3 = 0;</td>
<td>Statement to configure the CLOCK pin as an output.</td>
</tr>
<tr>
<td></td>
<td>CLOCK_HI</td>
<td>TRISBbits.TRISB3 = 1;</td>
<td>Statement to configure the CLOCK pin as an input.</td>
</tr>
</tbody>
</table>

After these definitions have been made, the user must recompile the I²C 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 I2C bus Acknowledge condition. |
| Include: | sw_i2c.h |
| Prototype: | unsigned char SWAckI2C( void ); unsigned char SWNotAckI2C( void ); |
| Remarks: | This function is called to generate an I2C 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 I2C bus. |
| Include: | sw_i2c.h |
| Prototype: | unsigned char SWGetcI2C( unsigned char *rdptr, unsigned char length ); |
| Arguments: | rdptr Location to store the data read from the I2C bus. \( \text{length} \) 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 |
| Code Example: | char x[10]; SWGetcI2C( x, 5 ); |
**MPLAB® C18 C Compiler Libraries**

### SWNotAckI2C

See SWAckI2C.

### SWPutcl2C

See SWWriteI2C.

### SWPutsI2C

**Function:** Write a string to the I²C bus.

**Include:** `sw_i2c.h`

**Prototype:**

```c
unsigned char SWPutsI2C( unsigned char *wrdptr );
```

**Arguments:**

- `wrdptr`: Pointer to data to be written to the I²C 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 I²C bus

0 otherwise

**File Name:** `swptsi2c.c`

**Code Example:**

```c
char mybuff [20];
SWPutsI2C(mybuff);
```

### SWReadI2C

**Function:** Read a byte from the I²C bus.

**Include:** `sw_i2c.h`

**Prototype:**

```c
unsigned char SWReadI2C( void );
```

**Remarks:**

This function reads in a single data byte by generating the appropriate signals on the predefined I²C clock line.

**Return Value:**

This function returns the acquired I²C data byte.

-1 if there was an error in this function.

