



**MPLAB® C18
COMPILEADOR C
BIBLIOTECAS**

Tenga en cuenta los siguientes detalles de la función de protección de código en los dispositivos Microchip:

- Los productos de Microchip cumplen con la especificación contenida en su Hoja de Datos de Microchip particular.
- Microchip cree que su familia de productos es una de las familias más seguras de su tipo en el mercado actual, cuando se usa de la manera prevista y en condiciones normales.
- Se utilizan métodos deshonestos y posiblemente ilegales para infringir la función de protección del código. Todos estos métodos, a nuestro conocimiento, requieren el uso de los productos de Microchip de una manera fuera de las especificaciones de funcionamiento contenidas en las hojas de datos de Microchip. Lo más probable es que la persona que lo haga esté involucrada en el robo de propiedad intelectual.
- Microchip está dispuesto a trabajar con el cliente que se preocupa por la integridad de su código.
- Ni Microchip ni ningún otro fabricante de semiconductores puede garantizar la seguridad de su código. La protección del código no significa que estemos garantizando el producto como "irrompible".

Protección del código esta en constante evolución. En Microchip nos comprometemos a mejorar continuamente las funciones de protección de código de nuestros productos. Los intentos de romper la función de protección de código de Microchip pueden ser una violación de la Ley de derechos de autor del milenio digital. Si dichos actos permiten el acceso no autorizado a su software u otro trabajo protegido por derechos de autor, es posible que tenga derecho a demandar para obtener reparación en virtud de esa Ley.

La información contenida en esta publicación con respecto a las aplicaciones del dispositivo y similares se proporciona solo para su conveniencia y puede ser reemplazada por actualizaciones. Es su responsabilidad asegurarse de que su aplicación cumpla con sus especificaciones.
MICROCHIP NO HACE DECLARACIONES NI GARANTÍAS DE NINGÚN TIPO, YA SEA EXPRESA O IMPLÍCITA, ESCRITA U ORAL, LEGAL O DE OTRO MODO, EN RELACIÓN CON LA INFORMACIÓN, INCLUYENDO, ENTRE OTROS, SU CONDICIÓN, CALIDAD, RENDIMIENTO, COMERCIALIDAD O IDONEIDAD PARA EL FIN.

Microchip se exime de toda responsabilidad derivada de esta información y su uso. No se autoriza el uso de productos de Microchip como componentes críticos en sistemas de soporte vital, excepto con la aprobación expresa por escrito de Microchip. No se transmiten licencias, implícitamente o de otro modo, bajo ningún derecho de propiedad intelectual de Microchip.

Marcas registradas

El nombre y el logotipo de Microchip, el logotipo de Microchip, Accuron, dsPIC, KEELOQ, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, PowerSmart, rfPIC y SmartShunt son marcas comerciales registradas de Microchip Technology Incorporated en EE. UU. y otros países.

AmpLab, FilterLab, Migratable Memory, MXDEV, MXLAB, PICMASTER, SEEVAL, SmarISensor y The Embedded Control Solutions Company son marcas comerciales registradas de Microchip Technology Incorporated en EE. UU.

Analógico para la era digital, aplicación Maestro, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, programación en serie en circuito, ICSP, ICEPIC, termistor activo lineal, MPASM, MPLIB, MPLINK, MPSIM, PICkit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, rfLAB, rfPICDEM, Select Mode, Smart Serial, SmartTel, Total Endurance y WiperLock son marcas comerciales de Microchip Technology Incorporated en EE. UU. y otros países .

SQTP es una marca de servicio de Microchip Technology Incorporated en EE. UU.

Todas las demás marcas comerciales mencionadas en este documento son propiedad de sus respectivas empresas.

© 2005, Microchip Technology Incorporated, Impreso en los EE. UU., Todos los derechos reservados.



Impreso en papel reciclado.

Microchip recibió la certificación del sistema de calidad ISO/TS-16949:2002 para su sede mundial, instalaciones de diseño y fabricación debleas en Chandler y Tempe, Arizona y Mountain View, California en octubre de 2003. Los procesos y procedimientos del sistema de calidad de la empresa son para su PICmicro® 8-bit MCU, dispositivos de salto de código KEELOQ®, EEPROM seriales, microporífericos, memoria no volátil y productos analógicos. Además, el sistema de calidad de Microchip para el diseño y fabricación de sistemas de desarrollo cuenta con la certificación ISO 9001:2000.

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



COMPILEADOR MPLAB® C18 C BIBLIOTECAS

Tabla de contenido

Prefacio	1
Capítulo 1. Descripción general	
1.1 Introducción	5
1.2 Descripción general de las bibliotecas MPLAB C18	5
1.3 Código de inicio	7
1.4 Biblioteca independiente del procesador	7
1.5 Bibliotecas específicas del procesador	7
Capítulo 2. Funciones de los periféricos de hardware	
2.1 Introducción	9
2.2 Funciones del convertidor A/D	9
2.3 Funciones de captura de entrada I2C™	17
2.4 Funciones del puerto de E/S	34
2.5 Funciones del microhilos	37
2.6 Funciones de modulación de ancho de pulso SPI™	44
2.7 Funciones del temporizador	48
2.8 Funciones del USART	57
2.9 Funciones del USART	66
Capítulo 3. Biblioteca de periféricos de software	
3.1 Introducción	75
3.2 Funciones de la pantalla LCD externa	75
3.3 Funciones externas de CAN2510	82
3.4 Funciones del software I2C	105
3.5 Funciones del software SPI®	111
3.6 Funciones del software UART	114
Capítulo 4. Biblioteca de software general	
4.1 Introducción	119
4.2 Funciones de clasificación de caracteres	119
4.3 Funciones de conversión de datos	124
4.4 Funciones de manipulación de cadenas y memoria	128
4.5 Funciones de retardo	144
4.6 Funciones de reinicio	146
4.7 Funciones de salida de caracteres	149
Capítulo 5. Bibliotecas matemáticas	
5.1 Introducción	159
5.2 Biblioteca matemática de coma flotante de 32 bits	159
5.3 Las funciones matemáticas de la biblioteca estándar de C	162

Bibliotecas del compilador **MPLAB® C18 C**

Glosario	169
Índice	175
Ventas y servicio en todo el mundo	180



COMPILEADOR MPLAB® C18 C BIBLIOTECAS

Prefacio

AVISO A LOS CLIENTES

Toda la documentación se vuelve anticuada y este manual no es una excepción. Las herramientas y la documentación de Microchip están en constante evolución para satisfacer las necesidades de los clientes, por lo que algunos cuadros de diálogo y/o descripciones de herramientas reales pueden diferir de los de este documento. Consulte nuestro sitio web (www.microchip.com) para obtener la documentación más reciente disponible.

Los documentos se identifican con un número "DS". Este número se encuentra en la parte inferior de cada página, delante del número de página. La convención de numeración para el número DS es "DSXXXXA", donde "XXXX" es el número de documento y "A" es el nivel de revisión del documento.

Para obtener la información más actualizada sobre las herramientas de desarrollo, consulte la ayuda en línea de MPLAB® IDE. Seleccione el menú Ayuda y luego Temas para abrir una lista de archivos de ayuda en línea disponibles.

INTRODUCCIÓN

El objetivo de este documento es brindar información detallada sobre las bibliotecas y los archivos de objetos precompilados que se pueden usar con el compilador MPLAB® C18 C de Microchip.

DISEÑO DEL DOCUMENTO

El diseño del documento es el siguiente: •

Capítulo 1: Descripción general : describe las bibliotecas y los archivos de objetos precompilados disponible.

• **Capítulo 2: Funciones de periféricos de hardware :** describe cada hardware función de biblioteca periférica. •

Capítulo 3: Biblioteca de periféricos de software : describe cada periférico de software función de biblioteca.

• **Capítulo 4: Biblioteca de software general :** describe cada biblioteca de software general función.

• **Capítulo 5: Biblioteca matemática :** analiza las funciones de la biblioteca matemática. •

Glosario : un glosario de términos utilizados en esta guía. • **Índice :** lista de referencias cruzadas de términos, características y secciones de este documento.

Bibliotecas del compilador MPLAB® C18 C

LAS CONVENCIONES USADAS EN ESTA GUÍA

Este manual utiliza las siguientes convenciones de documentación:

CONVENCIONES DE DOCUMENTACIÓN

Descripción	representa	Ejemplos
fuente arial:		
Caracteres en cursiva	Libros de referencia	Guía del usuario de MPLAB® IDE
Fuente de mensajería:		
Mensajero sencillo	Ejemplo de código fuente	#define INICIO
	Nombres de archivo	autoexec.bat
	Rutas de archivos	c:\mcc18\asm,
	Palabras clave	_endasm, estática
	Opciones de línea de comandos	-Opa+, -Opa
Mensajero en cursiva	Un argumento variable	archivo.o, donde archivo puede ser cualquier nombre de archivo válido
Obnnnn	Un número binario donde n es un dígito binario	0b00100, 0b10
0xnnnn	Un número hexadecimal donde n es un dígito hexadecimal	0xFFFF, 0x007A
Corchetes [] Argumentos opcionales	Corchetes y	mcc18 [opciones] archivo [opciones] nivel de error {0}
Elección del carácter de carácter de selección OR Elipses...	repetición (llamado argumentos; una elección OR Elipses...)	{1}
	Reemplaza el texto repetido	var_name [, var_name...] vacío principal
	Representa el código proporcionado por usuario	(vacío) { ... }

LECTURA RECOMENDADA

Para obtener más información sobre las bibliotecas incluidas y los archivos de objetos precompilados para los compiladores, el funcionamiento de MPLAB IDE y el uso de otras herramientas, se recomienda leer lo siguiente.

Léame.c18

Para obtener la información más reciente sobre el uso del compilador MPLAB C18 C, lea el archivo readme.c18 (texto ASCII) incluido con el software. Este archivo Léame contiene información actualizada que puede no estar incluida en este documento.

Léame.xxx

Para obtener la información más reciente sobre otras herramientas de Microchip (MPLAB IDE, MPLINK™ linker, etc.), lea los archivos Léame asociados (archivo de texto ASCII) incluidos con el software.

Guía de inicio del compilador MPLAB® C18 C (DS51295)

Describe cómo instalar el compilador MPLAB C18, cómo escribir programas simples y cómo usar MPLAB IDE con el compilador.

Guía del usuario del compilador MPLAB® C18 C (DS51288)

Guía completa que describe el funcionamiento y las funciones del compilador MPLAB C18 C de Microchip para dispositivos PIC18.

Guía de inicio rápido de MPLAB® IDE (DS51281)

Describe cómo configurar el software MPLAB IDE y usarlo para crear proyectos y programar dispositivos.

Guía del usuario de MPASM™ Assembler, MPLINK™ Object Linker y MPLIB™ Object Librarian (DS33014)

Describe cómo utilizar el ensamblador (MPASM), el enlazador (MPLINK) y el bibliotecario (MPLIB) de Microchip PICmicro® MCU.

Manual de referencia de la familia de microcontroladores PICmicro® 18C (DS39500)

Se centra en la familia de dispositivos MCU mejorada. Se explica el funcionamiento de la arquitectura de la familia MCU mejorada y los módulos periféricos, pero no cubre los detalles de cada dispositivo.

Hojas de datos del dispositivo PIC18 y notas de aplicación

Las hojas de datos describen el funcionamiento y las especificaciones eléctricas de los dispositivos PIC18. Las notas de aplicación describen cómo utilizar los dispositivos PIC18.

Para obtener cualquiera de los documentos enumerados anteriormente, visite el sitio web de Microchip (www.microchip.com) para recuperar estos documentos en formato Adobe Acrobat (.pdf).

EL SITIO WEB DE MICROCHIP

Microchip proporciona soporte en línea a través de nuestro sitio web en www.microchip.com. Este sitio web se utiliza como un medio para que los archivos y la información estén fácilmente disponibles para los clientes. Accesible mediante su navegador de Internet favorito, el sitio web contiene la siguiente información:

- **Soporte de productos :** hojas de datos y erratas, notas de aplicación y muestra programas, recursos de diseño, guías de usuario y documentos de soporte de hardware, últimas versiones de software y software archivado
- **Soporte técnico general :** preguntas frecuentes (FAQ), solicitudes de soporte técnico, grupos de discusión en línea, lista de miembros del programa de consultores de Microchip
- **Negocios de Microchip :** selector de productos y guías de pedidos, los últimos comunicados de prensa de Microchip, lista de seminarios y eventos, listas de oficinas de ventas, distribuidores y representantes de fábrica de Microchip.

Bibliotecas del compilador MPLAB® C18 C

SERVICIO DE NOTIFICACIÓN DE CAMBIO DE CLIENTES DE SISTEMAS DE DESARROLLO

El servicio de notificación al cliente de Microchip ayuda a mantener a los clientes actualizados sobre los productos de Microchip. Los suscriptores recibirán una notificación por correo electrónico cada vez que haya cambios, actualizaciones, revisiones o erratas relacionadas con una familia de productos específica o una herramienta de desarrollo de interés.

Para registrarse, acceda al sitio web de Microchip en www.microchip.com, haga clic en Notificación de cambio de cliente y siga las instrucciones de registro.

Las categorías del grupo de productos de sistemas de desarrollo son:

Compiladores: la información más reciente sobre los compiladores Microchip C y otras herramientas de lenguaje.

Estos incluyen los compiladores MPLAB C18 y MPLAB C30 C; ensambladores MPASM™ y MPLAB ASM30; enlazadores de objetos MPLINK™ y MPLAB LINK30; y bibliotecarios de objetos MPLIB™ y MPLAB LIB30.

• Emuladores: la información más reciente sobre emuladores en circuito de Microchip. Esto incluye MPLAB ICE 2000 y MPLAB ICE 4000.

• Depuradores en circuito: la información más reciente sobre el Microchip en circuito depurador, MPLAB ICD 2.

MPLAB® IDE: la información más reciente sobre Microchip MPLAB IDE, Windows®

Entorno de desarrollo integrado para herramientas de sistemas de desarrollo. Esta lista se centra en MPLAB IDE, MPLAB SIM simulador, MPLAB IDE Project Manager y funciones generales de edición y depuración.

• Programadores: la información más reciente sobre los programadores de Microchip. Estos incluyen los programadores de dispositivos MPLAB PM3 y PRO MATE® II y el PICSTART® Programadores de desarrollo Plus y PICkit®.

ATENCIÓN AL CLIENTE

Los usuarios de productos Microchip pueden recibir asistencia a través de varios canales:

- Distribuidor o Representante • Oficina de Ventas Local
- Ingeniero de Aplicaciones de Campo (FAE) • Soporte Técnico • Línea de Información de Sistemas de Desarrollo

Los clientes deben comunicarse con su distribuidor, representante o ingeniero de aplicaciones de campo (FAE) para obtener asistencia. oficinas de ventas locales también están disponibles para ayudar a los clientes. Al dorso de este documento se incluye una lista de las oficinas y ubicaciones de ventas.

El soporte técnico está disponible a través del sitio web en: <http://support.microchip.com>

Además, hay una Línea de información de sistemas de desarrollo que enumera las últimas versiones de los productos de software de sistemas de desarrollo de Microchip. Esta línea también brinda información sobre cómo los clientes pueden recibir los kits de actualización disponibles actualmente.

Los números de la Línea de Información de Sistemas de Desarrollo son:

1-800-755-2345 – Estados Unidos y la mayor parte de Canadá

1-480-792-7302 – Otras ubicaciones internacionales



Capítulo 1. Descripción general

1.1 INTRODUCCIÓN

Este capítulo ofrece una descripción general de los archivos de la biblioteca MPLAB C18 y los archivos de objetos precompilados que se pueden incluir en una aplicación.

1.2 DESCRIPCIÓN GENERAL DE LAS BIBLIOTECAS MPLAB C18

Una biblioteca es una colección de funciones agrupadas para referencia y facilidad de enlace. Consulte MPASM™ Assembler, MPLINK™ Object Linker, MPLIB™ Object Librarian User's Guide (DS33014) para obtener más información sobre la creación y el mantenimiento de bibliotecas.

Las bibliotecas MPLAB C18 están incluidas en el subdirectorio lib de la instalación. Estos se pueden vincular directamente a una aplicación utilizando el vinculador MPLINK.

Estos archivos se compilaron previamente en el directorio c:\mcc18\src de Microchip. El directorio src\tradicional contiene los archivos para el modo no extendido y src\extended contiene los archivos para el modo extendido. Si elige no instalar el compilador y los archivos relacionados en el directorio c:\mcc18, el código fuente de las bibliotecas no se mostrará en el archivo de lista del enlazador y no se podrá recorrer cuando se usa MPLAB IDE.

Para incluir el código de la biblioteca en el archivo .lst y poder realizar un solo paso a través de las funciones de la biblioteca, siga las instrucciones de la **Sección 1.3.3**, la **Sección 1.4.3** y la **Sección 1.5.3** para reconstruir las bibliotecas utilizando los archivos por lotes suministrados (.bat) que se encuentra en los directorios src, src\traditional y src\extended.

1.3 CÓDIGO DE INICIO

1.3.1 Descripción general

Se proporcionan tres versiones del código de inicio con MPLAB C18, con diferentes niveles de inicialización. Los archivos de objeto c018*.o se utilizan con el compilador que funciona en el modo no extendido. Los archivos de objeto c018*_e.o se usan con el compilador cuando se opera en modo extendido. En orden creciente de complejidad, son: **c018.o/c018_e.o** inicializa la pila de software C y salta al inicio de la función de la aplicación, **main()**. **c018i.o/c018i_e.o** realiza todas las mismas tareas que c018.o/c018_e.o y también asigna los valores apropiados a los datos inicializados antes de llamar a la aplicación del usuario.