**File Name:** `swgtci2c.c`

### SWRestartI2C

**Function:** Generate an I²C Restart bus condition.

**Include:** `sw_i2c.h`

**Prototype:**

```c
void SWRestartI2C( void );
```

**Remarks:**

This function is called to generate an I²C 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.

```c
#include <p18cxxxx.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(0x10); // 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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClearSWCSSPI</td>
<td>Clear the chip select (CS) pin.</td>
</tr>
<tr>
<td>OpenSWSPI</td>
<td>Configure the I/O pins for use as an SPI.</td>
</tr>
<tr>
<td>putcSWSPI</td>
<td>Write a byte of data to the software SPI.</td>
</tr>
<tr>
<td>SetSWCSSPI</td>
<td>Set the chip select (CS) pin.</td>
</tr>
<tr>
<td>WriteSWSPI</td>
<td>Write a byte of data to the software SPI bus.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>LCD Controller Line</th>
<th>Macros</th>
<th>Default Value</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS Pin</td>
<td>SW_CS_PIN</td>
<td>PORTBbits.RB2</td>
<td>Pin used for the chip select (CS) line.</td>
</tr>
<tr>
<td></td>
<td>TRIS_SW_CS_PIN</td>
<td>TRISBbits.TRISB2</td>
<td>Bit that controls the direction of the pin associated with the CS line.</td>
</tr>
<tr>
<td>DIN Pin</td>
<td>SW_DIN_PIN</td>
<td>PORTBbits.RB3</td>
<td>Pin used for the DIN line.</td>
</tr>
<tr>
<td></td>
<td>TRIS_SW_DIN_PIN</td>
<td>TRISBbits.TRISB3</td>
<td>Bit that controls the direction of the pin associated with the DIN line.</td>
</tr>
<tr>
<td>DOUT Pin</td>
<td>SW_DOUT_PIN</td>
<td>PORTBbits.RB7</td>
<td>Pin used for the DOUT line.</td>
</tr>
<tr>
<td></td>
<td>TRIS_SW_DOUT_PIN</td>
<td>TRISBbits.TRISB7</td>
<td>Bit that controls the direction of the pin associated with the DOUT line.</td>
</tr>
<tr>
<td>SCK Pin</td>
<td>SW_SCK_PIN</td>
<td>PORTBbits.RB6</td>
<td>Pin used for the SCK line.</td>
</tr>
<tr>
<td></td>
<td>TRIS_SW_SCK_PIN</td>
<td>TRISBbits.TRISB6</td>
<td>Bit that controls the direction of the pin associated with the SCK line.</td>
</tr>
</tbody>
</table>

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.
TABLE 3-10:  MACROS FOR SELECTING MODES

<table>
<thead>
<tr>
<th>Macro</th>
<th>Default Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODE0</td>
<td>defined</td>
<td>CKP = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CKE = 0</td>
</tr>
<tr>
<td>MODE1</td>
<td>not defined</td>
<td>CKP = 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CKE = 0</td>
</tr>
<tr>
<td>MODE2</td>
<td>not defined</td>
<td>CKP = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CKE = 1</td>
</tr>
<tr>
<td>MODE3</td>
<td>not defined</td>
<td>CKP = 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CKE = 1</td>
</tr>
</tbody>
</table>

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 (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 (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 output state and logic level.

File Name: opensspi.c

putcSWSPI

See WriteSWSPI.

SetSWCSSPI

Function: Set the chip select (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 (CS) pin for the software SPI.

File Name: setcsspi.c
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:

**TABLE 3-11: SOFTWARE UART FUNCTIONS**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getcUART</td>
<td>Read a byte from the software UART.</td>
</tr>
<tr>
<td>getsUART</td>
<td>Read a string from the software UART.</td>
</tr>
<tr>
<td>OpenUART</td>
<td>Configure I/O pins for use as a UART.</td>
</tr>
<tr>
<td>putcUART</td>
<td>Write a byte to the software UART.</td>
</tr>
<tr>
<td>putsUART</td>
<td>Write a string to the software UART.</td>
</tr>
<tr>
<td>ReadUART</td>
<td>Read a byte from the software UART.</td>
</tr>
<tr>
<td>WriteUART</td>
<td>Write a byte to the software UART.</td>
</tr>
</tbody>
</table>

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/traditional/pmc/sw_uart or scr/extended/pmc/sw_uart subdirectory of the compiler installation:

**TABLE 3-12: MACROS FOR SELECTING UART PIN ASSIGNMENTS**

<table>
<thead>
<tr>
<th>LCD Controller Line</th>
<th>Definition</th>
<th>Default Value</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>TX Pin</td>
<td>SWTXD</td>
<td>PORTB</td>
<td>Port used for the transmit line.</td>
</tr>
<tr>
<td></td>
<td>SWTXDpin</td>
<td>4</td>
<td>Bit in the SWTXD port used for the TX line.</td>
</tr>
<tr>
<td></td>
<td>TRIS_SWTXD</td>
<td>TRISB</td>
<td>Data Direction register associated with the port used for the TX line.</td>
</tr>
<tr>
<td>RX Pin</td>
<td>SWRXD</td>
<td>PORTB</td>
<td>Port used for the receive line.</td>
</tr>
<tr>
<td></td>
<td>SWRXDpin</td>
<td>5</td>
<td>Bit in the SWRXD port used for the RX line.</td>
</tr>
<tr>
<td></td>
<td>TRIS_SWRXD</td>
<td>TRISB</td>
<td>Data Direction register associated with the port used for the RX line.</td>
</tr>
</tbody>
</table>

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

**TABLE 3-13: SOFTWARE UART DELAY FUNCTIONS**

<table>
<thead>
<tr>
<th>Function</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>DelayTXBitUART</td>
<td>Delay for: (((2\times F_{osc}) / (4\times \text{baud}) + 1) / 2) - 12 cycles</td>
</tr>
<tr>
<td>DelayRXHalfBitUART</td>
<td>Delay for: (((2\times F_{osc}) / (8\times \text{baud}) + 1) / 2) - 9 cycles</td>
</tr>
<tr>
<td>DelayRXBitUART</td>
<td>Delay for: (((2\times F_{osc}) / (4\times \text{baud}) + 1) / 2) - 14 cycles</td>
</tr>
</tbody>
</table>
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 output 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

<table>
<thead>
<tr>
<th>Function</th>
<th>Read a byte from the software UART.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td><code>sw_uart.h</code></td>
</tr>
<tr>
<td>Prototype</td>
<td><code>char ReadUART( void );</code></td>
</tr>
<tr>
<td>Remarks</td>
<td>This function reads a byte of data out the software UART.</td>
</tr>
<tr>
<td>Return Value</td>
<td>Returns the byte of data that was read from the receive data (RXD) pin of the software UART.</td>
</tr>
<tr>
<td>File Name</td>
<td>readuart.asm</td>
</tr>
<tr>
<td>Code Example</td>
<td><code>char x;</code></td>
</tr>
<tr>
<td></td>
<td><code>x = ReadUART();</code></td>
</tr>
</tbody>
</table>

### WriteUART
#### putcUART

<table>
<thead>
<tr>
<th>Function</th>
<th>Write a byte to the software UART.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td><code>sw_uart.h</code></td>
</tr>
<tr>
<td>Prototype</td>
<td><code>void WriteUART( char data );</code></td>
</tr>
<tr>
<td>Arguments</td>
<td><code>data</code></td>
</tr>
<tr>
<td>Remarks</td>
<td>Byte of data to be written to software UART.</td>
</tr>
<tr>
<td></td>
<td>This function writes the specified byte of data out the software UART.</td>
</tr>
<tr>
<td>File Name</td>
<td>writuart.asm</td>
</tr>
<tr>
<td>Code Example</td>
<td><code>char x = 'H';</code></td>
</tr>
<tr>
<td></td>
<td><code>WriteUART( x );</code></td>
</tr>
</tbody>
</table>

#### 3.6.2 Example of Use

```c
#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
    }
}
```

Chapter 4. General Software Library

4.1 INTRODUCTION

This chapter documents general software library functions found in the precompiled standard C library file. The source code for all of these functions is included with MPLAB C18 in the following subdirectories of the compiler installation:
- src\traditional\stdlib
- src\extended\stdlib
- src\traditional\delays
- src\extended\delays

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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isalnum</td>
<td>Determine if a character is alphanumeric.</td>
</tr>
<tr>
<td>isalpha</td>
<td>Determine if a character is alphabetic.</td>
</tr>
<tr>
<td>iscntrl</td>
<td>Determine if a character is a control character.</td>
</tr>
<tr>
<td>isdigit</td>
<td>Determine if a character is a decimal digit.</td>
</tr>
<tr>
<td>isgraph</td>
<td>Determine if a character is a graphical character.</td>
</tr>
<tr>
<td>islower</td>
<td>Determine if a character is a lower case alphabetic character.</td>
</tr>
<tr>
<td>isprint</td>
<td>Determine if a character is a printable character.</td>
</tr>
<tr>
<td>ispunct</td>
<td>Determine if a character is a punctuation character.</td>
</tr>
<tr>
<td>isspace</td>
<td>Determine if a character is a white space character.</td>
</tr>
<tr>
<td>isupper</td>
<td>Determine if a character is an upper case alphabetic character.</td>
</tr>
<tr>
<td>isxdigit</td>
<td>Determine if a character is a hexadecimal digit.</td>
</tr>
</tbody>
</table>
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:**

```c
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:**

```c
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:**

```c
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:**

```c
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:**

```c
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:**

```c
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('	'), 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:**

```c
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:**

```c
unsigned char isxdigit( unsigned char ch );
```

**Arguments:**

- `ch`  
  Character to be checked.