La inicialización es necesaria si las variables globales o estáticas se establecen en un valor cuando se definen. Este es el código de inicio que se incluye en los archivos de script del enlazador que se proporcionan con MPLAB C18. **c018iz.o/c018iz_e.o** realiza todas las mismas tareas que c018i.o/c018i_e.o y también asigna cero a todas las variables no inicializadas, como se requiere para el cumplimiento estricto de ANSI.

Bibliotecas del compilador MPLAB® C18 C

1.3.2 Código fuente

El código fuente de las rutinas de inicio se puede encontrar en los subdirectorios src\traditional\startup y src\extended\startup de la instalación del compilador.

1.3.3 Reconstrucción

El archivo por lotes makestartup.bat se puede usar para reconstruir el código de inicio y copiar los archivos de objetos generados en el directorio lib.

Antes de reconstruir el código de inicio con makestartup.bat, verifique que MPLAB C18 (mcc18.exe) esté en su camino.

1.4 BIBLIOTECA INDEPENDIENTE DEL PROCESADOR

1.4.1 Descripción general

La biblioteca C estándar (clib.lib o clib_e.lib) proporciona funciones que son compatibles con la arquitectura central PIC18: aquellas que son compatibles con todos los procesadores de la familia. Estas funciones se describen en los siguientes capítulos:

- Biblioteca de software general, Capítulo 4.
- Bibliotecas de matemáticas, Capítulo 5.

1.4.2 Código fuente

El código fuente de las funciones de la biblioteca C estándar se puede encontrar en los siguientes subdirectorios de la instalación del compilador:

- src\tradicional\matemáticas
- src\extended\math
- src\tradicional\retrasos •
- src\extended\retrasos •
- src\tradicional\stdclib • src\extended\stdclib

1.4.3 Reconstrucción

El archivo por lotes makeclib.bat se puede usar para reconstruir la biblioteca independiente del procesador. Antes de invocar este archivo por lotes, verifique que las siguientes herramientas estén en su camino: • MPLAB C18 (mcc18.exe) • Ensamblador MPASM (mpasm.exe) • Bibliotecario MPLIB (mplib.exe)

Además, antes de reconstruir la biblioteca C estándar, asegúrese de que la variable de entorno MCC_INCLUDE esté configurada en la ruta de los archivos de inclusión de MPLAB C18 (p. ej., c:\mcc18\h).

1.5 BIBLIOTECAS ESPECÍFICAS DEL PROCESADOR

1.5.1 Descripción general

Los archivos de biblioteca específicos del procesador contienen definiciones que pueden variar entre los miembros individuales de la familia PIC18. Esto incluye todas las rutinas periféricas y las definiciones del Registro de función especial (SFR). Las rutinas periféricas que se proporcionan incluyen tanto aquellas diseñadas para usar los periféricos de hardware como aquellas que implementan una interfaz periférica usando líneas de E/S de propósito general. Las funciones incluidas en las bibliotecas específicas del procesador se describen en los siguientes capítulos:

- **Capítulo 2. “Funciones de periféricos de hardware”** •

Capítulo 3. “Biblioteca de periféricos de software”

Las bibliotecas específicas del procesador se denominan:

p processor.lib: biblioteca específica del procesador en modo no extendido

p procesador_e.lib: biblioteca específica del procesador en modo extendido Por

ejemplo, el archivo de biblioteca para PIC18F4620 se llama p18f4620.lib para la versión no extendida de la biblioteca y p18f4620_e.lib para la versión extendida de la biblioteca.

1.5.2 Código fuente

El código fuente de las bibliotecas específicas del procesador se puede encontrar en los siguientes subdirectorios de la instalación del compilador: • src\tradicional\pmc • src\extended\pmc • src\tradicional\proc

- src\extended\proc

1.5.3 Reconstrucción

El archivo por lotes makeplib.bat se puede usar para reconstruir las bibliotecas específicas del procesador. Antes de invocar este archivo por lotes, verifique que las siguientes herramientas estén en su camino:

- MPLAB C18 (mcc18.exe) • Ensamblador MPASM (mpasm.exe) • Bibliotecario MPLIB (mplib.exe)

Además, antes de invocar makeplib.bat, asegúrese de que la variable de entorno MCC_INCLUDE esté configurada en la ruta de los archivos de inclusión de MPLAB C18 (p. ej., c:\mcc18\h).

Bibliotecas del compilador **MPLAB® C18 C**

NOTAS:



COMPILEADOR MPLAB® C18 C BIBLIOTECAS

Capítulo 2. Funciones de los periféricos de hardware

2.1 INTRODUCCIÓN

Este capítulo documenta las funciones de los periféricos de hardware que se encuentran en las bibliotecas específicas del procesador. El código fuente de todas estas funciones se incluye con MPLAB C18 en los subdirectorios `src\traditional\pmc` y `src\extended\pmc` de la instalación del compilador.

Consulte MPASM™ Assembler, MPLINK™ Object Linker, MPLIB™ Object Librarian User's Guide (DS33014) para obtener más información sobre la administración de bibliotecas mediante MPLIB library.

Los siguientes periféricos son compatibles con las rutinas de la biblioteca MPLAB

C18: • Convertidor A/D ([Sección 2.2 “Funciones del convertidor A/D”](#)) • Captura de entrada ([Sección 2.3 “Funciones de captura de entrada”](#)) • I2C™ ([Sección 2.4 “Funciones I2C™”](#)) • Puertos de E/S ([Sección 2.5 “Funciones de los puertos de E/S”](#)) • Microwire ([Sección 2.6 “Funciones de Microwire”](#)) • Modulación de ancho de pulso (PWM) ([Sección 2.7 “Modulación de ancho de pulso”](#)) •

SPI™ ([Sección 2.8 “Funciones SPI™”](#)) •

Temporizador ([Sección 2.9 “Funciones del temporizador”](#)) • USART ([Sección 2.10 “Funciones USART”](#))

2.2 FUNCIONES DEL CONVERTIDOR A/D

El periférico A/D es compatible con las siguientes funciones:

TABLA 2-1: FUNCIONES DEL CONVERTIDOR A/D

Función	Descripción
BusyADC	¿El convertidor A/D está realizando actualmente una conversión?
CerrarADC	Deshabilite el convertidor A/D.
ConvertADC	Inicie una conversión A/D.
OpenADC	Configure el convertidor A/D.
LeerADC	Lea los resultados de una conversión A/D.
EstablecerChanADC	Seleccione el canal A/D que se utilizará.

Bibliotecas del compilador MPLAB® C18 C

2.2.1 Descripciones de funciones

BusyADC

Función:	¿El convertidor A/D está realizando una conversión actualmente?
Incluir:	adc.h
Prototipo:	char BusyADC(vacío);
Observaciones:	Esta función indica si el periférico A/D está en proceso de convertir un valor. 1 si el periférico A/D está realizando una conversión. 0 si el periférico A/D no está
Valor de retorno:	realizando una conversión.
Nombre del archivo:	adcbusy.c

CerrarADC

Función:	Deshabilite el convertidor A/D.
Incluir:	adc.h
Prototipo:	vacio CloseADC (vacío);
Observaciones:	Esta función desactiva el convertidor A/D y el mecanismo de interrupción A/D.

ConvertADC

Función:	Inicia el proceso de conversión A/D.
Incluir:	adc.h
Prototipo:	vacio ConvertADC (vacío);
Observaciones:	Esta función inicia una conversión A/D. La función BusyADC() se puede utilizar para detectar la finalización de la conversión.
Nombre del archivo:	adccconv.c

OpenADC

PIC18CXX2, PIC18FXX2, PIC18FXX8, PIC18FXX39

Función:	Configure el convertidor A/D.														
Incluir:	adc.h														
Prototipo:	void OpenADC (configuración de caracteres sin firmar, configuración de caracteres sin firma2);														
Argumentos:	config Una máscara de bits que se crea realizando una operación AND bit a bit ('&') con un valor de cada una de las categorías enumeradas a continuación. Estos valores se definen en el archivo adc.h.														
Fuente de reloj A/D:	<table border="0"> <tr> <td>ADC_FOSC_2</td> <td>FOSC / 2</td> </tr> <tr> <td>ADC_FOSC_4</td> <td>FOSC / 4</td> </tr> <tr> <td>ADC_FOSC_8</td> <td>FOSC / 8</td> </tr> <tr> <td>ADC_FOSC_16</td> <td>FOSC / 16</td> </tr> <tr> <td>ADC_FOSC_32</td> <td>FOSC / 32</td> </tr> <tr> <td>ADC_FOSC_64</td> <td>FOSC / 64</td> </tr> <tr> <td>ADC_FOSC_RC</td> <td>Oscilador RC Interno</td> </tr> </table>	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	Oscilador RC Interno
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	Oscilador RC Interno														

Justificación del resultado A/D:

ADC_RIGHT JUST	Resultado en bits menos significativos
ADC_LEFT JUST	Resultado en bits más significativos

**OpenADC
PIC18CXX2, PIC18FXX2, PIC18FXX8, PIC18FXX39 (continuación)**

Fuente de referencia de voltaje A/D:

ADC_8ANA_0REF	VREF+=VDD, VREF-=VSS, Todos los canales analógicos
ADC_7ANA_1REF	AN3=VREF+, Todos los canales analógicos excepto AN3
ADC_6ANA_2REF	AN3=VREF+, AN2=VREF
ADC_6ANA_0REF	VREF+=VDD, VREF-=VSS
ADC_5ANA_1REF	AN3=VREF+, VREF-=VSS
ADC_5ANA_0REF	VREF+=VDD, VREF-=VSS
ADC_4ANA_2REF	AN3=VREF+, AN2=VREF
ADC_4ANA_1REF	AN3=VREF+
ADC_3ANA_2REF	AN3=VREF+, AN2=VREF
ADC_3ANA_0REF	VREF+=VDD, VREF-=VSS
ADC_2ANA_2REF	AN3=VREF+, AN2=VREF
ADC_2ANA_1REF	AN3=VREF+
ADC_1ANA_2REF	AN3=VREF+, AN2=VREF-, AN0=A
ADC_1ANA_0REF	AN0 es entrada analógica
ADC_0ANA_0REF	Todas las E/S digitales

config2

Una máscara de bits que se crea realizando una operación AND bit a bit ('&') con un valor de cada una de las categorías enumeradas a continuación. Estos valores se definen en el archivo adch.h.

Canal:

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

Interrupciones A/D:

ADC_INT_ON	Interrupciones habilitadas
ADC_INT_OFF	Interrupciones deshabilitadas

Observaciones:

Esta función restablece el periférico A/D al estado POR y configura los registros de funciones especiales (SFR) relacionados con A/D de acuerdo con las opciones especificadas.

Nombre del archivo: adcopen.c
Ejemplo de código: OpenADC (ADC_FOSC_32 &
ADC_DERECHO JUST &
ADC_1ANA_0REF,
ADC_CH0 &
ADC_INT_OFF);

Bibliotecas del compilador MPLAB® C18 C

OpenADC

**PIC18C658/858, PIC18C601/801,
FOTO18F6X20, FOTO18F8X20**

Función:	Configure el convertidor A/D.
Incluir:	adc.h
Prototipo:	void OpenADC (configuración de caracteres sin firmar, configuración de caracteres sin firmar2);
Argumentos:	config

Una máscara de bits que se crea realizando una operación AND bit a bit ('&') con un valor de cada una de las categorías enumeradas a continuación. Estos valores se definen en el archivo adc.h.

Fuente de reloj A/D:

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	Oscilador RC Interno

Justificación del resultado A/D:

ADC_RIGHT_JUST Resultado en bits menos significativos
ADC_LEFT_JUST Resultado en bits más significativos

Configuración del puerto A/D:

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

config2

Una máscara de bits que se crea realizando una operación AND bit a bit ('&') con un valor de cada una de las categorías enumeradas a continuación. Estos valores se definen en el archivo adc.h.

OpenADC

**PIC18C658/858, PIC18C601/801,
PIC18F6X20, PIC18F8X20 (Continuación)**

Canal:

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

Interrupciones A/D:

ADC_INT_ON	Interrupciones habilitadas
ADC_INT_OFF	Interrupciones deshabilitadas

Configuración A/D VREF+ :

ADC_VREFPLUS_VDD VREF+ = AVDD
ADC_VREFPLUS_EXT VREF+ = externo

A/D VREF- configuración:

ADC_VREFMINUS_VSS VREF- = AVSS
ADC_VREFMINUS_EXT VREF- = externo Esta

Observaciones: función restablece los registros relacionados con A/D al estado POR y luego configura el reloj, el formato de resultado, la referencia de voltaje, el puerto y el canal.

Nombre del archivo: adcopen.c

Ejemplo de código: OpenADC (ADC_FOSC_32 &
ADC_DERECHO JUST &
ADC_14ANA,
ADC_CH0 &
ADC_INT_OFF);

Bibliotecas del compilador MPLAB® C18 C

OpenADC

Todos los demás procesadores

Función:	Configure el convertidor A/D.
Incluir:	adc.h
Prototipo:	void OpenADC (configuración de caracteres sin firmar, configuración de caracteres sin firmar2 , configuración de puerto de caracteres sin firmar);
Argumentos:	config

Una máscara de bits que se crea realizando una operación AND bit a bit ('&') con un valor de cada una de las categorías enumeradas a continuación. Estos valores se definen en el archivo adc.h.

Fuente de reloj A/D:

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	Oscilador RC Interno

Justificación del resultado A/D

D: ADC_RIGHT JUST Resultado en bits menos significativos

ADC_LEFT JUST Resultado en bits más significativos Selección

de **tiempo de adquisición A/D:** 0 Tad ADC_0_TAD 2 Tad ADC_2_TAD
8 Tad ADC_8_TAD 12 Tad ADC_12_TAD 16 Tad ADC_16_TAD
32 Tad ADC_32_TAD 64 Tad ADC_64_TAD 128 Tad ADC_128_TAD
Config Una máscara de bits que se crea realizando una operación AND bit a bit ('&') con un valor de cada una de las categorías enumeradas a continuación. Estos valores se definen en el archivo adc.h.

Canal:

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

OpenADC

Todos los demás procesadores (continuación)

Interrupciones A/D:

ADC_INT_ON	Interrupciones habilitadas
ADC_INT_OFF	Interrupciones deshabilitadas

Configuración de voltaje A/D:

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

portconfig E

valor de portconfig es cualquier valor de 0 a 127 inclusive para PIC18F1220/1320 y de 0 a 15 inclusive para todos los demás procesadores. Este es el valor de los bits del 0 al 6 o de los bits del 0 al 3 del registro ADCON1, que son los bits de configuración del puerto.

Observaciones: Esta función restablece los registros relacionados con A/D al estado POR y luego configura el reloj, el formato de resultado, la referencia de voltaje, el puerto y el canal.

Nombre del archivo: adcopen.c

Ejemplo de código: OpenADC (ADC_FOSC_32 &
ADC_DERECHO JUST &
ADC_12_TAD,
ADC_CHO &
ADC_INT_OFF, 15);

LeerADC

Función: Leer el resultado de una conversión A/D.

Incluir: adc.h

Prototipo: int ReadADC (vacío);

Observaciones: Esta función lee el resultado de 16 bits de una conversión A/D.

Valor de retorno: Esta función devuelve el resultado con signo de 16 bits de la conversión A/D. Según la configuración del convertidor A/D (p. ej., utilizando la función OpenADC()), el resultado estará contenido en los bits menos significativos o más significativos del resultado de 16 bits.

Nombre del archivo: adcread.c

Bibliotecas del compilador MPLAB® C18 C

EstablecerChanADC

Función: Seleccione el canal utilizado como entrada al convertidor A/D.

Incluir: adc.h

Prototipo: void SetChanADC (canal de caracteres sin firmar);

Argumentos: **canal**
Uno de los siguientes valores (definidos en adc.h): Canal
 ADC_CH0 0 Canal 1 Canal 2 Canal 3
 ADC_CH1 Canal 4 Canal 5 Canal 6
 ADC_CH2 Canal 7 Canal 8 Canal 9
 ADC_CH3 Canal 10 Canal 11
 ADC_CH4
 ADC_CH5
 ADC_CH6
 ADC_CH7
 ADC_CH8
 ADC_CH9
 ADC_CH10
 ADC_CH11

Observaciones: Selecciona el pin que se usará como entrada al convertidor A/D.

Nombre del archivo: adcsetch.c

Ejemplo de código: SetChanADC(ADC_CH0);

2.2.2 Ejemplo de uso de las rutinas del convertidor A/D

```
#incluir <p18C452.h> #incluir
<adc.h> #incluir <stdlib.h>
#incluir <retrasos.h>

resultado int;

vacío principal (vacío) {

    // configurar convertidor A/D
    OpenADC (ADC_FOSC_32 y ADC_RIGHT_JUST y ADC_8ANA_0REF,
              ADC_CH0 y ADC_INT_OFF);

    Retardo10TCYx( 5 );           // Retraso por 50TCY
    ConvertirADC();               // Iniciar la conversión
    while( BusyADC() ); resultado = ReadADC(); // leer resultado

    CerrarADC();                 // Deshabilitar convertidor A/D
}
```

2.3 FUNCIONES DE CAPTURA DE ENTRADA

El periférico de captura es compatible con las siguientes funciones:

TABLA 2-2: FUNCIONES DE CAPTURA DE ENTRADA

Función	Descripción
CerrarCapturex	Deshabilitar captura periférico x.
AbrirCapturex	Configurar periférico de captura x.
LeerCapturex	Leer un valor del periférico de captura x.
CerrarECapturex(1)	Deshabilitar periférico de captura mejorada x.
AbrirECapturex(1)	Configure el periférico de captura mejorado x.
LeerECapturex(1)	Lea un valor del periférico de captura mejorado x.