**Remarks:** A character is considered to be a hexadecimal 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 hexadecimal 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 functions provided are:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>atob</td>
<td>Convert a string to an 8-bit signed byte.</td>
</tr>
<tr>
<td>atof</td>
<td>Convert a string into a floating point value.</td>
</tr>
<tr>
<td>atoi</td>
<td>Convert a string to a 16-bit signed integer.</td>
</tr>
<tr>
<td>atol</td>
<td>Convert a string into a long integer representation.</td>
</tr>
<tr>
<td>btoa</td>
<td>Convert an 8-bit signed byte to a string.</td>
</tr>
<tr>
<td>atoi</td>
<td>Convert a 16-bit signed integer to a string.</td>
</tr>
<tr>
<td>ltoa</td>
<td>Convert a signed long integer to a string.</td>
</tr>
<tr>
<td>rand</td>
<td>Generate a pseudo-random integer.</td>
</tr>
<tr>
<td>srand</td>
<td>Set the starting seed for the pseudo-random number generator.</td>
</tr>
<tr>
<td>tolower</td>
<td>Convert a character to a lower case alphabetical ASCII character.</td>
</tr>
<tr>
<td>toupper</td>
<td>Convert a character to an upper case alphabetical ASCII character.</td>
</tr>
<tr>
<td>ultoa</td>
<td>Convert an unsigned long integer to a string.</td>
</tr>
</tbody>
</table>

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
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
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:**
```c
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:**
```c
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:**
```c
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 an 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 an 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 an 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 pseudo-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:**
cctype.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:**
cctype.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:

### TABLE 4-3: MEMORY AND STRING MANIPULATION FUNCTIONS

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>memchr</code></td>
<td>Search for a value in a specified memory region.</td>
</tr>
<tr>
<td><code>memcmp</code></td>
<td>Compare the contents of two arrays.</td>
</tr>
<tr>
<td><code>memcmp</code></td>
<td></td>
</tr>
<tr>
<td><code>memcmppgm</code></td>
<td></td>
</tr>
<tr>
<td><code>memcmppgm2ram</code></td>
<td></td>
</tr>
<tr>
<td><code>memcmpram2pgm</code></td>
<td></td>
</tr>
<tr>
<td><code>memcpy</code></td>
<td>Copy a buffer from data or program memory into data memory.</td>
</tr>
<tr>
<td><code>memcpypgm2ram</code></td>
<td></td>
</tr>
<tr>
<td><code>memmove</code></td>
<td>Copy a buffer from data or program memory into data memory.</td>
</tr>
<tr>
<td><code>memmovepgm2ram</code></td>
<td></td>
</tr>
<tr>
<td><code>memset</code></td>
<td>Initialize an array with a single repeated value.</td>
</tr>
<tr>
<td><code>memset</code></td>
<td></td>
</tr>
<tr>
<td><code>strcat</code></td>
<td>Append a copy of the source string to the end of the destination string.</td>
</tr>
<tr>
<td><code>strcatpgm2ram</code></td>
<td></td>
</tr>
<tr>
<td><code>strchr</code></td>
<td>Locate the first occurrence of a value in a string.</td>
</tr>
<tr>
<td><code>strcmp</code></td>
<td>Compare two strings.</td>
</tr>
<tr>
<td><code>strcmppgm2ram</code></td>
<td></td>
</tr>
<tr>
<td><code>strcpy</code></td>
<td>Copy a string from data or program memory into data memory.</td>
</tr>
<tr>
<td><code>strcpypgm2ram</code></td>
<td></td>
</tr>
<tr>
<td><code>strcspn</code></td>
<td>Calculate the number of consecutive characters at the beginning of a string that are not contained in a set of characters.</td>
</tr>
<tr>
<td><code>strlen</code></td>
<td>Determine the length of a string.</td>
</tr>
<tr>
<td><code>strlwr</code></td>
<td>Convert all upper case characters in a string to lower case.</td>
</tr>
<tr>
<td><code>strncat</code></td>
<td>Append a specified number of characters from the source string to the end of the destination string.</td>
</tr>
<tr>
<td><code>strncatpgm2ram</code></td>
<td></td>
</tr>
<tr>
<td><code>strncpy</code></td>
<td>Copy characters from the source string into the destination string, up to the specified number of characters.</td>
</tr>
<tr>
<td><code>strncpypgm2ram</code></td>
<td></td>
</tr>
<tr>
<td><code>strpbrk</code></td>
<td>Search a string for the first occurrence of a character from a set of characters.</td>
</tr>
<tr>
<td><code>strrchr</code></td>
<td>Locate the last occurrence of a specified character in a string.</td>
</tr>
<tr>
<td><code>strspn</code></td>
<td>Calculate the number of consecutive characters at the beginning of a string that are contained in a set of characters.</td>
</tr>
<tr>
<td><code>strstr</code></td>
<td>Locate the first occurrence of a string inside another string.</td>
</tr>
<tr>
<td><code>strtok</code></td>
<td>Break a string into substrings, or tokens, by inserting null characters in place of specified delimiters.</td>
</tr>
<tr>
<td><code>strupr</code></td>
<td>Convert all lower case characters in a string to upper case.</td>
</tr>
</tbody>
</table>
### Function Descriptions

#### memchr

**Function:** Locate the first occurrence of a byte value in a specified memory region.

**Include:** `string.h`

**Prototype:**
```c
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

**Function:** Compare the contents of two arrays of bytes.

**Include:** `string.h`

**Prototype:**
```c
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 );
```

**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 \textit{memsize} number of bytes in \textit{buf1} to
the first \textit{memsize} number of bytes in \textit{buf2} and returns a value
indicating whether the buffers are less than, equal to or greater than
each other.

Return Value:
\texttt{memcmp} returns a value that is:
\begin{itemize}
\item <0 if \textit{buf1} is less than \textit{buf2}
\item ==0 if \textit{buf1} is the same as \textit{buf2}
\item >0 if \textit{buf1} is greater than \textit{buf2}
\end{itemize}

File Names:
\begin{itemize}
\item \texttt{memcmp.asm}
\item \texttt{memcmpp2p.asm}
\item \texttt{memcmpp2r.asm}
\item \texttt{memcmpr2p.asm}
\end{itemize}

\begin{verbatim}
memcpypgm2ram

Function: Copy the contents of the source buffer into the destination buffer.
Include: \texttt{string.h}
Prototype:
void * memcpypgm2ram(
    void * dest,
    const rom void * src,
    sizeram_t memsize
);

Arguments:
\begin{itemize}
\item \texttt{dest}
    Pointer to destination array.
\item \texttt{src}
    Pointer to source array.
\item \texttt{memsize}
    Number of bytes of \texttt{src} array to copy into \texttt{dest}.
\end{itemize}

Remarks: This function copies the first \textit{memsize} number of bytes in \texttt{src} to the
array \texttt{dest}. If \texttt{src} and \texttt{dest} overlap, the behavior is undefined.