Nota 1: Las funciones de captura mejorada solo están disponibles en aquellos dispositivos con un registro ECCPxCON.

2.3.1 Descripciones de funciones**CerrarCapturar1****CerrarCapturar2****CerrarCapturar3****CerrarCapturar4****CerrarCapturar5****CerrarECapture1**

Función: Deshabilitar la captura de entrada x.

Incluir: capturar.h

Prototipo: vacío CloseCapture1 (vacío); vacío
CloseCapture2 (vacío); vacío CloseCapture3
(vacío); vacío CloseCapture4 (vacío); vacío
CloseCapture5 (vacío); vacío CloseECapture1
(vacío);

Observaciones: Esta función deshabilita la interrupción correspondiente a la captura de entrada especificada.

Nombre del archivo:
cp1cerrar.c
cp2cerrar.c
cp3cerrar.c
cp4cerrar.c
cp5cerrar.c
ep1cerrar.c

Bibliotecas del compilador MPLAB® C18 C

AbrirCaptura1

AbrirCaptura2

AbrirCaptura3

AbrirCaptura4

AbrirCaptura5

AbrirECapture1

Función: Configure y habilite la captura de entrada x.

Incluir: capturar.h

Prototipo: void OpenCapture1 (configuración de caracteres sin firmar); void OpenCapture2 (configuración de caracteres sin firmar); void OpenCapture3 (configuración de caracteres sin firmar); void OpenCapture4 (configuración de caracteres sin firmar); void OpenCapture5 (configuración de caracteres sin firmar); void OpenECapture1 (configuración de caracteres sin firmar);

Argumentos: config

Una máscara de bits que se crea realizando una operación AND bit a bit ('&') con un valor de cada una de las categorías enumeradas a continuación. Estos valores se definen en el archivo capture.h:

Habilitar interrupciones de CCP:

CAPTURE_INT_ON	Interrupciones habilitadas
CAPTURE_INT_OFF	Interrupciones deshabilitadas

Disparador de interrupción (reemplace x con el número de módulo CCP):

Cx_EVERY_FALL_EDGE	Interrumpir en cada borde descendente
Cx_EVERY_RISE_EDGE	Interrumpir en cada flanco ascendente
Cx_EVERY_4_RISE_EDGE	Interrupción en cada cuarto flanco ascendente
Cx_EVERY_16_RISE_EDGE	Interrupción cada 16 flancos ascendentes
EC1_EVERY_FALL_EDGE	Interrupción en cada flanco descendente (mejorado)
EC1_EVERY_RISE_EDGE	Interrupción en cada flanco ascendente (mejorado)
EC1_EVERY_4_RISE_EDGE	Interrupción en cada cuarto flanco ascendente (mejorado)
EC1_EVERY_16_RISE_EDGE	Interrupción cada 16 flancos ascendentes (mejorado)

Observaciones: Esta función primero restablece el módulo de captura al estado POR y luego configura la captura de entrada para la detección de borde especificada.

Las funciones de captura utilizan una estructura, definida en capture.h, para indicar el estado de desbordamiento de cada uno de los módulos de captura. Esta estructura se llama CapStatus y tiene los siguientes campos de bits: Cap1OVF Cap2OVF Cap3OVF Cap4OVF Cap5OVF ECap1OVF

Además de abrir la captura, se debe habilitar el módulo de temporizador adecuado antes de que funcione cualquiera de las capturas. Consulte la hoja de datos para las configuraciones de interconexión de temporizador y CCP y la **Sección 2.9 "Funciones de temporizador"** para conocer los argumentos utilizados con CCP en OpenTimer3.

AbrirCaptura1**AbrirCaptura2****AbrirCaptura3****AbrirCaptura4****AbrirCaptura5****OpenECapture1 (continuación)**

Nombre del archivo: cp1abierto.c
 cp2abierto.c
 cp3abierto.c
 cp4abierto.c
 cp5abierto.c
 ep1abierto.c

Ejemplo de código: Captura abierta1(CAPTURE_INT_ON &
 C1_EVERY_4_RISE_EDGE);

LeerCapturar1**LeerCapturar2****LeerCapturar3****LeerCapturar4****LeerCapturar5****LeerECapturar1**

Función: Leer el resultado de un evento de captura de la captura de entrada especificada.

Incluir: capturar.h

Prototipo: int sin firmar ReadCapture1 (vacío); int sin firmar
 ReadCapture2 (vacío); int sin firmar ReadCapture3
 (vacío); int sin firmar ReadCapture4 (vacío); int sin
 firmar ReadCapture5 (vacío); int sin firmar
 ReadECapture1 (vacío);

Observaciones: Esta función lee el valor de los SFR de la captura de entrada respectiva.

Valor de retorno: Esta función devuelve el resultado del evento de captura.

Nombre del archivo: cp1leer.c
 cp2leer.c
 cp3leer.c
 cp4leer.c
 cp5leer.c
 ep1leer.c

2.3.2 Ejemplo de uso de las rutinas de captura

Este ejemplo demuestra el uso de las rutinas de la biblioteca de captura en un entorno "sondeado" (no controlado por interrupciones).

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

vacío principal (vacío) {

    resultado int sin firmar; char
    cadena[7];

    // Configurar Captura1
    Captura abierta1 (C1_EVERY_4_RISE_EDGE &
                      CAPTURE_INT_OFF);

    // Configurar Temporizador3
    AbrirTemporizador3( TIMER_INT_OFF &
                        T3_FUENTE_INT);

    // Configurar USART
    AbrirUSART( USART_TX_INT_OFF &
                USART_RX_INT_OFF &
                USART_SYNCH_MODE &
                USART_EIGHT_BIT &
                USART_CONT_RX,
                25);

    while(!PIR1bits.CCP1IF); // Esperar el resultado del evento =
    ReadCapture1(); // lee el resultado // convierte a cadena ultoa(result,str);

    // Escriba la cadena en el USART si // no se ha producido una
    condición de desbordamiento. if(CapStatus.Cap1OVF) {

        poneUSART(str);
    }

    // Limpiar
    CerrarCaptura1();
    CerrarTemporizador3();
    CerrarUSART();
}
```

2.4 FUNCIONES I2C™

Las siguientes rutinas se proporcionan para dispositivos con un único periférico I2C :

TABLA 2-3: FUNCIONES DEL PERIFÉRICO I2C SIMPLE

Función	Descripción
AckI2C	Generar condición de reconocimiento de bus I2C .
CerrarI2C	Deshabilite el módulo SSP.
DatosRdyI2C	¿Están disponibles los datos en el búfer I2C ?
obtenerI2C	Lea un solo byte del bus I2C .
obtieneI2C	Lee una cadena del bus I2C que funciona en modo maestro I2C .
InactivaI2C	Bucle hasta que el bus I2C esté inactivo.
NotAckI2C	Generar condición de no reconocimiento de bus I2C .
AbiertoI2C	Configure el módulo SSP.
putI2C	Escriba un solo byte en el bus I2C .
ponerI2C	Escriba una cadena en el bus I2C que funciona en modo Maestro o Esclavo.
LeerI2C	Lea un solo byte del bus I2C .
ReiniciarI2C	Genere una condición de reinicio del bus I2C .
InicioI2C	Genere una condición de inicio de bus I2C .
DetenerI2C	Genere una condición de parada de bus I2C .
escribirI2C	Escriba un solo byte en el bus I2C .

Las siguientes rutinas se proporcionan para dispositivos con múltiples periféricos I2C :

TABLA 2-4: MÚLTIPLES FUNCIONES DE PERIFÉRICOS I2C

Función	Descripción
AckI2Cx	Generar condición de reconocimiento de bus I2Cx .
CerrarI2Cx	Deshabilite el módulo SS x .
DataRdyI2Cx	¿Están disponibles los datos en el búfer I2Cx ?
obtenerI2Cx	Lea un solo byte del bus I2Cx .
obtieneI2Cx	Lee una cadena del bus I2Cx que funciona en modo maestro I2C .
InactivaI2Cx	Bucle hasta que el bus I2Cx esté inactivo.
NotAckI2Cx	Generar condición de no reconocimiento de bus I2Cx .
AbiertoI2Cx	Configure el módulo SSPx .
putI2Cx	Escriba un solo byte en el bus I2Cx .
ponerI2Cx	Escriba una cadena en el bus I2Cx que funciona en modo Maestro o Esclavo.
ReadI2Cx	Lea un solo byte del bus I2Cx .
RestartI2Cx	Genera una condición de reinicio del bus I2Cx .
InicioI2Cx	Genere una condición de inicio de bus I2Cx .
DetenerI2Cx	Genere una condición de parada de bus I2Cx .
EscribirI2Cx	Escriba un solo byte en el bus I2Cx .

Bibliotecas del compilador MPLAB® C18 C

Las siguientes funciones también se proporcionan para interactuar con un dispositivo de memoria EE como el Microchip 24LC01B mediante la interfaz I2C :

TABLA 2-5: FUNCIONES DE LA INTERFAZ PARA DISPOSITIVOS DE MEMORIA EE

Función	Descripción
EEAckPollingx	Genere la secuencia de sondeo de reconocimiento.
EEByteWritex	Escribe un solo byte.
EECurrentAddReadx	Leer un solo byte desde la siguiente ubicación.
EEPageWritex	Escribe una cadena de datos.
EERandomReadx	Leer un solo byte de una dirección arbitraria.
EESequentialReadx	Leer una cadena de datos.

2.4.1 Descripciones de funciones

AckI2C**AckI2C1****AckI2C2**

Función: Generar condición de reconocimiento de bus I2C .

Incluir: i2c.h

Prototipo: vacío AckI2C (vacío); vacío
AckI2C1 (vacío); vacío AckI2C2
(vacío);

Observaciones: Esta función genera una condición de reconocimiento del bus I2Cx .

Nombre del archivo: i2c_ack.c i2c1ack.c i2c2ack.c

CerrarI2C**CerrarI2C1****CerrarI2C2**

Función: Deshabilite el módulo SSPx.

Incluir: i2c.h

Prototipo: vacío CloseI2C (vacío); vacío
CloseI2C1 (vacío); vacío CloseI2C2
(vacío);

Observaciones: Esta función desactiva el módulo SSPx .

Nombre del archivo: i2c_cerrar.c
i2c1cerrar.c
i2c2cerrar.c

DatosRdyI2C**DatosRdyI2C1****DatosRdyI2C2**

Función:	¿Hay datos disponibles en el búfer I2Cx ?
Incluir:	i2c.h
Prototipo:	char sin firmar DataRdyI2C(void); carácter sin firmar DataRdyI2C1(void); carácter sin firmar DataRdyI2C2(void);
Observaciones:	Determina si hay un byte para leer en el búfer SSPx. 1 si hay datos en el búfer SSPx 0 si no hay datos en el búfer SSPx
Valor de retorno:	
Nombre del archivo:	i2c_dtrd.c i2c1dtrd.c i2c2dtrd.c
Ejemplo de código:	<pre>si (DataRdyI2C()) { var = getI2C(); }</pre>

getI2C**getI2C1****getI2C2**

getI2Cx se define como ReadI2Cx. Ver **ReadI2Cx**.

obtieneI2C**obtieneI2C1****obtieneI2C2**

Función:	Lea una cadena de longitud fija del bus I2Cx que opera en el I2C maestro modo.
Incluir:	i2c.h
Prototipo:	carácter sin firmar getsI2C(carácter sin signo * rdptr, longitud de carácter sin signo); carácter sin signo); carácter sin firmar getsI2C1(carácter sin signo * rdptr, longitud de carácter sin signo); carácter sin firmar getsI2C2(carácter sin signo rdptr, longitud de carácter sin signo);
Argumentos:	rdptr Puntero de tipo de carácter a PICmicro RAM para el almacenamiento de datos leídos desde el dispositivo I2C. longitud Número de bytes para leer desde el dispositivo I2Cx .
Observaciones:	Esta rutina lee una longitud de cadena de datos predefinida del bus I2Cx .

Bibliotecas del compilador MPLAB® C18 C

obtieneI2C

obtieneI2C1

obtieneI2C2 (Continuación)

Valor de retorno: 0 si se han enviado todos los bytes
-1 si se ha producido una colisión de bus

Nombre del archivo: i2c_gets.c
i2c1gets.c
i2c2gets.c

Ejemplo de código: cadena de caracteres sin firmar[15];
obtieneI2C(cadena, 15);

Inactivoi2C

Inactivoi2C1

Inactivoi2C2

Función: Bucle hasta que el bus I2Cx esté inactivo.

Incluir: i2c.h

Prototipo: vacío IdleI2C (vacío);

Observaciones: Esta función comprueba el estado del periférico I2C y espera a que el bus esté disponible. La función IdleI2C es necesaria ya que el periférico de hardware I2C no permite la puesta en cola de secuencias de bus.
El periférico I2C debe estar en estado inactivo antes de que se pueda iniciar una operación I2C o se generará una colisión de escritura.

Nombre del archivo: inactivoi2c.c

NotAckI2C

NotAckI2C1

NotAckI2C2

Función: Generar condición de no reconocimiento de bus I2Cx .

Incluir: i2c.h

Prototipo: void NotAckI2C(void); void
NotAckI2C1(void); void
NotAckI2C2(void);

Observaciones: Esta función genera una condición de No reconocimiento del bus I2Cx .

Nombre del archivo: i2c_nack.c i2c1nack.c i2c2nack.c

Abiertol2C**Abiertol2C1****Abiertol2C2**

Función:	Configure el módulo SSPx .
Incluir:	i2c.h
Prototipo:	void OpenI2C (modo de sincronización de caracteres sin firmar , giro de caracteres sin firmar); void OpenI2C1 (sincronización de caracteres sin firmar, cambio de caracteres sin firmar); void OpenI2C2 (modo de sincronización de caracteres sin firmar , giro de caracteres sin firmar);
Argumentos:	sync_mode Uno de los siguientes valores, definido en i2c.h: I2C Modo 2C Modo esclavo, dirección de esclavo, I2C Modo maestro MAESTRO
	montón Uno de los siguientes valores, definido en i2c.h: SLEW_OFF SLEW_ON Tasa de respuesta deshabilitada para el modo de 100 kHz Velocidad de respuesta habilitada para el modo de 400 kHz
Observaciones:	OpenI2Cx restablece el módulo SSPx al estado POR y luego configura el módulo para el modo Maestro/Esclavo y la velocidad de respuesta seleccionada.
Nombre del archivo:	i2c_abierto.c i2c1abierto.c i2c2abierto.c
Ejemplo de código:	OpenI2C(MAESTRO, SLEW_ON);

putcl2C**putcl2C1****putcl2C2**

putcl2Cx se define como WriteI2Cx. Ver **EscribirI2Cx**.

Bibliotecas del compilador MPLAB® C18 C

ponel2C

ponel2C1

ponel2C2

Función: Escriba una cadena de datos en el bus I2Cx que opera en Maestro o Esclavo modo.

Incluir: i2c.h

Prototipo: carácter sin firmar putsl2C(carácter
sin firmar *wrptr);
carácter sin firmar putsl2C1(carattere sin firmar *wrptr);
char sin firmar putsl2C2(carattere sin firmar *wrptr);

Argumentos: wrptr
Puntero a los datos que se escribirán en el bus I2C .

Observaciones: Esta rutina escribe una cadena de datos en el bus I2Cx hasta que se alcanza un carácter nulo. El carácter nulo en sí no se transmite. Esta rutina puede operar en modo Maestro o Esclavo.

Valor de retorno:

Modo maestro I2C :
0 si se alcanzó el carácter nulo en la cadena de datos -2 si
el dispositivo esclavo I2Cx respondió con un NO ACK
-3 si ocurrió una colisión de escritura

Modo I2C esclavo :

0 si se alcanzó el carácter nulo en la cadena de datos -2 si
el dispositivo I2Cx maestro respondió con un NO ACK que
terminó la transferencia de datos

Nombre del archivo: i2c_puts.c
i2c1puts.c
i2c2puts.c

Ejemplo de código: cadena de caracteres sin firmar[] = "datos para enviar";
ponel2C(cadena);

LeerI2C

LeerI2C1

LeerI2C2

getcl2C

getcl2C1

getcl2C2

Función: Lea un solo byte del bus I2Cx .

Incluir: i2c.h

Prototipo: carácter sin firmar Readl2C (vacío); carácter sin
firmar Readl2C1 (vacío); carácter sin firmar
Readl2C2 (vacío); char sin firmar getcl2C (vacío);
char sin firmar getcl2C1 (vacío); char sin firmar
getcl2C2 (vacío);

Observaciones: Esta función lee un solo byte del bus I2Cx . getcl2Cx se define como
Readl2Cx en i2c.h.

Valor de retorno: El byte de datos leído del bus I2Cx .

LeerI2C**LeerI2C1****LeerI2C2****getI2C****getI2C1****getI2C2 (Continuación)**

Nombre del archivo: i2c_read.c
 i2c1read.c
 i2c2read.c #
 definir en i2c.h # definir en
 i2c.h # definir en i2c.h

Ejemplo de código: valor de carácter sin firmar;
 valor = LeerI2C();

ReiniciarI2C**ReiniciarI2C1****ReiniciarI2C2**

Función: Genere una condición de reinicio del bus I2Cx .
Incluir: i2c.h
Prototipo: vacío ReiniciarI2C (vacío); vacío
 ReiniciarI2C1 (vacío); vacío ReiniciarI2C2
 (vacío);
Observaciones: Esta función genera una condición de reinicio del bus I2Cx .
Nombre del archivo: i2c_rstr.c
 i2c1rstr.c
 i2c2rstr.c

InicioI2C**InicioI2C1****InicioI2C2**

Función: Genere una condición de inicio de bus I2Cx .
Incluir: i2c.h
Prototipo: vacío StartI2C (vacío); vacío
 StartI2C1 (vacío); vacío StartI2C2
 (vacío);
Observaciones: Esta función genera una condición de inicio de bus I2Cx .
Nombre del archivo: i2c_inicio.c i2c1inicio.c i2c2inicio.c

Bibliotecas del compilador MPLAB® C18 C

DetenerI2C

DetenerI2C1

DetenerI2C2

Función: Generar condición de parada de bus I2Cx .

Incluir: i2c.h

Prototipo: vacío StopI2C (vacío); vacío
StopI2C1(vacío); vacío
StopI2C2(vacío);

Observaciones: Esta función genera una condición de parada del bus I2Cx .

Nombre del archivo: i2c_stop.c i2c1stop.c i2c2stop.c

escribirI2C

EscribirI2C1

EscribirI2C2

putcl2C

putcl2C1

putcl2C2

Función: Escriba un solo byte en el dispositivo de bus I2Cx .

Incluir: i2c.h

Prototipo: char sin firmar Writel2C(char sin
firmar **data_out**); char sin firmar
Writel2C1(char sin firmar **data_out**); char sin
firmar Writel2C2(char sin firmar
data_out); putcl2C del carácter sin firmar
(**data_out** del carácter sin firmar) ;
putcl2C1 del carácter sin firmar (**data_out** del
carácter sin firmar) ; putcl2C2 del
carácter sin firmar (**data_out** del carácter sin
firmar) ;

Argumentos:

salida_datos

Un solo byte de datos para escribir en el dispositivo de bus I2Cx .

Observaciones:

Esta función escribe un solo byte de datos en el dispositivo de bus I2Cx .
putcl2Cx se define como Writel2Cx en i2c.h.

Valor de retorno:

0 si la escritura fue exitosa -1 si
hubo una colisión de escritura

Nombre del archivo:

i2c_write.c
i2c1write.c
i2c2write.c
#definir en i2c.h #definir
en i2c.h #definir en i2c.h

Ejemplo de código: EscribirI2C('a');

2.4.2 Descripciones de las funciones de la interfaz del dispositivo de memoria EE

EEAckPolling

EEAckPolling1

EEAckPolling2

Función: Genere la secuencia de sondeo de reconocimiento para los dispositivos de memoria Microchip EE I2C.

Incluir: i2c.h

Prototipo: carácter sin firmar EEAckPolling(
control de caracteres sin firmar);
 carácter sin firmar EEAckPolling1(
control de caracteres sin firmar);
 carácter sin firmar EEAckPolling2(
control de caracteres sin firmar);

Argumentos: **control**

Byte de dirección de selección de dispositivo de bus/control de EEPROM.

Observaciones: Esta función se utiliza para generar la secuencia de sondeo de reconocimiento para dispositivos de memoria EE I2C que utilizan sondeo de reconocimiento.

Valor de retorno: 0 si no hubo errores -1 si
 hubo un error de colisión de bus -3 si
 hubo un error de colisión de escritura

Nombre del archivo: i2c_ecap.c
 i2c1ecap.c
 i2c2ecap.c

Ejemplo de código: temperatura = EEAckPolling(0xA0);

EEByteWrite

EEByteWrite1

EEByteWrite2

Función: Escriba un solo byte en el bus I2Cx .

Incluir: i2c.h

Prototipo: char sin firmar EEByteWrite(
control de caracteres sin firmar ,
dirección de caracteres sin
 firmar , **datos** de caracteres sin
 firmar); char sin firmar EEByteWrite1
 (**control de** char sin firmar ,
dirección de char sin firmar ,
datos de char sin firmar); char sin
 firmar EEByteWrite2 (**control de** char sin
 firmar , **dirección de** char sin
 firmar , **datos de** char sin firmar);

Argumentos: **control**

EEPROM control / bus dispositivo seleccionar byte de dirección.

dirección Ubicación de la dirección interna de la EEPROM.

datos Datos para escribir en la dirección EEPROM especificada
 en la dirección del parámetro de función.

Bibliotecas del compilador MPLAB® C18 C

EEByteWrite

EEByteWrite1

EEByteWrite2 (continuación)

Observaciones:	Esta función escribe un solo byte de datos en el bus I2Cx . Esta rutina se puede utilizar para cualquier dispositivo de memoria Microchip I2C EE que requiera solo 1 byte de información de dirección.
Valor de retorno:	0 si no hubo errores -1 si hubo un error de colisión de bus -2 si hubo un error de NOT ACK -3 si hubo un error de colisión de escritura
Nombre del archivo:	i2c_ecbw.c i2c1ecbw.c i2c2ecbw.c
Ejemplo de código:	temperatura = EEByteWrite (0xA0, 0x30, 0xA5);

EEActualAñadirLectura

EEActualAñadirLectura1

EECurrentAddRead2

Función:	Lea un solo byte del bus I2Cx .
Incluir:	i2c.h
Prototipo:	int sin firmar EECurrentAddRead (control de caracteres sin firmar); int sin firmar EECurrentAddRead1 (control de caracteres sin firmar); int sin firmar EECurrentAddRead2 (control de caracteres sin firmar);
Argumentos:	control EEPROM control / bus dispositivo seleccionar byte de dirección.
Observaciones:	Esta función lee un solo byte del bus I2Cx . La ubicación de la dirección de los datos para leer es la del puntero actual dentro del dispositivo I2C EE. El dispositivo de memoria contiene un contador de direcciones que mantiene la dirección de la última palabra a la que se accedió, incrementada en uno.
Valor de retorno:	-1 si se produjo un error de colisión de bus -2 si se produjo un error de NOT ACK -3 si se produjo un error de colisión de escritura De lo contrario, el resultado se devuelve como una cantidad de 16 bits sin signo. Dado que el búfer en sí tiene solo 8 bits de ancho, esto significa que el byte más significativo será cero y el byte menos significativo contendrá el contenido del búfer de lectura.
Nombre del archivo:	i2c_eecr.c i2c1eecr.c i2c2eecr.c
Ejemplo de código:	temp = EEActualAddRead(0xA1);

EEPAGEWRITE

EEPAGEWRITE1

EEPAGEWRITE2

Función: Escriba una cadena de datos en el dispositivo EE desde el bus I2Cx .

Incluir: i2c.h

Prototipo: char sin firmar EEPAGEWRITE(
 control de caracteres sin firmar ,
 dirección de caracteres sin firmar ,
 carácter sin firmar * wrptr);
 EEPAGEWRITE1 (**control de char sin firmar** ,
 dirección de char sin firmar , char sin
 firmar * **wrptr**); char sin firmar
 EEPAGEWRITE2 (**control de char sin firmar** ,
 dirección de char sin firmar , char sin firmar *
 wrptr);

Argumentos: **control**

Byte de dirección de selección de dispositivo de bus/control de EEPROM. **dirección** Ubicación de la dirección interna de la EEPROM. **wrptr** Puntero de tipo de carácter en PICmicro RAM. Los objetos de datos señalados por wrptr se escribirán en el dispositivo EE.

Observaciones: Esta función escribe una cadena de datos terminada en nulo en el dispositivo de memoria I2C EE. El carácter nulo en sí no se transmite. 0 si

Valor de retorno: no hubo errores -1 si hubo un error de colisión de bus -2 si hubo un error de NOT ACK -3 si hubo un error de colisión de escritura

Nombre del archivo: i2c_eepw.c
i2c1eepw.c
i2c2eepw.c

Ejemplo de código: temp = EEPAGEWRITE(0xA0, 0x70, wrptr);

Bibliotecas del compilador MPLAB® C18 C

EERandomRead

EERandomRead1

EERandomRead2

Función: Lea un solo byte del bus I2Cx .

Incluir: i2c.h

Prototipo: unsigned int EERandomRead(

control de caracteres sin firmar ,
dirección de caracteres sin firmar);

unsigned int EERandomRead1 (**control de**
caracteres sin firmar, **dirección de**
caracteres sin firmar);

int sin firmar EERandomRead2 (**control de**
caracteres sin firmar, **dirección de**
caracteres sin firmar);

Argumentos: **control**

Byte de dirección de selección de dispositivo de bus/control de EEPROM.

dirección

Ubicación de la dirección interna de la EEPROM.

Observaciones: Esta función lee un solo byte del bus I2Cx . La rutina se puede utilizar para dispositivos de memoria Microchip I2C EE que solo requieren 1 byte de información de dirección.

Valor de retorno: El valor de retorno contiene el valor leído en el byte menos significativo y la condición de error en el byte más significativo. La condición de error es: -1 si hubo un error de colisión de bus -2 si hubo un error de NOT ACK -3 si hubo un error de colisión de escritura

Nombre del archivo: i2c_eerr.c
i2c1eerr.c
i2c2eerr.c

Ejemplo de código: temperatura interna sin
firmar; temperatura = EERandomRead(0xA0,0x30);

EESequentialRead

EESequentialRead1

EESequentialRead2

Función: Lea una cadena de datos del bus I2Cx .

Incluir: i2c.h

Prototipo:

```
char sin firmar EESequentialRead ( control de
    char sin firmar , dirección de char sin
    firmar, char sin firmar * rdptr, longitud
    de char sin firmar ); char sin firmar
    EESequentialRead1 ( control de char sin
    firmar , dirección de char sin firmar, char sin
    firmar * rdptr, longitud de char sin firmar );
    char sin firmar EESequentialRead2
    ( control de char sin firmar , dirección de
    char sin firmar, char sin firmar * rdptr,
    longitud de char sin firmar );
```

Argumentos:**control**

EEPROM control / bus dispositivo seleccionar byte de dirección. **dirección** Ubicación de la dirección interna de la EEPROM. **rdptr** Puntero de tipo de carácter al área PICmicro

RAM para la colocación de datos leídos desde el dispositivo

EEPROM. **longitud** Número de bytes para leer desde el dispositivo EEPROM.

Observaciones:

Esta función lee una longitud de cadena predefinida de datos del bus I2Cx . La rutina se puede utilizar para dispositivos de memoria Microchip I2C EE que solo requieren 1 byte de información de dirección. 0 si no hubo errores -1 si hubo un

Valor de retorno:

error de colisión de bus -2 si hubo un error de NOT ACK -3 si hubo un error de colisión de escritura

Nombre del archivo:

i2c_eesr.c

i2c1eesr.c

i2c2eesr.c

Ejemplo de código:

error de carácter sin firmar;

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

Bibliotecas del compilador MPLAB® C18 C

2.4.3 Ejemplo de uso El

siguiente es un ejemplo de código simple que ilustra el módulo SSP configurado para la comunicación maestra I2C. La rutina ilustra las comunicaciones I2C con un dispositivo de memoria Microchip 24LC01B I2C EE.

```
#incluir "p18cxx.h" #incluir
"i2c.h"

matriz de caracteres sin firmarwr[] = {1,2,3,4,5,6,7,8,0}; matriz de caracteres
sin firmar [20];

/********************* *** vacío principal (vacío) {

OpenI2C(MASTER, SLEW_ON); // Inicializa el módulo I2C //400kHz Baud
SSPADD = 9; // clock(9) @16MHz
Reloj en baudios de 100kHz(39) @16MHz

mientras(1)
{
    EEByteWrite(0xA0, 0x30, 0xA5);
    EEAckPolling(0xA0);
    EEActualAddRead(0xA0);
    EEPageWrite(0xA0, 0x70, arraywr);
    EEAckPolling(0xA0);
    EESequentialRead(0xA0, 0x70, arrayrd, 20);
    EELectura aleatoria (0xA0,0x30);
}
}
```

2.5 FUNCIONES DEL PUERTO DE E/S

PORTB es compatible con las siguientes funciones:

TABLA 2-6: FUNCIONES DEL PUERTO DE E/S

Función	Descripción
CerrarPORTB	Deshabilite las interrupciones y las resistencias pull-up internas para PORTB.
CerrarRBxINT	Deshabilite las interrupciones para el pin PORTB x .
Deshabilitar Pullups	Deshabilite las resistencias pull-up internas en PORTB.
Habilitar pullups	Habilite las resistencias pull-up internas en PORTB.
AbrirPORTB	Configure las interrupciones y las resistencias pull-up internas en PORTB.
AbrirRBxINT	Habilite las interrupciones para el pin PORTB x .

2.5.1 Descripciones de funciones

CerrarPORTB

Función:	Deshabilite las interrupciones y las resistencias pull-up internas para PORTB.
Incluir:	portb.h
Prototipo:	void CerrarPORTB(void);
Observaciones:	Esta función desactiva la interrupción en el cambio de PORTB y las resistencias pull-up internas.
Nombre del archivo:	pbclose.c

CerrarRB0INT

CerrarRB1INT

CerrarRB2INT

Función:	Deshabilite las interrupciones para el pin PORTB especificado.
Incluir:	portb.h
Prototipo:	vacio CerrarRB0INT(vacío); vacío CerrarRB1INT (vacío); vacío CerrarRB2INT (vacío);
Observaciones:	Esta función desactiva la interrupción por cambio de PORTB.
Nombre del archivo:	rb0cerrar.c rb1cerrar.c rb2cerrar.c

Deshabilitar Pullups

Función:	Deshabilite las resistencias pull-up internas en PORTB.
Incluir:	portb.h
Prototipo:	vacio DisablePullups (vacío);
Observaciones:	Esta función desactiva las resistencias pull-up internas en PORTB.

Habilitar pullups

Función:	Habilite las resistencias pull-up internas en PORTB.
Incluir:	portb.h
Prototipo:	void EnablePullups(void);
Observaciones:	Esta función habilita las resistencias pull-up internas en PORTB.
Nombre del archivo:	pullen.c

Bibliotecas del compilador MPLAB® C18 C

AbrirPORTB

Función:	Configure las interrupciones y las resistencias pull-up internas en PORTB.	
Incluir:	portb.h	
Prototipo:	void OpenPORTB (configuración de caracteres sin firmar);	
Argumentos:	config Una máscara de bits que se crea realizando una operación AND bit a bit ('&') con un valor de cada una de las categorías enumeradas a continuación. Estos valores se definen en el archivo portb.h.	
Interrupción al cambiar:		
	PORTB_CHANGE_INT_ON	Interrupción habilitada
	PORTB_CHANGE_INT_OFF	Interrupción deshabilitada
Habilitar pullups:	PORTB_PULLUPS_ON	
	PORTB_PULLUPS_OFF	resistencias pull-up habilitadas resistencias pull-up deshabilitadas
Observaciones:	Esta función configura las interrupciones y las resistencias pull-up internas en PORTB.	
Nombre del archivo:	pbopen.c	
Ejemplo de código:	AbrirPORTB(PORTB_CHANGE_INT_ON & PORTB_PULLUPS_ON);	

AbrirRB0INT

AbrirRB1INT

OpenRB2INT

Función:	Habilite las interrupciones para el pin PORTB especificado.	
Incluir:	portb.h	
Prototipo:	void OpenRB0INT (configuración de caracteres sin firmar); void OpenRB1INT (configuración de caracteres sin firmar); void OpenRB2INT (configuración de caracteres sin firmar);	
Argumentos:	config Una máscara de bits que se crea realizando una operación AND bit a bit ('&') con un valor de cada una de las categorías enumeradas a continuación. Estos valores se definen en el archivo portb.h.	
Interrupción en cambio:		
	PORTB_CHANGE_INT_ON	Interrupción habilitada
	PORTB_CHANGE_INT_OFF	Interrupción deshabilitada
Interrupción en flanco:	RISING_EDGE_INT FALLING_EDGE_INT	
Habilitar pullups:	PORTB_PULLUPS_ON	Interrupción en flanco ascendente
	PORTB_PULLUPS_OFF	Esta función configura la interrupción en flanco descendente
	interrupciones y las resistencias pull-up internas en PORTB.	
		resistencias pull-up habilitadas resistencias pull-up deshabilitadas

Observaciones:

Nombre del archivo:	rb0open.c
	rb1open.c
	rb2open.c
Ejemplo de código:	OpenRB0INT(PORTB_CHANGE_INT_ON & RISING_EDGE_INT & PORTB_PULLUPS_ON);

Funciones de periféricos de hardware

2.6 FUNCIONES DEL MICROALAMBRE

Las siguientes rutinas se proporcionan para dispositivos con un solo periférico Microwire:

TABLA 2-7: FUNCIONES DEL PERIFÉRICO DE MICROALAMBRE ÚNICO

Función	Descripción
CerrarMwire	Deshabilite el módulo SSP utilizado para la comunicación Microwire.
DatosRdyMwire	Indica la finalización del ciclo de escritura interna.
obtenercMwire	Lea un byte del dispositivo Microwire.
obtieneMwire	Lea una cadena del dispositivo Microwire.
OpenMwire	Configure el módulo SSP para el uso de Microwire.
ponercMwire	Escriba un byte en el dispositivo Microwire.
ReadMwire	Lea un byte del dispositivo Microwire.
EscribirMwire	Escriba un byte en el dispositivo Microwire.

Las siguientes rutinas se proporcionan para dispositivos con múltiples periféricos Microwire:

TABLA 2-8: FUNCIONES DE MÚLTIPLES PERIFÉRICOS DE MICROALAMBRE

Función	Descripción
CerrarMwirex	Deshabilite el módulo SSPx utilizado para la comunicación Microwire.
DatosRdyMwirex	Indica la finalización del ciclo de escritura interna.
obtenercMwirex	Lea un byte del dispositivo Microwire.
obtieneMwirex	Lea una cadena del dispositivo Micowire.
AbrirMwirex	Configure el módulo SSPx para el uso de Microwire.
putcMwirex	Escriba un byte en el dispositivo Micowire.
LeerMwirex	Lea un byte del dispositivo Micowire.
EscribirMwirex	Escriba un byte en el dispositivo Micowire.

2.6.1 Descripciones de funciones

CerrarMwire

CerrarMwire1

CerrarMwire2

Función:	Deshabilite el módulo SSPx.
Incluir:	mwire.h
Prototipo:	void CloseMwire(void); vacío CloseMwire1 (vacío); vacío CloseMwire2 (vacío);
Observaciones:	Pin I/O regresa bajo el control de la configuración de registro TRISC y LATC.
Nombre del archivo:	mw_close.c mw1close.c mw2close.c

Bibliotecas del compilador MPLAB® C18 C

DatosRdyMwire

DatosRdyMwire1

DatosRdyMwire2

Función: Indique si el dispositivo Microwirex ha completado el ciclo de escritura interna.

Incluir: mwire.h

Prototipo: char sin firmar DataRdyMwire(void); carácter sin firmar DataRdyMwire1(void); carácter sin firmar DataRdyMwire2(void);

Observaciones: Determina si el dispositivo Microwirex está listo.

Valor de retorno: 1 si el dispositivo Microwirex está listo 0 si el ciclo de escritura interna no está completo o si ocurrió un error de bus

Nombre del archivo: mw_drdy.c
mw1drdy.c
mw2drdy.c

Ejemplo de código: while (!DataRdyMwire());

getcMwire

getcMwire1

getcMwire2

getcMwirex se define como ReadMwirex. Consulte [ReadMwirex](#).

obtieneMwire

obtieneMwire1

obtieneMwire2

Función: Lea una cadena del dispositivo Microwirex.

Incluir: mwire.h

Prototipo: void getsMwire(unsigned char * rdptr, unsigned char length); void getsMwire1(caracter sin firmar * rdptr, longitud del carácter sin firmar); void getsMwire2(caracter sin firmar * rdptr, longitud del carácter sin firmar);

Argumentos: **rdptr**

Puntero a PICmicro RAM para la colocación de datos leídos desde el dispositivo Microwirex . **longitud** Número de bytes para leer del dispositivo Microwirex.

Observaciones: Esta función se utiliza para leer una longitud predeterminada de datos de un dispositivo Microwirex. Antes de usar esta función, se debe emitir un comando Readx con la dirección adecuada.

Nombre del archivo: mw_gets.c
mw1gets.c
mw2gets.c

Ejemplo de código: char sin firmar arryrd[LONGITUD];
putcMwire(LEER); putcMwire(dirección);
getsMwire(matriz, LONGITUD);

OpenMwire

AbrirMwire1

OpenMwire2

Función: Configure el módulo SSPx .

Incluir: mwire.h

Prototipo: void OpenMwire (modo

de sincronización de caracteres sin **firmar**);

Argumentos: **sync_mode**

Uno de los siguientes valores definidos en mwire.h: reloj =

MWIRE_FOSC_4 reloj = FOSC/16 MWIRE_FOSC_16

MWIRE_FOSC_64 reloj = TMR2 salida/reloj = FOSC/64

OpenMwirex restablece el módulo SSPx al uso de FOSC_TMR2

para comunicaciones Microwire. Luego configura el módulo

Observaciones:

Nombre del archivo: mw_open.c

mw1open.c

mw2open.c

Ejemplo de código: OpenMwire(MWIRE_FOSC_16);

putcMwire

putcMwire1

putcMwire2

putcMwirex se define como WriteMwirex. Consulte **WriteMwirex**.

Bibliotecas del compilador MPLAB® C18 C

ReadMwire

ReadMwire1

ReadMwire2

getcMwire

getcMwire1

getcMwire2

Función: Leer un byte de un dispositivo Microwirex .

Incluir: mwire.h

Prototipo: carácter sin firmar ReadMwire(

 byte_alto de caracteres sin firmar ,

byte_bajo de caracteres sin firmar);

carácter sin firmar ReadMwire1(

 byte_alto de caracteres sin firmar ,

byte_bajo de caracteres sin firmar);

carácter sin firmar ReadMwire2(

 byte_alto de caracteres sin firmar ,

byte_bajo de caracteres sin firmar);

carácter sin firmar getcMwire(

 byte_alto de caracteres sin firmar ,

byte_bajo de caracteres sin firmar);

carácter sin firmar getcMwire1(

 byte_alto de caracteres sin firmar ,

byte_bajo de caracteres sin firmar);

carácter sin firmar getcMwire2(

 byte_alto de caracteres sin firmar ,

byte_bajo de caracteres sin firmar);

Argumentos: **high_byte**

Primer byte de la palabra de instrucción de

16 bits. **low_byte** Segundo byte de la palabra

de instrucción de 16 bits.

Observaciones: Esta función lee un solo byte de un dispositivo Microwirex. El bit de inicio, el código

de operación y la dirección componen los bytes altos y bajos que se pasan a esta

función. getcMwire se define como ReadMwirex en mwire.h.

Valor de retorno: El valor devuelto es el byte de datos leído del dispositivo Microwirex. mw_read.c

Nombre del archivo: mw1read.c mw2read.c #definir en mwire.h #definir en mwire.h #definir en mwire.h

Ejemplo de código: ReadMwire(0x03, 0x00);

EscribirMwire**EscribirMwire1****EscribirMwire2****putcMwire****putcMwire1****putcMwire2**

Función: Esta función se utiliza para escribir un solo byte de datos (un carácter).

Incluir: mwire.h

Prototipo:

```
char sin firmar WriteMwire(
    data_out de caracteres sin firmar );
char sin firmar WriteMwire1( char sin firmar
    data_out ); char sin firmar
WriteMwire2( char sin firmar data_out );

carácter sin firmar putcMwire(
    data_out de caracteres sin firmar );
carácter sin firmar putcMwire1(
    data_out de caracteres sin firmar );
carácter sin firmar putcMwire2(
    data_out de caracteres sin firmar );
```

Argumentos: **data_out** Un solo byte de datos para escribir en el dispositivo Microwirex.

Observaciones: Esta función escribe un solo byte de datos en un dispositivo Microwirex que utiliza el módulo SSPx . putcMwirex se define como WriteMwirex en mwire.h.

Valor de retorno: 0 si la escritura fue exitosa -1 si hubo una colisión de escritura

Nombre del archivo:

```
mw_write.c
mw1write.c
mw2write.c
#define en mwire.h #define
en mwire.h #define en mwire.h
```

Ejemplo de código: WriteMwire(0x55);

Bibliotecas del compilador MPLAB® C18 C

2.6.2 Ejemplo de uso

El siguiente es un ejemplo de código simple que ilustra el módulo SSP que se comunica con un dispositivo de memoria Microchip 93LC66 Microwire EE.

```
#incluye "p18cxx.h" #incluye
"mwire.h"

// 93LC66x8
// FUNCIÓN Prototipos void main(void);
vacío ew_enable (vacío); vacío
borrar_todo (vacío); vacío
ocupado_encuesta (vacío); void
write_all (datos de caracteres sin
firmar); void byte_read (dirección de caracteres sin firmar);
void read_mult(dirección de carácter sin firmar, carácter sin firmar
*rdptr, longitud de carácter sin firmar); void write_byte (dirección
de caracteres sin firmar,
datos de caracteres sin firmar);

// Definiciones de VARIABLE unsigned
char arrayd[20]; var char sin firmar;

// DEFINE 93LC66 MACROS -- vea la hoja de datos para más detalles
#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 LATCs #define

vacío principal (vacío)
{
    TRISCbits.TRISC2 = 0; W_CS =
0; //asegurarse de que CS sea negado OpenMwire(MWIRE_FOSC_16); //
habilitar periférico SSP ew_enable(); //enviar borrar/mwireremote(0x10,0x34); //escribir byte
(dirección, datos) busy_poll(); nop(); lecturabusyread(0x10,0x34,0xd0)byleer direcciones
bytes erase_all(); //borra toda la matriz CloseMwire(); //deshabilitar periférico SSP
}
```

Funciones de periféricos de hardware

```

vacío ew_enable (vacío) {

    W_CS = 1;           //afirmar selección de chip
    putcMable(EWEN1); //habilitar comando de escritura byte 1
    putcMwire(EWEN2); //habilitar comando de escritura byte 2 //negar selección
    W_CS = 0;           de chip

} vacío ocupado_encuesta (vacío)
{
    W_CS = 1;
    while(! DataRdyMwire() ); W_CS = 0;

}

void write_byte (dirección de caracteres sin firmar,
                 datos de caracteres sin firmar)
{
    W_CS = 1;
    putcMwire(ESCRIBIR);      //escribir comando
    putcMwire(dirección); //dirección
    putcMwire(datos); // escribe un solo byte
    W_CS = 0;
}

void byte_read (dirección de caracteres sin firmar) {

    W_CS = 1;
    getcMwire(LEER,dirección); //leer un byte
    W_CS = 0;
}

void read_mult(dirección de caracteres sin firmar,
               carácter sin firmar *rdptr, longitud
               de carácter sin firmar)
{
    W_CS = 1;
    putcMwire(LEER); //leer comando putcMwire(dirección); //
    dirección (A7 - A0) getsMwire(rdptr, longitud); //leer varios bytes

    W_CS = 0;
}

anular borrar_todo (anular) {

    W_CS = 1;
    putcMable(ERAL1); //borra todo el byte de comando 1
    putcMwire(ERAL2); // borrar todo el byte de comando 2
    W_CS = 0;
}

```

Bibliotecas del compilador MPLAB® C18 C

2.7 FUNCIONES DE MODULACIÓN POR ANCHO DE IMPULSO

El periférico PWM es compatible con las siguientes funciones:

TABLA 2-9: FUNCIONES PWM

Función	Descripción
CerrarPWMx	Deshabilite el canal PWM x.
AbiertoPWMx	Configure el canal PWM x.
EstablecerDCPWMx	Escriba un nuevo valor de ciclo de trabajo en el canal PWM x.
Establecer Salida	Establece los bits de configuración de salida PWM para ECCP x.
PWMx CerrarEPWMx (1)	Deshabilite el canal PWM mejorado x.
AbrirEPWMx(1)	Configure el canal x de PWM mejorado.
EstablecerDCEPWMx(1)	Escriba un nuevo valor de ciclo de trabajo en el canal x de PWM mejorado .
Establecer SalidaEPWMx(1)	Establece los bits de configuración de salida PWM mejorados para ECCP x.

Nota 1: Las funciones PWM mejoradas solo están disponibles en aquellos dispositivos con un registro ECCPxCON.

2.7.1 Descripciones de funciones

CerrarPWM1

CerrarPWM2

CerrarPWM3

CerrarPWM4

CerrarPWM5

CerrarEPWM1

Función: Deshabilitar el canal PWM.

Incluir: pwm.h

Prototipo: vacío ClosePWM1 (vacío); vacío
ClosePWM2 (vacío); vacío ClosePWM3
(vacío); vacío ClosePWM4 (vacío); vacío
ClosePWM5 (vacío); vacío CerrarEPWM1
(vacío);

Observaciones: Esta función desactiva el canal PWM especificado.

Nombre del archivo:
pw1cerrar.c
pw2cerrar.c
pw3cerrar.c
pw4cerrar.c
pw5cerrar.c
ew1cerrar.c

AbiertoPWM1**AbiertoPWM2****AbiertoPWM3****AbiertoPWM4****AbiertoPWM5****AbrirEPWM1**

Función:	Configurar el canal PWM.
Incluir:	pwm.h
Prototipo:	void OpenPWM1 (período de caracteres); void OpenPWM2 (período de caracteres); void OpenPWM3 (período de caracteres); void OpenPWM4 (período de caracteres); void OpenPWM5 (período de caracteres); void OpenEPWM1 (período de caracteres);
Argumentos:	período Puede ser cualquier valor de 0x00 a 0xff. Este valor determina la frecuencia de PWM utilizando la siguiente fórmula: Período de PWM = [(período) + 1] x 4 x TOSC x preescalador TMR2
Observaciones:	Esta función configura el canal PWM especificado para el período y la base de tiempo. PWM usa solo Timer2. Además de abrir el PWM, Timer2 también debe abrirse con una instrucción OpenTimer2(...) antes de que funcione el PWM.
Nombre del archivo:	pw1open.c pw2open.c pw3open.c pw4open.c pw5open.c ew1open.c
Ejemplo de código:	AbrirPWM1(0xff);

Bibliotecas del compilador MPLAB® C18 C

EstablecerDCPWM1**EstablecerDCPWM2****EstablecerDCPWM3****ConjuntoDCPWM4****EstablecerDCPWM5****EstablecerDCEPWM1**

Función: Escriba un nuevo valor de ciclo de trabajo en los registros de ciclo de trabajo del canal PWM especificado.

Incluir: pwm.h

Prototipo: void SetDCPWM1 (ciclo de trabajo int sin **firmar**); void SetDCPWM2 (ciclo de trabajo int sin **firmar**); void SetDCPWM3 (ciclo de trabajo int sin **firmar**); void SetDCPWM4 (ciclo de trabajo int sin **firmar**); void SetDCPWM5 (ciclo de trabajo int sin **firmar**); void SetDCEPWM1 (ciclo de trabajo int sin **firmar**);

Argumentos: **dutycycle** El valor de dutycycle puede ser cualquier número de 10 bits. Solo los 10 bits inferiores del ciclo de trabajo se escriben en los registros del ciclo de trabajo. El ciclo de trabajo, o más específicamente el tiempo alto de la forma de onda PWM, se puede calcular a partir de la siguiente fórmula:

$$\text{PWM} \times \text{Ciclo de trabajo} = (\text{DCx}<9:0>) \times \text{TOSC}$$

donde DCx<9:0> es el valor de 10 bits especificado en la llamada a esta función.

Observaciones: Esta función escribe el nuevo valor para el ciclo de trabajo en los registros de ciclo de trabajo del canal PWM especificado.

La resolución máxima de la forma de onda PWM se puede calcular a partir del período utilizando la siguiente fórmula:

$$\text{Resolución (bits)} = \text{registro (FOSC/Fpwm) / registro (2)}$$

Nombre del archivo:
pw1setdc.c
pw2setdc.c
pw3setdc.c
pw4setdc.c
pw5setdc.c
ew1setdc.c

Ejemplo de código: EstablecerDCPWM1(0);

Funciones de periféricos de hardware

EstablecerSalidaPWM1 EstablecerSalidaPWM2 EstablecerSalidaPWM3 EstablecerSalidaEPWM1

Función:	Establece los bits de configuración de salida PWM para CCP.						
Incluir:	pwm.h						
Prototipo:	<pre>void SetOutputPWM1 (configuración de salida de caracteres sin firmar , modo de salida de caracteres sin firmar); void SetOutputPWM2 (configuración de salida de caracteres sin firmar , modo de salida de caracteres sin firmar); void SetOutputPWM3 (configuración de salida de caracteres sin firmar , modo de salida de caracteres sin firmar); void SetOutputEPWM1 (configuración de salida de caracteres sin firmar , modo de salida de caracteres sin firmar);</pre>						
Argumentos:	<p>outputconfig El valor de outputconfig puede ser cualquiera de los siguientes valores (definidos en pwm.h): SINGLE_OUT FULL_OUT_FWD HALF_OUT FULL_OUT_REV</p> <p>outputmode El valor de outputmode puede ser cualquiera de los siguientes valores (definidos en pwm.h): PWM_MODE_1 puente completo hacia adelante salida de medio puente salida de puente completo inversa</p>						
	<table border="0"> <tr> <td>PWM_MODE_2</td> <td>P1A y P1C activo-alto, P1B y P1D activo-alto</td> </tr> <tr> <td>PWM_MODE_3</td> <td>P1A y P1C activo-bajo, P1B y P1D activo-alto</td> </tr> <tr> <td>PWM_MODE_4</td> <td>P1A y P1C activo-bajo, P1B y P1D activo-bajo</td> </tr> </table>	PWM_MODE_2	P1A y P1C activo-alto, P1B y P1D activo-alto	PWM_MODE_3	P1A y P1C activo-bajo, P1B y P1D activo-alto	PWM_MODE_4	P1A y P1C activo-bajo, P1B y P1D activo-bajo
PWM_MODE_2	P1A y P1C activo-alto, P1B y P1D activo-alto						
PWM_MODE_3	P1A y P1C activo-bajo, P1B y P1D activo-alto						
PWM_MODE_4	P1A y P1C activo-bajo, P1B y P1D activo-bajo						
Observaciones:	Esto solo se aplica a aquellos dispositivos con Extended o Enhanced CCP (ECCP).						
Nombre del archivo:	pw1setoc.c pw2setoc.c pw3setoc.c ew1setoc.c						
Ejemplo de código:	SetOutputPWM1 (SINGLE_OUT, PWM_MODE_1);						

Bibliotecas del compilador MPLAB® C18 C

2.8 FUNCIONES SPI™

Las siguientes rutinas se proporcionan para dispositivos con un único periférico SPI:

TABLA 2-10: FUNCIONES DEL PERIFÉRICO SPI SIMPLE

Función	Descripción
CerrarSPI	Deshabilite el módulo SSP utilizado para las comunicaciones SPI.
DatosRdySPI	Determinar si un nuevo valor está disponible desde el búfer SPI.
obtenercSPI	Leer un byte del bus SPI.
obtieneSPI	Leer una cadena del bus SPI.
OpenSPI	Inicialice el módulo SSP utilizado para las comunicaciones SPI.
putcSPI	Escribe un byte en el bus SPI.
poneSPI	Escriba una cadena en el bus SPI.
Leer SPI	Leer un byte del bus SPI.
Escribir SPI	Escribe un byte en el bus SPI.

Las siguientes rutinas se proporcionan para dispositivos con múltiples periféricos SPI:

TABLA 2-11: MÚLTIPLES FUNCIONES DEL PERIFÉRICO SPI

Función	Descripción
CerrarSPIX	Deshabilite el módulo SSPx utilizado para las comunicaciones SPI.
DatosRdySPIX	Determinar si un nuevo valor está disponible desde el búfer SPIx.
obtenercSPIX	Leer un byte del bus SPIx .
obtieneSPIX	Leer una cadena del bus SPIx .
OpenSPIx	Inicialice el módulo SSPx utilizado para las comunicaciones SPI.
putcSPIx	Escribe un byte en el bus SPIx .
poneSPIx	Escriba una cadena en el bus SPIx .
LeerSPIX	Leer un byte del bus SPIx .
EscribirSPIX	Escribe un byte en el bus SPIx .

2.8.1 Descripciones de funciones

CerrarSPI

CerrarSPI1

CerrarSPI2

Función:	Deshabilite el módulo SSPx.
Incluir:	spi.h
Prototipo:	vacio CloseSPI (vacío); vacio CloseSPI1 (vacío); vacio CloseSPI2 (vacío);
Observaciones:	Esta función desactiva el módulo SSPx . Pin I/O regresa bajo el control de los registros TRIS y LAT apropiados. spi_clos.c spi1clos.c spi2clos.c
Nombre del archivo:	

DatosRdySPI

DatosRdySPI1

DatosRdySPI2

Función:	Determine si el SSPBUFx contiene datos.
Incluir:	spi.h
Prototipo:	char sin firmar DataRdySPI(void); carácter sin firmar DataRdySPI1(void); carácter sin firmar DataRdySPI2(void);
Observaciones:	Esta función determina si hay un byte para leer del registro SSPBUFx.
Valor de retorno:	0 si no hay datos en el registro SSPBUFx 1 si hay datos en el registro SSPBUFx spi_dtrd.c spi1dtrd.c
Nombre del archivo:	spi2dtrd.c

Ejemplo de código: mientras (!DataRdySPI());

getcSPI

getcSPI1

getcSPI2

getcSPIx se define como ReadSPIx. Consulte **ReadSPIX**.

Bibliotecas del compilador MPLAB® C18 C

obtiene**SPI obtiene****SPI1 obtiene SPI2****Función:** Leer una cadena del bus SPIx . spi.h**Incluir:****Prototipo:** void getsSPI(unsigned char *rdptr, unsigned char**length);**

void getsSPI1(carácter sin firmar *rdptr,

longitud de caracteres sin firmar);

void getsSPI2(carácter sin firmar *rdptr,

longitud de caracteres sin firmar);**Argumentos:****rdptr**

Puntero a la ubicación para almacenar datos leídos desde el dispositivo SPIx.

longitud

Número de bytes para leer desde el dispositivo SPIx .

Observaciones:

Esta función lee una longitud de cadena de datos predeterminada del bus SPIx.

Nombre del archivo:

spi_gets.c

spi1gets.c

spi2gets.c

Ejemplo de código:

wrptr de caracteres sin firmar[10];

obtieneSPI(wrptr, 10);

OpenSPI**OpenSPI1****OpenSPI2****Función:** Inicialice el módulo SSPx.**Incluir:** spi.h**Prototipo:** void OpenSPI (modo de sincronización de caracteres sin
firmar , **modo de bus de caracteres**
sin firmar , **fase_smp de caracteres** sin
firmar); void OpenSPI1 (modo de sincronización de caracteres
sin firmar , **modo de bus de caracteres**
sin firmar , **fase_smp de caracteres** sin
firmar); void OpenSPI2 (modo de sincronización de caracteres
sin firmar , **modo de bus de caracteres**
sin firmar , **fase_smp de caracteres** sin
firmar);**Argumentos:** **sync_mode**

Uno de los siguientes valores, definido en spi.h:

SPI_FOSC_4 Modo maestro SPI_FOSC_4 SPI_FOSC_4/16

SPI_FOSC_64 Modo maestro SPI_FOSC_64 SPI_FOSC_64/16

control de clavija /SS habilitado SPI_FOSC_SS_MRA Modo de SPI_FOSC_SS_MRA

control de clavija /SS deshabilitado SLV_SSOFF

bus_mode

Uno de los siguientes valores, definido en spi.h: Configuración

Configuración para el modo de SPI_MODE_0 SPI_MODE_0 MODE_00

Configuración para el modo de SPI_MODE_1 SPI_MODE_1 MODE_10

OpenSPI**OpenSPI1****OpenSPI2 (continuación)**

smp_phase

Uno de los siguientes valores, definido en spi.h: Muestra

SMPEND de datos de entrada al final de la salida de

SMPMID datos Muestra de datos de entrada a la mitad de

Observaciones: la salida de datos Esta función configura el módulo SSPx para su uso con un dispositivo**Nombre del archivo:** de bus SPIx. spi_open.c spi1open.c spi2open.c**Ejemplo de código:** OpenSPI(SPI_FOSC_16, MODE_00, SMPEND);

putcSPI**putcSPI1****putcSPI2**

putcSPIx se define como WriteSPIx. Consulte **WriteSPIX**.

pone SPI**pone SPI1****pone SPI2**

Función: Escriba una cadena en el bus SPIx .**Incluir:** spi.h**Prototipo:** void putsSPI(caracter sin firmar *wrptr); void putsSPI1(unsigned char *wrptr); void putsSPI2(caracter sin firmar *wrptr);**Argumentos:** **wrptr**

Puntero al valor que se escribirá en el bus SPIx.

Observaciones: Esta función escribe una cadena de datos en el dispositivo de bus SPIx . La rutina finaliza leyendo un carácter nulo en la cadena de datos (el carácter nulo no se escribe en el bus). spi_puts.c spi1puts.c spi2puts.c**Nombre del archivo:****Ejemplo de código:** char unsigned wrptr[] = "¡Hola!"; pone SPI(wrptr);

Bibliotecas del compilador MPLAB® C18 C

Leer SPI

LeerSPI1

LeerSPI2

getcSPI

getcSPI1

getcSPI2

Función: Leer un byte del bus SPIx .

Incluir: spi.h

Prototipo: carácter sin firmar ReadSPI (vacío); carácter sin firmar ReadSPI1 (vacío); carácter sin firmar ReadSPI2 (vacío); char sin firmar getcSPI(void); char sin firmar getcSPI1(void); char sin firmar getcSPI2(void);

Observaciones: Esta función inicia un ciclo de bus SPIx para la adquisición de un byte de datos. getcSPIx se define como ReadSPIx en spi.h.

Valor de retorno: Esta función devuelve un byte de datos leídos durante un ciclo de lectura SPIx.

Nombre del archivo: spi_read.c spi1read.c spi2read.c #definir en spi.h #definir en spi.h #definir en spi.h

Ejemplo de código: carácter x;
x = LeerSPI();

Escribir SPI**escribirSPI1****escribirSPI2****putcSPI****putcSPI1****putcSPI2**

Función: Escribe un byte en el bus SPIx .**Incluir:** spi.h**Prototipo:** char sin firmar WriteSPI (char sin
firmar **data_out**); char sin firmar
WriteSPI1(char sin firmar **data_out**); char sin firmar
WriteSPI2(char sin firmar **data_out**);
putcSPI del carácter sin firmar (data_out del carácter sin
firmar) ; putcSPI1 del carácter sin firmar
(data_out del carácter sin firmar) ; putcSPI2 del carácter sin
firmar (data_out del carácter sin firmar) ;**Argumentos:** **data_out**
Valor que se escribirá en el bus SPIx.**Observaciones:** Esta función escribe un solo byte de datos y luego verifica si hay una colisión de escritura. putcSPIx se define como WriteSPIx en spi.h. 0 si no ocurrió ninguna**Valor de retorno:** colisión de escritura -1 si ocurrió una colisión de escritura**Nombre del archivo:** spi_writ.c
spi1writ.c
spi2writ.c
#definir en spi.h #definir
en spi.h #definir en spi.h**Ejemplo de código:** WriteSPI('a');

Bibliotecas del compilador MPLAB® C18 C

2.8.2 Ejemplo de uso

El siguiente ejemplo demuestra el uso de un módulo SSP para comunicarse con un dispositivo de memoria Microchip 25C080 SPI EE.

```
#incluir <p18cxx.h> #incluir
<spi.h>

// FUNCIÓN Prototipos void main(void);
vacío set_wren (vacío); vacío
ocupado_sondeo (vacío); char sin
firmar status_read (vacío); void
status_write (datos de caracteres sin firmar); void
byte_write (caracter sin firmar addhigh, char sin firmar addlow,
datos de char sin firmar);

void page_write(caracter sin firmar addhigh, char sin firmar
                addlow, char sin firmar *wrptr); void
array_read(addhigh de caracteres sin
firmar, addlow de caracteres sin firmar, char *rdptr sin firmar,
recuento de caracteres sin firmar);

byte_read de caracteres sin firmar (añadir alto de caracteres sin firmar,
añadir bajo de caracteres sin firmar);

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

//25C040/080/160 tamaño de escritura de página char
sin firmar arrayrd[16]; var char sin firmar;

#define SPI_CS LATCbits.LATC2

/********************* vacío principal (vacío) {

TRISCbits.TRISC2 = 0; SPI_CS =
1; // asegúrese de que el dispositivo de memoria SPI // Chip Select
    se restablezca OpenSPI (SPI_FOSC_16,
    MODE_00, SMPEND); set_wren(); estado_escribir(0);

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

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

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

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

Funciones de periféricos de hardware

```

        Cerrar SPI();
        mientras(1);
    }

    vacío set_wren (vacío) {

        SPI_CS = 0; var           //afirmar selección de chip
        = putcSPI(SPI_WREN); //enviar comando de activación de escritura //negar selección
        SPI_CS = 1;             de chip
    }

    void page_write (char sin firmar addhigh,
                     carácter sin firmar addlow,
                     carácter sin firmar *wrptr)
    {
        SPI_CS = 0; //afirmar chip select var = putcSPI(SPI_WRITE); //enviar comando
        de escritura var = putcSPI(addhigh); //enviar byte alto de dirección var =
        putcSPI(addlow); //envía el byte bajo de la dirección de datos SPI //negar selección de chip

        SPI_CS = 1;
    }

    void array_read (caracter sin firmar addhigh, char sin firmar
                     addlow, char sin firmar *rdptr,
                     conteo de caracteres sin firmar)

    {
        SPI_CS = 0; var           //afirmar selección de chip
        = putcSPI(SPI_READ); //enviar comando de lectura var =
        putcSPI(addhigh); //enviar byte alto de dirección var = putcSPI(addlow); //envía el
        byte bajo de la dirección getsSPI(rdptr, count); //leer varios bytes

        SPI_CS = 1;
    }

    void byte_write (caracter sin firmar addhigh, char sin firmar
                     addlow, datos de char sin firmar)

    {
        SPI_CS = 0; var           //afirmar selección de chip
        = putcSPI(SPI_WRITE); //enviar comando de escritura var =
        putcSPI(addhigh); //envía el byte alto de la dirección //envía el byte bajo de la dirección
        datos var = putcSPI(datos); //negar selección de chip
        var = putcSPI(addlow); //enviar byte de

        SPI_CS = 1;
    }

    byte_read de carácter sin firmar (añadir alto de carácter sin firmar, añadir
                                     bajo de carácter sin firmar)
    {
        SPI_CS = 0; var           //afirmar selección de chip
        = putcSPI(SPI_READ); //enviar comando de lectura var =
        putcSPI(addhigh); //envía el byte alto de la dirección //envía el byte bajo de la dirección
        var = getcSPI();           var = putcSPI(addlow); //leer un solo byte

        SPI_CS = 1;
        retorno (var);
    }
}

```

Bibliotecas del compilador MPLAB® C18 C

```
char sin firmar status_read (vacío) {  
  
    SPI_CS = 0; var           //afirmar selección de chip  
    = putcSPI(SPI_RDSR); //enviar comando de estado de lectura //leer byte de  
    de chip                 datos var = getcSPI(); //negar selección  
    SPI_CS = 1;  
    retorno (var);  
}  
  
void status_write (datos de caracteres sin firmar) {  
  
    SPI_CS = 0; var  
    = putcSPI(SPI_WRSR); //comando de estado de escritura var =  
    putcSPI(datos); // byte de estado a escribir  
    SPI_CS = 1; //negar selección de chip  
}  
  
vacío ocupado_sondeo (vacío) {  
  
    hacer  
    {  
        SPI_CS = 0; var = putcSPI(SPI_RDSR); //afirmar chip selección de lectura  
        de estado var = getcSPI(); //leer byte de datos //negar chip seleccionar SPI_CS  
        = 1; } mientras (var & 0x01); //permanecer en bucle hasta que !ocupado  
}  
}
```

2.9 FUNCIONES DEL TEMPORIZADOR

Los periféricos del temporizador son compatibles con las siguientes funciones:

TABLA 2-12: FUNCIONES DEL TEMPORIZADOR

Función	Descripción
CerrarTemporizadorx	Desactivar temporizador x.
OpenTimerx	Configure y habilite el temporizador x.
ReadTimerx	Lea el valor del temporizador x.
escribirtemporizadorx	Escriba un valor en el temporizador x.

2.9.1 Descripciones de funciones

CerrarTemporizador0

CerrarTemporizador1

cerrartemporizador2

CerrarTemporizador3

CerrarTemporizador4

Función: Deshabilitar el temporizador especificado.

Incluir: temporizadores.h

Prototipo: vacío CloseTimer0 (vacío); void
CloseTimer1(void); vacío CloseTimer2
(vacío); vacío CloseTimer3 (vacío); vacío
CloseTimer4 (vacío);

Observaciones: Esta función deshabilita la interrupción y el temporizador especificado.

Nombre del archivo: t0close.c
t1close.c
t2close.c
t3close.c
t4close.c

Bibliotecas del compilador MPLAB® C18 C

Temporizador abierto0

Función:	Configurar y habilitar timer0.
Incluir:	temporizadores.h
Prototipo:	void OpenTimer0(configuración de caracteres sin firmar);
Argumentos:	config
Una máscara de bits que se crea realizando una operación AND bit a bit ('&') con un valor de cada una de las categorías enumeradas a continuación. Estos valores se definen en el archivo timers.h.	
Habilitar la interrupción del temporizador 0:	
TIMER_INT_ON Interrupción habilitada	
TIMER_INT_OFF Interrupción deshabilitada	
Ancho del temporizador:	
T0_8BIT	modo de 8 bits
T0_16BIT	modo de 16 bits
Fuente del reloj:	
T0_SOURCE_EXT	Fuente de reloj externa (pin I/O)
T0_SOURCE_INT	Fuente de reloj interno (TOSC)
Disparador de reloj externo (para T0_SOURCE_EXT):	
T0_EDGE_FALL	Reloj externo en flanco descendente
T0_EDGE_RISE	Reloj externo en flanco ascendente
Valor de preescala:	
T0_PS_1_1	Preescala 1:1
T0_PS_1_2	Preescala 1:2
T0_PS_1_4	Preescala 1:4
T0_PS_1_8	Preescala 1:8
T0_PS_1_16	Preescala 1:16
T0_PS_1_32	Preescala 1:32
T0_PS_1_64	Preescala 1:64
T0_PS_1_128	Preescala 1:128
T0_PS_1_256	Preescala 1:256
Observaciones:	Esta función configura timer0 según las opciones especificadas y luego lo habilita.
Nombre del archivo:	abrir.c
Ejemplo de código:	AbrirTemporizador0(TIMER_INT_OFF & T0_8BIT & T0_FUENTE_INT & T0_PS_1_32);