Return Value: This function returns the value of \texttt{dest}.
File Names:
\begin{itemize}
\item \texttt{memcpy.asm}
\item \texttt{memcpyp2r.asm}
\end{itemize}
\end{verbatim}
## memmove

### Function:
Copy the contents of the source buffer into the destination buffer, even if the regions overlap.

### Include:
string.h

### Prototype:
```c
void * memmove( void * dest,
        const void * src,
        size_t memsize );
```

```c
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:
```c
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 function 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:**

```c
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:**

```c
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:**
```c
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:**
```c
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:

<table>
<thead>
<tr>
<th>str1</th>
<th>str2</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;hello&quot;</td>
<td>&quot;aeiou&quot;</td>
<td>1</td>
</tr>
<tr>
<td>&quot;antelope&quot;</td>
<td>&quot;aeiou&quot;</td>
<td>0</td>
</tr>
<tr>
<td>&quot;antelope&quot;</td>
<td>&quot;xyz&quot;</td>
<td>8</td>
</tr>
</tbody>
</table>

**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
Point to destination array.

src
Point 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
strncatp2r.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.

Return Value: \( \text{strncmp} \) 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

#### Function:
Copy characters from the source string into the destination string, up to the specified number of characters.

#### Include:
string.h

#### Prototype:
```c
char * strncpy( char * dest, const char * src, size_t n );
```
```c
char *strncpypgm2ram( char * dest, const rom char * src, sizerram_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 Names:
strncpy.asm
sncpypgm2ram.asm

### strpbrk

#### Function:
Search a string for the first occurrence of a character from a specified set of characters.

#### Include:
string.h

#### Prototype:
```c
char * strpbrk( const char * 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:**

```c
#include <string.h>
```

**Prototype:**

```c
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:**

```c
#include <string.h>
```

**Prototype:**

```c
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.

**Remarks:**

This function will determine the number of consecutive characters from the beginning of `str1` that are contained in `str2`. For example:

<table>
<thead>
<tr>
<th>str1</th>
<th>str2</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;banana&quot;</td>
<td>&quot;ab&quot;</td>
<td>2</td>
</tr>
<tr>
<td>&quot;banana&quot;</td>
<td>&quot;abn&quot;</td>
<td>6</td>
</tr>
<tr>
<td>&quot;banana&quot;</td>
<td>&quot;an&quot;</td>
<td>0</td>
</tr>
</tbody>
</table>

**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.

**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:**

```c
#include "string.h"
```

**Prototype:**

```c
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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay1TCY</td>
<td>Delay one instruction cycle.</td>
</tr>
<tr>
<td>Delay10TCYx</td>
<td>Delay in multiples of 10 instruction cycles.</td>
</tr>
<tr>
<td>Delay100TCYx</td>
<td>Delay in multiples of 100 instruction cycles.</td>
</tr>
<tr>
<td>Delay1KTCYx</td>
<td>Delay in multiples of 1,000 instruction cycles.</td>
</tr>
<tr>
<td>Delay10KTCYx</td>
<td>Delay in multiples of 10,000 instruction cycles.</td>
</tr>
</tbody>
</table>

### 4.5.1 Function Descriptions

**Delay1TCY**

**Function:** Delay 1 instruction cycle (TCY).

**Include:** delays.h

**Prototype:**

```c
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:**

```c
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:**

```c
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 2,560 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:**
```c
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:**
```c
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:** d10ktyx.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:

### TABLE 4-5: RESET FUNCTIONS

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isBOR</td>
<td>Determine if the cause of a Reset was the Brown-out Reset circuit.</td>
</tr>
<tr>
<td>isLVD</td>
<td>Determine if the cause of a Reset was a low voltage detect condition.</td>
</tr>
<tr>
<td>isMCLR</td>
<td>Determine if the cause of a Reset was the MCLR pin.</td>
</tr>
<tr>
<td>isPOR</td>
<td>Detect a Power-on Reset condition.</td>
</tr>
<tr>
<td>isWDTTO</td>
<td>Determine if the cause of a Reset was a Watchdog timer time-out.</td>
</tr>
<tr>
<td>isWDTWU</td>
<td>Determine if the cause of a wake-up was the Watchdog timer.</td>
</tr>
<tr>
<td>isWU</td>
<td>Detects if the microcontroller was just waken up from Sleep from the MCLR pin or an interrupt.</td>
</tr>
<tr>
<td>StatusReset</td>
<td>Set the POR and BOR bits.</td>
</tr>
</tbody>
</table>

**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:**

```c
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:

- `POR = 1`
- `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:**

```c
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 MCLR pin.

**Include:** reset.h

**Prototype:**

```c
char isMCLR( void );
```

**Remarks:** This function detects if the microcontroller was reset via the MCLR pin while in normal operation. This situation is indicated by the following Status bits:

- POR = 1
- If Brown-out is enabled, BOR = 1
- If WDT is enabled, TO = 1
- PD = 1

**Return Value:**

- 1 if the Reset was due to MCLR during normal operation
- 0 otherwise

**File Name:** ismclr.c

**isPOR**

**Function:** Detect a Power-on Reset condition.

**Include:** reset.h

**Prototype:**

```c
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:

- POR = 0
- BOR = 0
- TO = 1
- PD = 1

This condition also can occur for MCLR during normal operation and when the CLRWDT instruction is executed. After isPOR is called, StatusReset should be called to set the POR and 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) timeout.

**Include:** reset.h

**Prototype:**

```c
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:

- POR = 1
- BOR = 1
- TO = 0
- 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:
  POR = 1
  BOR = 1
  TO = 0
  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 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 MCLR pin or an interrupt. This condition is indicated by the following Status bits:
  POR = 1
  BOR = 1
  TO = 1
  PD = 0
Return Value: 1 if the device was brought out of Sleep by the MCLR pin or an interrupt
0 otherwise
File Name: iswu.c

StatusReset

Function: Set the POR and BOR bits in the CPUSTA register.
Include: reset.h
Prototype: void StatusReset( void );
Remarks: This function sets the POR and BOR bits in the CPUSTA register. These bits must be set in software after a Power-on Reset has occurred.
File Name: statrst.c
NOTES:
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™ (DS33014) 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 standard C library, which may be found in the lib subdirectory. The standard C library 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:

<table>
<thead>
<tr>
<th>Format</th>
<th>Exponent</th>
<th>Mantissa 0</th>
<th>Mantissa 1</th>
<th>Mantissa 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE-754</td>
<td>sxxxx</td>
<td>yxxxx</td>
<td>xxxx</td>
<td>xxxx xxxx</td>
</tr>
<tr>
<td>Microchip</td>
<td>xxxxx</td>
<td>xxxxx</td>
<td>xxxx xxxx</td>
<td>xxxx xxxx</td>
</tr>
</tbody>
</table>

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.
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:**

\[ \text{Exp} = \text{int}(Z) \]

where \( Z \) is the fractional exponent, \( A_{10} \) is the original decimal number, and \( \text{Exp} \) 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 \( \text{Exp} \).

Finally, a bias value of \( 0xFF \) is added to convert \( \text{Exp} \) to Microchip floating point format (\( \text{Exp}_{\text{MFP}} \)).

\[ \text{Exp}_{\text{MFP}} = \text{Exp} + 0xFF \]

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:

$$x = 1.25 \geq 2^0? \quad Yes \quad bit = 1; \quad x = 1.25 - 1 = 0.25$$

$$x = 0.25 \geq 2^{-1}? \quad No \quad bit = 0; \quad x = 0.25$$

$$x = 0.25 \geq 2^{-2}? \quad Yes \quad bit = 1; \quad x = 0.25 - 0.25 = 0$$

$$x = 0$$

Process complete

Therefore, the binary representation is:

$$A_2 = 1.010000000000000000000000$$

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 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.

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 $2^0$. 

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 $\text{Exp} = 0x84 - 0x7F = 0x05$. To process the mantissa, it is first translated to binary format and the MSb is set to prepare for the expansion.

$0x490FDB = \begin{array}{c} 0100 \\ 1100 \\ 1101 \\ 1101 \\ 1111 \\ 1101 \\ 1011 \\ \end{array}_2 \rightarrow \begin{array}{c} 0100 \\ 1001 \\ 0000 \\ 1111 \\ 1111 \\ 1101 \\ 1011 \\ \end{array}_2$

The expansion is then performed according to Equation 5-5.

$$A_2 = 2^0 + 2^{-1} + 2^{-4} + 2^{-7} + 2^{-12} + 2^{-13} + 2^{-15} + 2^{-16} + 2^{-17} + 2^{-19} + 2^{-20} + 2^{-22} + 2^{-23}$$

$$A_2 = 1.570796371$$

Finally, to calculate the actual floating point number, the exponent and expanded mantissa are plugged into the conversion formula (Equation 5-4).

$$A_{10} = 2^5 \cdot 1.570796371$$

$$A_{10} = 50.26548387$$

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.
Glossary

A

Absolute Section
A section with a fixed address that cannot be changed by the linker.

Access Memory
Special General Purpose Registers (GPR) 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 compiler.

Extended Mode
In Extended mode, the compiler will utilize the extended instructions (i.e., ADDFSR, ADDULNK, CALLW, MOVSF, MOVSS, PUSHL, SUBFSR and SUBULNK) and the indexed with literal offset addressing.

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>.

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.

N
Non-extended Mode
In Non-extended mode, the compiler will not utilize the extended instructions nor the indexed with literal offset addressing.
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.

Pragma
A directive that has meaning to a specific compiler.

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).

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.

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

## A
- A/D Converter ............................................................... 11
- Busy ........................................................................ 12
- Close ....................................................................... 12
- Convert ..................................................................... 12
- Example of Use .......................................................... 18
- Open ........................................................................ 12, 14, 16
- Read ........................................................................ 17
- Set Channel ............................................................... 18
- Ack2I2C ..................................................................... 23
- Alphabetical Character ............................................... 108
- Alphanumeric Character ............................................. 108
- ANSI ......................................................................... 7
- Asynchronous Mode ....................................................... 59
- atof .......................................................................... 112
- atoi .......................................................................... 113
- atol .......................................................................... 113

## B
- baudUSART ................................................................. 63
- Brown-out Reset ............................................................. 131
- btoa ......................................................................... 113
- build.bat ..................................................................... 8
- BusyADC ..................................................................... 12
- BusyUSART ................................................................ 57
- BusyXLCD .................................................................. 67

## C
- c018.o ..................................................................... 7
- c018.o ................................................................. 7
- c018i.o ..................................................................... 7
- c018i.o ................................................................. 7
- c018iz.o ................................................................. 7
- c018iz.o ................................................................. 7
- CAN2510, External ......................................................... 72
- Bit Modify .................................................................. 73
- Byte Read .................................................................. 74
- Byte Write .................................................................. 74
- Data Read .................................................................. 74
- Data Ready .................................................................. 75
- Disable ...................................................................... 76
- Enable ...................................................................... 76
- Error State ................................................................. 77
- Initialize ................................................................... 77
- Interrupt Enable ......................................................... 81
- Interrupt Status ......................................................... 82
- Load Extended to Buffer ............................................... 83
- Load Extended to RTR .................................................. 84
- Load Standard to Buffer ............................................... 82
- Load Standard to RTR .................................................. 84
- Read Mode .................................................................. 85
- Read Status .................................................................. 85
- Reset .......................................................................... 86
- Send Buffer .................................................................. 86
- Sequential Read ......................................................... 86
- Sequential Write ......................................................... 87
- Set Buffer Priority ......................................................... 87
- Set Message Filter to Extended ...................................... 89
- Set Message Filter to Standard ....................................... 88
- Set Mode ...................................................................... 88
- Set Single Filter to Extended .......................................... 90
- Set Single Filter to Standard .......................................... 90
- Set Single Mask to Extended ......................................... 91
- Set Single Mask to Standard ......................................... 91
- Write Extended Message ............................................... 93
- Write Standard Message ............................................... 92