Funciones de periféricos de hardware

Temporizador abierto1

Función:	Configurar y habilitar timer1.
Incluir:	temporizadores.h
Prototipo:	void OpenTimer1 (configuración de caracteres sin firmar);
Argumentos:	config Una máscara de bits que se crea realizando una operación AND bit a bit ('&') con un valor de cada una de las categorías enumeradas a continuación. Estos valores se definen en el archivo timers.h.
Habilitar la interrupción del temporizador 1:	
	TIMER_INT_ON Interrupción habilitada
	TIMER_INT_OFF Interrupción deshabilitada
Ancho del temporizador:	
T1_8BIT_RW	modo de 8 bits
T1_16BIT_RW	modo de 16 bits
Fuente del reloj:	
T1_SOURCE_EXT	Fuente de reloj externa (pin de E/S)
T1_SOURCE_INT	Fuente de reloj interno (TOSC)
Prescaler:	
T1_PS_1_1	Preescala 1:1
T1_PS_1_2	Preescala 1:2
T1_PS_1_4	Preescala 1:4
T1_PS_1_8	Preescala 1:8
Uso del oscilador:	
T1_OSC1EN_ON	Habilitar el oscilador Timer1
T1_OSC1EN_OFF	Deshabilitar el oscilador Timer1
Sincronizar entrada de reloj:	
T1_SYNC_EXT_ON	Entrada de sincronización de reloj externo
T1_SYNC_EXT_OFF	No sincronizar entrada de reloj externo
Uso con PCC:	
<u>Para dispositivos con 1 o 2 CCP</u>	
T3_SOURCE_CCP	Fuente Timer3 para ambos CCP
T1_CCP1_T3_CCP2	Fuente Timer1 para CCP1 y CCP2
T1_SOURCE_CCP	Fuente Timer1 para ambos CCP
<u>Para dispositivos con más de 2 CCP</u>	
T34_SOURCE_CCP	Timer3 y Timer4 son fuentes para todos PCCh
T12_CCP12_T34_CCP345	Timer1 y Timer2 son fuentes para CCP1 y CCP2 y Timer3 y Timer4 son fuentes para CCP3 a CCP5
T12_CCP1_T34_CCP2345	Timer1 y Timer2 son fuentes para CCP1 y Timer3 y Timer4 son fuentes para CCP2 a CCP5
T12_SOURCE_CCP	Timer1 y Timer2 son fuentes para todos PCCh
Observaciones:	Esta función configura el temporizador 1 de acuerdo con las opciones especificadas y luego lo habilita.
Nombre del archivo:	t1open.c
Ejemplo de código:	AbrirTemporizador1(TIMER_INT_ON & T1_8BIT_RW & T1_FUENTE_EXT & T1_PS_1_1 y T1_OSC1EN_OFF & T1_SYNC_EXT_OFF);

Bibliotecas del compilador MPLAB® C18 C

AbrirTemporizador2

Función:	Configurar y habilitar timer2.
Incluir:	temporizadores.h
Prototipo:	void OpenTimer2 (configuración de caracteres sin firmar);
Argumentos:	config
	Una máscara de bits que se crea realizando una operación AND bit a bit ('&') con un valor de cada una de las categorías enumeradas a continuación. Estos valores se definen en el archivo timers.h.
Habilitar interrupción de Timer2:	
	TIMER_INT_ON Interrupción habilitada
	TIMER_INT_OFF Interrupción deshabilitada
Valor de preescala:	
T2_PS_1_1	Preescala 1:1
T2_PS_1_4	Preescala 1:4
T2_PS_1_16	Preescala 1:16
Valor de posescala:	
T2_POST_1_1	1:1 posescala
T2_POST_1_2	1:2 posescala
:	:
T2_POST_1_15	1:15 posescala
T2_POST_1_16	1:16 posescala
Uso con PCC:	
<u>Para dispositivos con 1 o 2 CCP</u>	
T3_SOURCE_CCP	Fuente Timer3 para ambos CCP
T1_CCP1_T3_CCP2	Fuente Timer1 para CCP1 y
	Fuente Timer3 para CCP2
T1_SOURCE_CCP	Fuente Timer1 para ambos CCP
<u>Para dispositivos con más de 2 CCP</u>	
T34_SOURCE_CCP	Timer3 y Timer4 son fuentes para todos PCCh
T12_CCP12_T34_CCP345	Timer1 y Timer2 son fuentes para CCP1 y CCP2 y Timer3 y Timer4 son fuentes para CCP3 a CCP5
T12_CCP1_T34_CCP2345	Timer1 y Timer2 son fuentes para CCP1 y Timer3 y Timer4 son fuentes para CCP2 a CCP5
T12_SOURCE_CCP	Timer1 y Timer2 son fuentes para todos PCCh
Observaciones:	
	Esta función configura el temporizador 2 de acuerdo con las opciones especificadas y luego lo habilita.
Nombre del archivo:	t2open.c
Ejemplo de código:	AbrirTemporizador2(TIMER_INT_OFF & T2_PS_1_1 & T2_POST_1_8);

Funciones de periféricos de hardware

AbrirTemporizador3

Función:	Configurar y habilitar timer3.
Incluir:	temporizadores.h
Prototipo:	void OpenTimer3 (configuración de caracteres sin firmar);
Argumentos:	config
Una máscara de bits que se crea realizando una operación AND bit a bit ('&') con un valor de cada una de las categorías enumeradas a continuación. Estos valores se definen en el archivo timers.h.	
Habilitar la interrupción de Timer3:	
	TIMER_INT_ON Interrupción habilitada
	TIMER_INT_OFF Interrupción deshabilitada
Ancho del temporizador:	
	T3_8BIT_RW modo de 8 bits
	T3_16BIT_RW modo de 16 bits
Fuente del reloj:	
	T3_SOURCE_EXT Fuente de reloj externa (pin de E/S)
	T3_SOURCE_INT Fuente de reloj interno (TOSC)
Valor de preescala:	
	T3_PS_1_1 preescala 1:1
	T3_PS_1_2 Preescala 1:2
	T3_PS_1_4 Preescala 1:4
	T3_PS_1_8 Preescala 1:8
Sincronizar entrada de reloj:	
	T3_SYNC_EXT_ON Entrada de sincronización de reloj externo
	T3_SYNC_EXT_OFF No sincronizar entrada de reloj externo
Uso con PCC:	
<u>Para dispositivos con 1 o 2 CCP</u>	
	T3_SOURCE_CCP Fuente Timer3 para ambos CCP
	T1_CCPI_T3_CCPI2 Fuente Timer1 para CCP1 y
	Fuente Timer3 para CCP2
	T1_SOURCE_CCP Fuente Timer1 para ambos CCP
<u>Para dispositivos con más de 2 CCP</u>	
	T34_SOURCE_CCP Timer3 y Timer4 son fuentes para todos PCCh
	T12_CCPI_T34_CCPI2 Timer1 y Timer2 son fuentes para CCP1 y CCP2 y Timer3 y
	Timer4 son fuentes para CCP3 a CCP5
	T12_CCPI_T34_CCPI3 Timer1 y Timer2 son fuentes para CCP1 y Timer3 y Timer4 son fuentes para CCP2 a CCP5
	T12_SOURCE_CCP Timer1 y Timer2 son fuentes para todos PCCh
Observaciones:	Esta función configura timer3 de acuerdo con las opciones especificadas y luego lo habilita.
Nombre del archivo:	t3open.c
Ejemplo de código:	AbrirTemporizador3(TIMER_INT_ON & T3_8BIT_RW & T3_FUENTE_EXT & T3_PS_1_1 y T3_OSC1EN_OFF & T3_SYNC_EXT_OFF);

Bibliotecas del compilador MPLAB® C18 C

AbrirTemporizador4

Función:	Configurar y habilitar timer4.
Incluir:	temporizadores.h
Prototipo:	void OpenTimer4(configuración de caracteres sin firmar);
Argumentos:	config Una máscara de bits que se crea realizando una operación AND bit a bit ('&') con un valor de cada una de las categorías enumeradas a continuación. Estos valores se definen en el archivo timers.h.
Habilitar la interrupción de Timer4:	
	TIMER_INT_ON Interrupción habilitada
	TIMER_INT_OFF Interrupción deshabilitada
Valor de preescala:	
	T4_PS_1_1 Preescala 1:1
	T4_PS_1_4 Preescala 1:4
	T4_PS_1_16 Preescala 1:16
Valor de posescala:	
	T4_POST_1_1 1:1 posescala
	T4_POST_1_2 1:2 posescala
	: :
	T4_POST_1_15 1:15 posescala
	T4_POST_1_16 1:16 posescala
Observaciones:	Esta función configura timer4 de acuerdo con las opciones especificadas y luego lo habilita.
Nombre del archivo:	t4open.c
Ejemplo de código:	AbrirTemporizador4(TIMER_INT_OFF & T4_PS_1_1 T4_POST_1_8);

ReadTimer0

ReadTimer1

ReadTimer2

ReadTimer3

ReadTimer4

Función: Lee el valor del temporizador especificado.

Incluir: temporizadores.h

Prototipo: sin firmar int ReadTimer0 (vacío); int sin firmar ReadTimer1
 (vacío); carácter sin firmar ReadTimer2(void); int sin
 firmar ReadTimer3 (vacío); carácter sin firmar
 ReadTimer4(void);

Observaciones: Estas funciones leen el valor de los respectivos registros del temporizador.

Temporizador0:	TMR0L,TMR0H
Temporizador1:	TMR1L,TMR1H
Temporizador2:	TMR2
Temporizador3:	TMR3L,TMR3H
Temporizador4:	TMR4

Nota: Cuando se utiliza un temporizador en modo de 8 bits que puede configurarse en modo de 16 bits (por ejemplo, timer0), no se garantiza que el byte superior sea cero. El usuario puede desear convertir el resultado en un char para obtener resultados correctos. Por ejemplo:

```
// Ejemplo de lectura de un resultado de 16 bits // de un
temporizador de 16 bits que opera en // modo de 8 bits: resultado
int sin signo; resultado = (caracter sin firmar) ReadTimer0();
```

Valor de retorno: El valor actual del temporizador.

Nombre del archivo: t0leer.c
 t1leer.c
 t2leer.c
 t3leer.c t4leer.c

Bibliotecas del compilador MPLAB® C18 C

WriteTimer0 WriteTimer1 WriteTimer2 WriteTimer3 WriteTimer4

Función: Escriba un valor en el temporizador especificado.

Incluir: temporizadores.h

Prototipo: void WriteTimer0 (temporizador int sin firmar); void WriteTimer1 (temporizador int sin firmar); void WriteTimer2 (temporizador de caracteres sin firmar); void WriteTimer3 (temporizador int sin firmar); void WriteTimer4 (temporizador de caracteres sin firmar);

Argumentos: Temporizador

El valor que se cargará en el temporizador especificado.

Observaciones: Estas funciones escriben un valor en los respectivos registros del temporizador:

Temporizador0: TMR0L, TMR0H TMR1L, TMR1H

Temporizador1:

Temporizador2: TMR2

Temporizador3: TMR3L,TMR3H

Temporizador4: TMR4

Nombre del archivo: t0escribir.c

t1escribir.c

t2escribir.c

t3escribir.c

t4escribir.c

Ejemplo de código: WriteTimer0(10000);

2.9.2 Ejemplo de uso

```
#incluir <p18C452.h> #incluir
<temporizadores.h> #incluir
<usart.h> #incluir <stdlib.h>

vacío principal (vacío) {

    resultado int; char
    cadena[7];

    // configurar temporizador0
    AbrirTemporizador0( TIMER_INT_OFF &
                        T0_FUENTE_INT &
                        T0_PS_1_32);

    // configurar USART
    AbrirUSART( USART_TX_INT_OFF &
                USART_RX_INT_OFF &
                USART_SYNCH_MODE &
                USART_EIGHT_BIT &
                USART_CONT_RX, 25
                );

    mientras ( 1 ) {

        while( ! PORTBbits.RB3 ); // esperar a que RB3 sea alto // leer el resultado del
                                temporizador = ReadTimer0();

        si (resultado > 0xc000)           // sale del bucle si el valor // está
            descanso;                  fuera de rango

        EscribirTiempo0( 0 );           // reiniciar el temporizador

        ultoa(resultado, cadena);
        poneUSART(str);              // convierte el temporizador en una
                                      // cadena // imprime una cadena
    }

    CerrarTemporizador0();          // cerrar modulos
    CerrarUSART();
}
```

Bibliotecas del compilador MPLAB® C18 C

2.10 FUNCIONES USART

Las siguientes rutinas se proporcionan para dispositivos con un solo periférico USART:

TABLA 2-13: FUNCIONES DEL PERIFÉRICO USART ÚNICO

Función	Descripción
OcupadoUSART	¿Está transmitiendo el USART?
CerrarUSART	Deshabilite el USART.
DatosRdyUSART	¿Hay datos disponibles en el búfer de lectura USART?
obtenercUSART	Lee un byte del USART.
obtieneUSART	Lea una cadena del USART.
AbrirUSART	Configure el USART.
putcUSART	Escribe un byte en el USART.
poneUSART	Escriba una cadena desde la memoria de datos al USART.
putrsUSART	Escriba una cadena desde la memoria del programa al USART.
LeerUSART	Lee un byte del USART.
EscribirUSART	Escribe un byte en el USART.
baudiosUSART	Establezca los bits de configuración de velocidad en baudios para USART mejorado.

Las siguientes rutinas se proporcionan para dispositivos con múltiples periféricos USART:

TABLA 2-14: MÚLTIPLES FUNCIONES DEL PERIFÉRICO USART

Función	Descripción
BusyxUSART	¿USART x está transmitiendo?
CerrarxUSART	Deshabilitar USART x.
DatosRdyxUSART	¿Hay datos disponibles en el búfer de lectura de USART x?
getcxUSART	Leer un byte de USART x.
consiguerxUSART	Leer una cadena de USART x.
OpenxUSART	Configurar USART x.
putcxUSART	Escribe un byte en USART x.
putsxUSART	Escriba una cadena desde la memoria de datos a USART x.
putrsxusart	Escriba una cadena desde la memoria del programa a USART x.
LeerxUSART	Leer un byte de USART x.
EscribirxUSART	Escribe un byte en USART x.
baudiosUSART	Establezca los bits de configuración de velocidad en baudios para USART x mejorado.

2.10.1 Descripciones de funciones**OcupadoUSART****Busy1USART****Busy2USART**

Función:	¿Está transmitiendo el USART?
Incluir:	usart.h
Prototipo:	char BusyUSART(vacío); char Busy1USART(vacío); char Busy2USART(vacío);
Observaciones:	Devuelve un valor que indica si el transmisor USART está actualmente ocupado. Esta función debe usarse antes de comenzar una nueva transmisión. BusyUSART debe usarse en piezas con un solo periférico USART. Busy1USART y Busy2USART deben usarse en piezas con múltiples periféricos USART. 0 si el transmisor USART está inactivo
Valor de retorno:	1 si el transmisor USART está en uso
Nombre del archivo:	ubusy.c u1busy.c u2busy.c
Ejemplo de código:	mientras (OcupadoUSART());

CerrarUSART**Cerrar1USART****Cerrar2USART**

Función:	Deshabilite el USART especificado.
Incluir:	usart.h
Prototipo:	vacio Cerrar USART (vacío); vacío Close1USART (vacío); vacío Close2USART (vacío);
Observaciones:	Esta función desactiva las interrupciones, el transmisor y el receptor para el USART especificado. CloseUSART debe usarse en piezas con un solo periférico USART. Close1USART y Close2USART deben usarse en piezas con varios periféricos USART.
Nombre del archivo:	ucerrar.c u1cerrar.c u2cerrar.c

Bibliotecas del compilador MPLAB® C18 C

DatosRdyUSART

DatosRdy1USART

DatosRdy2USART

Función:	¿Hay datos disponibles en el búfer de lectura?
Incluir:	uart.h
Prototipo:	char DataRdyUSART(vacío); char DataRdy1USART(vacio); char DataRdy2USART(vacio);
Observaciones:	Esta función devuelve el estado del bit de bandera RCIF en el registro PIR. DataRdyUSART debe usarse en piezas con un solo periférico USART. DataRdy1USART y DataRdy2USART deben usarse en piezas con múltiples periféricos USART.
Valor de retorno:	1 si hay datos disponibles 0 si no hay datos disponibles
Nombre del archivo:	udrdy.c u1drdy.c u2drdy.c
Ejemplo de código:	while (!DataRdyUSART());

getcUSART

getc1USART

getc2USART

getcxUSART se define como ReadxUSART. Ver **ReadUSART**

obtieneUSART

obtiene1USART

obtiene2USART

Función:	Ler una cadena de caracteres de longitud fija del USART especificado.
Incluir:	uart.h
Prototipo:	void getsUSART (char * buffer, charlen sin firmar); void gets1USART (char * buffer, charlen sin firmar); void gets2USART (char * buffer, charlen sin firmar);
Argumentos:	búfer Un puntero a la ubicación donde se almacenarán los caracteres entrantes. Len El número de caracteres para leer del USART.
Observaciones:	Esta función solo funciona en el modo de transmisión/recepción de 8 bits. Esta función espera y lee una gran cantidad de caracteres del USART especificado. No hay tiempo de espera cuando se espera que lleguen los personajes. getsUSART debe usarse en piezas con un solo periférico USART. gets1USART y gets2USART deben usarse en piezas con varios periféricos USART.
Nombre del archivo:	ugets.c u1gets.c u2gets.c
Ejemplo de código:	char inputstr[10]; obtieneUSART(cadenaentrada, 5);