- CAN2510BitModify ....................................................... 73
- CAN2510ByteRead ....................................................... 74
- CAN2510ByteWrite ....................................................... 74
- CAN2510DataRead ....................................................... 74
- CAN2510DataReady ...................................................... 75
- CAN2510Disable ......................................................... 76
- CAN2510Enable ......................................................... 76
- CAN2510ErrorState ....................................................... 77
- CAN2510Init ............................................................... 77
- CAN2510InterruptEnable ............................................... 81
- CAN2510InterruptStatus ............................................... 82
- CAN2510LoadBufferStd ................................................ 82
- CAN2510LoadBufferXtd ................................................ 83
- CAN2510LoadRTRStd .................................................... 84
- CAN2510LoadRTRXtd .................................................... 84
- CAN2510ReadMode ....................................................... 85
- CAN2510ReadStatus ....................................................... 85
- CAN2510Reset ............................................................ 86
- CAN2510SendBuffer ..................................................... 86
- CAN2510SequentialRead ............................................... 86
- CAN2510SequentialWrite .............................................. 87
- CAN2510SetBufferPriority ............................................. 87
- CAN2510SetMode ......................................................... 88
- CAN2510SetMsgFilterStd ............................................... 88
- CAN2510SetMsgFilterXtd ............................................... 89
- CAN2510SetSingleFilterStd ............................................. 90
- CAN2510SetSingleFilterXtd ............................................. 90
- CAN2510SetSingleMaskStd ............................................. 91
- CAN2510SetSingleMaskXtd ............................................. 91
- CAN2510WriteStd ......................................................... 92
- CAN2510WriteXtd ......................................................... 93
MPLAB® C18 C Compiler Libraries

Capture .................................................. 19-20
Close ......................................................... 19
Example of Use ........................................ 22
Open ......................................................... 20
Read ......................................................... 21

Character Classification
- Alphabetic ........................................ 108
- Alphanumeric ................................. 108
- Control ............................................. 108
- Decimal ............................................ 109
- Graphical ........................................... 109
- Hexadecimal ..................................... 111
- Lower Case Alphabetic .................. 109
- Printable ........................................... 110
- Punctuation ...................................... 110
- Upper Case Alphabetic .................. 111
- White Space ...................................... 110

Character Classification Functions ................. 107
ClearSWCSSPI ........................................... 101
clib.lib ................................................. 8
clib_e.lib .............................................. 8
Clock_test ............................................ 95
CloseADC .............................................. 12
CloseADC .............................................. 12
CloseCapture ........................................ 19
CloseECapture ....................................... 19
Close2C .................................................. 24
CloseMwire ............................................ 34
ClosePORTB .......................................... 32
ClosePWM ............................................. 39
CloseRBxNT .......................................... 32
CloseSPI ............................................... 42
CloseTimer ........................................... 48
CloseUSB ............................................... 57
Control Character ................................ 108
ConvertADC ......................................... 12

D
Data Conversion Functions .......................... 112
- Byte to String .................................. 113
- Convert Character to Lower Case ........ 115
- Convert Character to Upper Case ..... 115
- Integer to String ................................ 114
- Long to String .................................. 114
- String to Byte .................................. 112
- String to Float .................................. 112
- String to Integer .............................. 113
- String to Long .................................. 113
- Unsigned Long to String .................. 116

Data Initialization .................................. 7
DataRdyMwire ........................................ 34
DataRdySPI ........................................... 42
DataRdyUSART ...................................... 58
Delay .................................................... 129
  1 Tcy .............................................. 129
  1,000 Tcy Multiples ......................... 130
  10 Tcy Multiples ............................ 129
  10,000 Tcy Multiples ................... 130
  100 Tcy Multiples ......................... 129
Delay100TCYx ....................................... 129
Delay1KTCYx ........................................ 130
Delay1TCY ............................................ 129

Delay1KTCYx ........................................ 130
Delay1TCY ............................................ 129

Directories
- h ................................................. 65, 94, 100
- lib .............................................. 7-8, 135
- math ............................................. 135
- pmc .............................................. 11, 65
- src ................................................. 7
- startup .......................................... 8

DisablePullups ..................................... 32
Documentation Conventions ....................... 3

E
ECapture
  Close ............................................. 19
  Open ............................................. 20

EE Memory Device Interface Functions ............ 28
EEAckPolling ........................................ 28
EEByteWrite ........................................ 28
EECurrentAddRead ................................ 29
EEPPageWrite ...................................... 29
EERandomRead ..................................... 30
EESequentialRead ................................ 30

EnablePullups ..................................... 33

Examples
- A/D Converter ................................... 18
- Capture .......................................... 22
- I²C, Hardware .................................. 31
- I²C, Software ................................... 98
- LCD ............................................. 71
- Microwire ...................................... 37
- SPI, Hardware .................................. 45
- SPI, Software ................................... 102
- Timers .......................................... 55
- UART, Software ................................ 105
- USART, Hardware ............................. 64

Exponent .......................................... 135, 137-138

F
Floating Point
  Conversion ....................................... 136
  Libraries ......................................... 135
  Representation .................................. 135
  FPREP ............................................. 136

G
getcIC .................................................. 24
getcMwire ........................................... 35
getcSPI ............................................... 42
getcUART .......................................... 104
getcUSART ......................................... 58
geticsIC .............................................. 24
getcMicrowire ..................................... 35
getcSPI ............................................... 43
getcUART .......................................... 104
getcUSART ......................................... 58

Graphical Character ................................ 109

H
h directory ....................................... 65, 94, 100
Index

I
I/O Port ........................................... 32
I2C, Hardware .................................. 23

Acknowledge .................................. 23
Close ............................................ 24
EEPROM Acknowledge Polling .......... 28
EEPROM Byte Write ..................... 28
EEPROM Current Address Read ...... 29
EEPROM Page Write .................... 29
EEPROM Random Read ............... 30
EEPROM Sequential Read .......... 30
Example of Use ......................... 31
Get Character .......................... 24
Get String .................................. 24
Idle ............................................. 25
No Acknowledge ...................... 25
Open ........................................... 25
Put Character ........................... 25
Put String ................................... 26
Read ........................................... 26
Restart ....................................... 26
Start .......................................... 27
Stop ............................................. 27
Write ........................................... 27

I2C, Software ............................... 94
Acknowledge ................................ 95
Clock Test .................................. 95
Example of Use ......................... 98
Get Character .......................... 95
Get String .................................. 95
No Acknowledge ...................... 95-96
Put Character ......................... 96
Put String .................................. 96
Read ........................................... 96
Restart ....................................... 96
Start .......................................... 97
Stop ............................................. 97
Write ........................................... 97

IdleI2C ........................................ 25
IEEE Floating Point Representation .... 135
Initialized Data ...................... .................
Interrupt Service Routine .......... 143
interrupt service routine ........ 143
isanum ........................................ 108
isalpha ...................................... 108
isB0R .......................................... 131
iscntrl ....................................... 108
isdigit ........................................ 109
isgraph ...................................... 109
islower ....................................... 109
islV0D ........................................ 131
isMCLR ...................................... 132
isPOV .......................................... 132
isprint ....................................... 110
ispunct ....................................... 110
isspace ....................................... 110
isupper ....................................... 111
isWDIT0 ..................................... 132
isWDTWU .................................... 133
isWU .......................................... 133

isxdigit ....................................... 111
itoa ............................................ 114

L
LCD, External .................................. 65
Busy ........................................... 67
Example of Use ......................... 71
Open ........................................... 67
Put Character ........................... 67, 70
Put ROM String ....................... 68
Put String ................................... 68
Read Address ............................. 68
Read Data ................................... 69
Set Character Generator Address ...... 69
Set Display Data Address ........ 69
Write Command ....................... 70
Write Data ................................... 70

lib directory ................................. 7-8, 135

Libraries
Processor-Independent ............... 8
Processor-Specific ................... 9
Rebuilding ................................ 7-9
Source Code ......................... 8-9

Library Overview ..................... 7
Little Endian ............................. 143
Lower Case Characters ........ 109, 115, 123

itoa ............................................ 114

M
main ............................................. 7
makeclib.bat .................................. 8
makeplib.bat ............................. 9
Maintissa ................................. 135, 137-138
Math Directory ......................... 135
MCLR ........................................... 132
memchr ...................................... 118
memcmp ...................................... 118
memcmpbgm ......................... 118
memcmpbgm2ram .................. 118
memcmpbram2pgm ........... 118
memcpy ....................................... 119
memcppg2ram .................. 119
memmove .................................... 120
memovepgm2ram .................. 120
Memory Manipulation Functions .... 117

Compare .................................. 118
Copy .......................................... 119
Move ......................................... 120
Search ....................................... 118
Set ............................................ 120

memset ...................................... 120
Microchip Web Site ................ 4

Microwire ...................................... 34

Close ........................................... 34
Data Ready ......................... 34
Example of Use .................. 37
Get Character ......................... 35
Get String .................................. 35
Open ........................................... 35
Put Character ......................... 35
Read ........................................... 36

© 2004 Microchip Technology Inc.
MPLAB® C18 C Compiler Libraries

N
NotAckI2C ................................................................. 25

O
OpenADC ................................................................. 12, 14, 16
OpenCapture .............................................................. 20
OpenECapture ............................................................ 20
OpenI2C ................................................................. 25
OpenMwire ................................................................. 35
OpenPORTB ............................................................... 33
OpenPWM ................................................................. 39
OpenRXtxNT .............................................................. 33
OpenSPI ................................................................. 43
OpenSWSWPI ............................................................. 101
OpenTimer ................................................................. 48-52
OpenUART ................................................................. 104
OpenUSART ............................................................... 59
OpenXLCUSD .............................................................. 67

P
Peripheral Libraries ...................................................... 9
pmc directory ............................................................ 11, 65
PORTB
Close ................................................................. 32
Disable Intearupts ..................................................... 32
Disable Pullups ......................................................... 32
Enable Inteerrupts .................................................... 33
Enable Pullups ......................................................... 33
Open ................................................................. 33
Pulse-Width Modulation Functions ................................ 39
putcI2C ................................................................. 25
putcMwire ............................................................... 35
putcSPI ................................................................. 43
putcSWSWPI ............................................................ 101
putcUART ................................................................. 104
putcUSART ............................................................... 60
putcXLCUSD ............................................................. 67, 70
putsUSART ............................................................... 60
putsXLCUSD ............................................................. 68
putsI2C ................................................................. 26
putsSPI ................................................................. 44
putsUART ................................................................. 104
putsUSART ............................................................... 60
putsXLCUSD ............................................................. 68

PWM
Close ................................................................. 39
Open ................................................................. 39
Set Duty Cycle .......................................................... 40
Set ECCP Output ........................................................ 41

R
rand ................................................................. 114
ReadADC ............................................................... 17
ReadAddrXLCUSD ..................................................... 68
ReadCapture ........................................................... 21
ReadDataXLCUSD ..................................................... 69
ReadI2C ................................................................. 26
ReadMwire ............................................................... 36
ReadSPI ................................................................. 44
ReadTimer ............................................................... 53
ReadUART ............................................................... 105
ReadUSART ............................................................. 61
References ............................................................. 3
Reset Functions ......................................................... 131
   Brown-out .......................................................... 131
   Low Voltage Detect .............................................. 131
   Master Clear ......................................................... 132
   Power-on ............................................................ 132
   Status ................................................................. 133
   Wake-up ............................................................. 133
   Watchdog Timer Time-out ...................................... 132
   Watchdog Timer Wake-up ....................................... 133
RestartI2C ............................................................. 26

S
SetCGRamAddr .......................................................... 69
SetChanADC ........................................................... 18
SetDCPWM ............................................................. 40
SetDDRamAddr .......................................................... 69
SetOutputPWM .......................................................... 41
SetSWCSSPI ............................................................ 101
SFR Definitions .......................................................... 9
Sleep ................................................................. 133
SPI, Hardware ............................................................ 42
   Close ................................................................. 42
   Data Ready .......................................................... 42
   Example of Use .................................................... 45
   Get Character ....................................................... 42
   Get String .......................................................... 43
   Open ................................................................. 43
   Put Character ....................................................... 43
   Put String .......................................................... 44
   Read ................................................................. 44
   Write ................................................................. 44
SPI, Software ............................................................ 100
   Clear Chip Select ................................................... 101
   Example of Use ..................................................... 102
   Open ................................................................. 101
   Put Character ....................................................... 101
   Set Chip Select ..................................................... 101
   Write ................................................................. 102
srand ................................................................. 115
src directory ........................................................... 7
SSP ................................................................. 23-24
Stack, Software .......................................................... 7
Standard C Library ..................................................... 8, 135
StartI2C ................................................................. 27
Startup Code ........................................................... 7
startup directory ........................................................ 8
StatusReset ........................................................... 133
StopI2C ................................................................. 27
strcat ................................................................. 121
strcatpgm2ram ......................................................... 121
strchr ................................................................. 121
strcmp ................................................................. 122
strcmpeqpgm2ram ..................................................... 122
strcpy ................................................................. 122
strncpypgm2ram ....................................................... 122
Index

strcsnp ................................................. 123
String Manipulation Functions .............. 117
Append .............................................. 121, 124
Compare ............................................. 122, 124
Convert ............................................. 123
Convert to Lower Case ......................... 128
Copy .................................................. 122, 125
Length .............................................. 123
Search ............................................. 121, 125-127
Tokenize .......................................... 127
strlen ............................................... 123
strchr ............................................... 125
strchr ................................................ 126
strcspn ........................................... 123
strcat .............................................. 124
strcatpgm2ram ................................... 124
strncat ........................................... 126
strncpy ............................................ 125
strncatpgm2ram ................................ 125
strncac ............................................ 126
strncat ........................................... 127
strtok ............................................. 127
strspn ............................................. 126
strstr .............................................. 127
strspn ............................................. 126
strrchr ............................................ 126
strpbrk ............................................ 125
strpcypgm2ram ................................. 125
strncpy ............................................ 125
strncpy ............................................ 125
strncpypgm2ram ................................. 125
strrchr ............................................ 126
strchr ............................................... 127
strchr ............................................... 127
SWAckI2C ....................................... 95-96
SWGetcl2C ........................................ 95
SWGetsl2C ........................................ 95
SWNotAckI2C .................................... 95
SWPutsI2C ........................................ 96
SWPutcI2C ........................................ 96
SWPuti2C ........................................ 96
SWReadI2C ....................................... 96
SWRestartI2C .................................... 96
SWStartI2C ....................................... 97
SWStopI2C ........................................ 97
SWWriteI2C ...................................... 97
Synchronous Mode ............................... 59

T
Timers ............................................. 48
Close ............................................. 48
Example of Use ................................. 55
Open .............................................. 48-52
Read .............................................. 53
Write ............................................. 54
tolower ............................................ 115
toupper ............................................ 115

U
UART, Software ................................. 103
Example of Use ................................. 105
Get Character ................................... 104
Get String ....................................... 104
Open .............................................. 104
Put Character ................................... 104
Put String ....................................... 104
Read .............................................. 105
Write ............................................. 105
ultoa .............................................. 116
Upper Case Characters ................. 111, 115, 123
USART, Hardware ............................. 56
baud .............................................. 63
Busy ............................................. 57

W
Watchdog Timer (WDT) ....................... 132-133
WriteCmdXLCD ................................. 70
WriteDataXLCD ................................. 70
WriteI2C ........................................ 27
WriteMwire ...................................... 36
WriteSPI ......................................... 44
WriteSWSPI ...................................... 102
WriteTimer ...................................... 54
WriteUART ...................................... 105
WriteUSART ................................. 62

 © 2004 Microchip Technology Inc.
AMERICAS
Corporate Office
2395 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

Atlanta
3780 Mansell Road, Suite 130
Alpharetta, GA 30022
Tel: 770-640-0034
Fax: 770-640-0070

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

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

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

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

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

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

San Jose
1300 Terra Bella Avenue
Mountain View, CA 94043
Tel: 650-215-1444
Fax: 650-961-0286

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

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

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

China - Chengdu
Rm. 2401-2402, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-88766200
Fax: 86-28-88766599

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

China - Hong Kong SAR
Unit 901-6, Tower 2, Metropolis
233 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2542-1200
Fax: 852-2401-3431

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

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

China - Shunde
Room 401, Hongjian Building, No. 2
Fengxiangnan Road, Ronggui Town, Shunde
District, Foshan City, Guangdong 528303, China
Tel: 86-757-28395507
Fax: 86-757-28395571

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

India
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-22290061
Fax: 91-80-22290062

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

Korea
188-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Singapore
200 Middle Road
#07-02 Prime Centre
Singapore, 189980
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan
Kaohsiung Branch
30F - 1 No. 8
Min Chuan 2nd Road
Kaohsiung 806, Taiwan
Tel: 866-7-536-4818
Fax: 866-7-536-4803

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

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

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

France
Parc d’Activite du Moulin de Massy
43 Rue du Saule Trapa
Batiment A - Ier Etage
91300 Massy, France
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-69-70

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

Italy
Via Quasimodo, 12
20025 Legnano (MI)
Tel: 39-0331-742611
Fax: 39-0331-467681

Netherlands
P. A. De Biesbosch 14
NL-5152 SC Drunen, Netherlands
Tel: 31-1-69-53-63-20
Fax: 33-1-69-30-90-79

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

02/17/04