AbrirUSART

Abierto1USART

Open2USART

Función:	Configure el módulo USART especificado.																												
Incluir:	uart.h																												
Prototipo:	<pre>void OpenUSART (configuración de caracteres sin firmar, spbrg int sin firmar); void Open1USART (configuración de caracteres sin firmar, spbrg int sin firmar); void Open2USART (configuración de caracteres sin firmar, spbrg int sin firmar);</pre>																												
Argumentos:	config Una máscara de bits que se crea realizando una operación AND bit a bit ('&') con un valor de cada una de las categorías enumeradas a continuación. Estos valores se definen en el archivo usart.h.																												
	Interrupción en la transmisión: <table border="0"> <tr> <td>USART_TX_INT_ON</td> <td>Interrupción de transmisión ON</td> </tr> <tr> <td>USART_TX_INT_OFF</td> <td>Interrupción de transmisión APAGADO</td> </tr> </table> Interrupción al recibir: <table border="0"> <tr> <td>USART_RX_INT_ON</td> <td>Interrupción de recepción ON</td> </tr> <tr> <td>USART_RX_INT_OFF</td> <td>Interrupción de recepción APAGADO</td> </tr> </table> Modo USART: <table border="0"> <tr> <td>USART_ASYNCH_MODE</td> <td>Modo asíncrono</td> </tr> <tr> <td>USART_SYNCH_MODE</td> <td>Modo síncrono</td> </tr> </table> Ancho de transmisión: <table border="0"> <tr> <td>USART_EIGHT_BIT</td> <td>transmisión/recepción de 8 bits</td> </tr> <tr> <td>USART_NINE_BIT</td> <td>transmisión/recepción de 9 bits</td> </tr> </table> Selección de esclavo/maestro*: <table border="0"> <tr> <td>USART_SYNC_SLAVE</td> <td>Modo esclavo síncrono</td> </tr> <tr> <td>USART_SYNC_MASTER</td> <td>Modo maestro síncrono</td> </tr> </table> Modo de recepción: <table border="0"> <tr> <td>USART_SINGLE_RX</td> <td>Recepción única</td> </tr> <tr> <td>USART_CONT_RX</td> <td>Recepción continua</td> </tr> </table> Tasa de baudios: <table border="0"> <tr> <td>USART_BRGH_ALTO</td> <td>Alta tasa de baudios</td> </tr> <tr> <td>USART_BRGH_LOW</td> <td>Tasa de baudios baja</td> </tr> </table>	USART_TX_INT_ON	Interrupción de transmisión ON	USART_TX_INT_OFF	Interrupción de transmisión APAGADO	USART_RX_INT_ON	Interrupción de recepción ON	USART_RX_INT_OFF	Interrupción de recepción APAGADO	USART_ASYNCH_MODE	Modo asíncrono	USART_SYNCH_MODE	Modo síncrono	USART_EIGHT_BIT	transmisión/recepción de 8 bits	USART_NINE_BIT	transmisión/recepción de 9 bits	USART_SYNC_SLAVE	Modo esclavo síncrono	USART_SYNC_MASTER	Modo maestro síncrono	USART_SINGLE_RX	Recepción única	USART_CONT_RX	Recepción continua	USART_BRGH_ALTO	Alta tasa de baudios	USART_BRGH_LOW	Tasa de baudios baja
USART_TX_INT_ON	Interrupción de transmisión ON																												
USART_TX_INT_OFF	Interrupción de transmisión APAGADO																												
USART_RX_INT_ON	Interrupción de recepción ON																												
USART_RX_INT_OFF	Interrupción de recepción APAGADO																												
USART_ASYNCH_MODE	Modo asíncrono																												
USART_SYNCH_MODE	Modo síncrono																												
USART_EIGHT_BIT	transmisión/recepción de 8 bits																												
USART_NINE_BIT	transmisión/recepción de 9 bits																												
USART_SYNC_SLAVE	Modo esclavo síncrono																												
USART_SYNC_MASTER	Modo maestro síncrono																												
USART_SINGLE_RX	Recepción única																												
USART_CONT_RX	Recepción continua																												
USART_BRGH_ALTO	Alta tasa de baudios																												
USART_BRGH_LOW	Tasa de baudios baja																												
	* Se aplica solo al modo síncrono																												
	spbrg Este es el valor que se escribe en el registro del generador de velocidad en baudios que determina la velocidad en baudios a la que opera el USART. Las fórmulas para la tasa de baudios son: Modo asíncrono, alta velocidad: FOSC / $(16 * (spbrg + 1))$ Modo asíncrono, baja velocidad: $FOSC / (64 * (spbrg + 1))$ Modo síncrono: FOSC / $(4 * (spbrg + 1))$ Donde FOSC es la frecuencia del oscilador.																												
Observaciones:	Esta función configura el módulo USART según las opciones de configuración especificadas. OpenUSART debe usarse en piezas con un solo periférico USART. Open1USART y Open2USART deben usarse en piezas con múltiples periféricos USART.																												
Nombre del archivo:	uopen.c u1open.c u2open.c																												

Bibliotecas del compilador MPLAB® C18 C

AbrirUSART

Abierto1USART

Open2USART (Continuación)

Ejemplo de código: AbrirUSART1(USART_TX_INT_OFF &
 USART_RX_INT_OFF &
 USART_SYNCH_MODE &
 USART_EIGHT_BIT &
 USART_CONT_RX &
 USART_BRGH_HIGH, 25
);

putcUSART

putc1USART

putc2USART

putcxUSART se define como WritexUSART. Ver **WriteUSART**

putsUSART

puts1USART

puts2USART

putrsUSART

putrs1USART

putrs2USART

Función: Escribe una cadena de caracteres en el USART, incluido el carácter nulo.

Incluir: usart.h

Prototipo: void putsUSART(char *datos); void

puts1USART(char *datos); void

puts2USART(char *datos); void

putrsUSART(const rom char *data); void putrs1USART(const

rom char *data); void putrs2USART(const rom char *data);

Argumentos: **data**

Puntero a una cadena de datos terminada en nulo.

Observaciones: Esta función solo funciona en el modo de transmisión/recepción de 8 bits. Esta función escribe una cadena de datos en el USART, incluido el carácter nulo.

Las cadenas ubicadas en la memoria de datos deben usarse con las versiones "puts" de estas funciones.

Las cadenas ubicadas en la memoria del programa, incluidos los literales de cadena, deben usarse con las versiones "putrs" de estas funciones. putsUSART y putrsUSART deben usarse en piezas con un único periférico USART. Las otras funciones deben usarse en partes con múltiples periféricos USART.

Nombre del archivo:

uputs.c

u1puts.c

u2puts.c

uputrs.c

u1putrs.c

u2putrs.c

Ejemplo de código:

putrsUSART("¡Hola mundo!");

LeerUSART

Leer1USART

Leer2USART

getcUSART

getc1USART

getc2USART

Función: Lea un byte (un carácter) del búfer de recepción de USART, incluido el noveno bit si está habilitado.

Incluir: usart.h

Prototipo:

```
char ReadUSART( vacío ); char
Read1USART(vacio); char
Read2USART(vacio); char
getcUSART(vacio); char
getc1USART(vacio); char
getc2USART(vacio);
```

Observaciones: Esta función lee un byte del búfer de recepción de USART. Los bits de estado y los bits de datos noveno se guardan en una unión con la siguiente declaración:

```
sindicato USART
{
    valor de carácter sin firmar;
    estructura
{
    RX_NINE sin firmar: 1; sin firmar
    TX_NINE: 1; FRAME_ERROR
    sin firmar: 1; OVERRUN_ERROR sin
    firmar: 1; relleno sin firmar: 4; }; };
```

El noveno bit es de solo lectura si el modo de 9 bits está habilitado. Los bits de estado siempre se leen.

En una pieza con un solo periférico USART, se deben usar las funciones getcUSART y ReadUSART y la información de estado es leer en una variable llamada USART_Status que es del tipo USART descrito anteriormente.

En una pieza con varios periféricos USART, se deben usar las funciones getcxUSART y ReadxUSART y la información de estado es

leer en una variable llamada USARTx_Status que es del tipo USART descrito anteriormente.

Valor de retorno: Esta función devuelve el siguiente carácter en el búfer de recepción de USART.

Nombre del archivo:

```
uread.c
u1leer.c
u2read.c
#define en usart.h #define en
usart.h #define en usart.h
```

Ejemplo de código:

```
resultado int;
resultado = ReadUSART(); resultado
|= (int sin signo)
    USART_Estado.RX_NUEVE << 8;
```

Bibliotecas del compilador MPLAB® C18 C

EscribirUSART

Escribir1USART

Escribir2USART

putcUSART

putc1USART

putc2USART

Función: Escriba un byte (un carácter) en el búfer de transmisión de USART, incluido el noveno bit si está habilitado.

Incluir: usart.h

Prototipo: void WriteUSART (datos **de** caracteres); void
Write1USART (datos **de** caracteres); void
Write2USART (datos **de** caracteres); void
putcUSART (datos **de** caracteres); void
putc1USART (datos **de** caracteres); void
putc2USART (datos **de** caracteres);

Argumentos: **datos**

El valor que se va a escribir en el USART.

Observaciones: Esta función escribe un byte en el búfer de transmisión de USART. Si el modo de 9 bits está habilitado, el bit 9 se escribe desde el campo TX_NINE, que se encuentra en una variable de tipo USART:

```
unión USART {  
  
    valor de carácter sin firmar;  
    estructura {  
  
        RX_NINE sin firmar: 1; sin firmar  
        TX_NINE: 1; FRAME_ERROR  
        sin firmar: 1; OVERRUN_ERROR sin  
        firmar: 1; relleno sin firmar: 4; }; };
```

En una pieza con un solo periférico USART, se deben usar las funciones putcUSART y WriteUSART y el registro de estado se denomina USART_Status, que es del tipo USART descrito anteriormente.

En una parte con múltiples periféricos USART, se deben usar las funciones putcxUSART y WritexUSART y el registro de estado se llama USARTx_Status, que es del tipo USART descrito encima.

Nombre del archivo: uwrtie.c
u1write.c
u2write.c
#define en usart.h #define en
usart.h #define en usart.h

Ejemplo de código: int outval sin firmar;
USART1_Status.TX_NINE = (valor superior y 0x0100)
 >> 8;
Write1USART((char) outval);

baudioUSART baudio1USART baudio2USART

Función:	Establezca los bits de configuración de velocidad en baudios para una operación USART mejorada.
Incluir:	uart.h
Prototipo:	void baudUSART (baudconfig char sin firmar); void baud1USART (baudconfig char sin firmar); void baud2USART (baudconfig char sin firmar);
Argumentos:	baudconfig Una máscara de bits que se crea realizando una operación AND bit a bit ('&') con un valor de cada una de las categorías enumeradas a continuación. Estos valores se definen en el archivo uart.h: Estado de inactividad del reloj: BAUD_IDLE_CLK_HIGH El estado de inactividad del reloj es un nivel alto El estado de inactividad del reloj es un nivel bajo
	BAUD_IDLE_CLK_LOW Generación de velocidad de transmisión: BAUD_16_BIT_RATE BAUD_8_BIT_RATE
	Tasa de generación de baudios de 16 bits Tasa de generación de baudios de 8 bits
Monitoreo de pines RX:	
BAUD_WAKEUP_ON	pin RX monitoreado
BAUD_WAKEUP_OFF	Pin RX no monitoreado
Medición de la tasa de baudios:	
BAUD_AUTO_ON	Medición automática de velocidad en baudios habilitada
BAUD_AUTO_OFF	Medición automática de velocidad en baudios deshabilitada
Observaciones:	Estas funciones solo están disponibles para procesadores con capacidad USART mejorada.
Nombre del archivo:	ubaud.c u1baud.c u2baud.c
Ejemplo de código:	baudiosUSART (BAUD_IDLE_CLK_HIGH & BAUD_16_BIT_RATE & BAUD_WAKEUP_ON & BAUD_AUTO_ON);

2.10.2 Ejemplo de uso

```
#incluye <p18C452.h> #incluye  
<uart.h>  
  
vacío principal (vacío) {  
  
    // configurar USART  
    AbrirUSART( USART_TX_INT_OFF &  
                USART_RX_INT_OFF &  
                USART_SYNCH_MODE &  
                USART_EIGHT_BIT &  
                USART_CONT_RX &  
                USART_BRGH_HIGH, 25);  
  
    mientras(1) {  
  
        while( ! PORTAbits.RA0 ); //esperar a RA0 alto  
  
        WriteUSART(PORTD);           //escribe el valor de PORTD  
  
        si (PORTD == 0x80)           // comprobar si hay terminación // valor  
            descanso;  
    }  
  
    CerrarUSART();  
}
```



COMPILEADOR MPLAB® C18 C BIBLIOTECAS

Capítulo 3. Biblioteca de periféricos de software

3.1 INTRODUCCIÓN

Este capítulo documenta las funciones de la biblioteca de periféricos de software. El código fuente de todas estas funciones se incluye con MPLAB C18 en los subdirectorios `src\traditional\pmc` y `src\extended\pmc` de la instalación del compilador.

Consulte MPASM™ Assembler, MPLINK™ Object Linker, MPLIB™ Object Librarian User's Guide (DS33014) para obtener más información sobre la creación de bibliotecas.

Los siguientes periféricos son compatibles con las rutinas de la biblioteca MPLAB C18 •

Funciones de LCD externas (**Sección 3.2 “Funciones de LCD externas”**) • Funciones de CAN2510 externas (**Sección 3.3 “Funciones de CAN2510 externas”**) • Funciones de software I2C™ (**Sección 3.4 “Funciones de software I2C”**) • Funciones de Software SPI (**Sección 3.5 “Funciones de Software SPI®”**) • Funciones de Software UART (**Sección 3.6 “Funciones de Software UART”**)

3.2 FUNCIONES LCD EXTERNAS

Estas funciones están diseñadas para permitir el control de un controlador LCD Hitachi HD44780 utilizando pines de E/S de un microcontrolador PIC18. Se proporcionan las siguientes funciones:

TABLA 3-1: FUNCIONES LCD EXTERNAS

Función	Descripción
BusyXLCD	¿Está ocupado el controlador LCD?
AbrirXLCD	Configure las líneas de E/S utilizadas para controlar la pantalla LCD e inicialice la pantalla LCD.
putcXLCD	Escriba un byte en el controlador LCD.
poneXLCD	Escriba una cadena desde la memoria de datos a la pantalla LCD.
putrsXLCD	Escriba una cadena desde la memoria del programa a la pantalla LCD.
ReadAddrXLCD	Lea el byte de dirección del controlador LCD.
ReadDataXLCD	Lea un byte del controlador LCD.
EstablecerCGRamAddr	Configure la dirección del generador de caracteres.
EstablecerDDRamAddr	Configure la dirección de datos de visualización.
EscribirCmdXLCD	Escriba un comando en el controlador LCD.
WriteDataXLCD	Escriba un byte en el controlador LCD.

Las versiones precompiladas de estas funciones usan asignaciones de pines predeterminadas que se pueden cambiar redefiniendo las siguientes asignaciones de macros en el archivo `xlcd.h`, que se encuentra en el subdirectorio `h` de la instalación del compilador:

Bibliotecas del compilador MPLAB® C18 C

TABLA 3-2: MACROS PARA SELECCIONAR ASIGNACIONES DE PIN LCD

LCD Controlador Línea	macros	Valor por defecto	Utilizar
Alfiler E	E_PIN	PUERTOBbits.RB4	Pin utilizado para la línea E.
	TRIS_E	DDRBbits.RB4	Bit que controla la dirección del pin asociado a la línea E.
Pasador RS	RS_PIN	PUERTOBbits.RB5	Pin utilizado para la línea RS.
	TRIS_RS	DDRBbits.RB5	Bit que controla la dirección del pin asociado a la línea RS.
Pasador RW	RW_PIN	PUERTOBbits.RB6	Pin utilizado para la línea RW.
	TRIS_RW	DDRBbits.RB6	Bit que controla la dirección del pin asociado a la línea RW.
Líneas de datos DATA_PORT		PUERTOB	Pines utilizados para líneas de DATOS. Estas rutinas asumen que todos los pines están en un solo puerto.
	TRIS_DATA_PORT	DDRB	Registro de dirección de datos asociado con las líneas de DATOS.

Las bibliotecas que se proporcionan pueden funcionar en modo de 4 bits o de 8 bits. Cuando se opera en modo de 8 bits, se utilizan todas las líneas de un solo puerto. Cuando se opera en modo de 4 bits, se utilizan los 4 bits superiores o los 4 bits inferiores de un solo puerto. La siguiente tabla enumera las macros utilizadas para seleccionar entre el modo de 4 u 8 bits y para seleccionar qué bits de un puerto se utilizan cuando se opera en el modo de 4 bits.

TABLA 3-3: MACROS PARA SELECCIONAR EL MODO DE 4 U 8 BITS

Macro	Valor por defecto	Utilizar
BIT8	no definida	Si este valor se define cuando se crean las funciones de la biblioteca, funcionarán en el modo de transferencia de 8 bits. De lo contrario, funcionarán en modo de transferencia de 4 bits.
SUPERIOR	no definida	Cuando BIT8 no está definido, este valor determina qué nibble de DATA_PORT se usa para la transferencia de datos. Si se define UPPER, se utilizan los 4 bits superiores (4:7) de DATA_PORT. Si UPPER no está definido, se utilizan los 4 bits inferiores (0:3) de DATA_PORT.

Una vez realizadas estas definiciones, el usuario debe volver a compilar las rutinas XLCD y luego incluir los archivos actualizados en el proyecto. Esto se puede lograr agregando los archivos fuente XLCD al proyecto o recompilando los archivos de la biblioteca utilizando los archivos por lotes proporcionados.

Las bibliotecas XLCD también requieren que el usuario defina las siguientes funciones para proporcionar los retrasos apropiados:

TABLA 3-4: FUNCIONES DE RETARDO XLCD

Función	Conducta
Retraso por 18TCY	Retardo de 18 ciclos.
RetrasoPORXLCD	Retardo de 15 ms.
DelayXLCD	Retardo de 5 ms.

3.2.1 Descripciones de funciones**BusyXLCD**

Función:	¿Está ocupado el controlador LCD?
Incluir:	xlcd.h
Prototipo:	carácter sin firmar BusyXLCD(void);
Observaciones:	Esta función devuelve el estado del indicador de ocupado del controlador LCD Hitachi HD44780.
Valor de retorno:	1 si el controlador está ocupado 0 en caso contrario.
Nombre del archivo:	ocupadoxlc.c
Ejemplo de código:	while(BusyXLCD());

AbrirXLCD

Función:	Configure los pines de E/S de PIC® e inicialice el controlador LCD.
Incluir:	xlcd.h
Prototipo:	void OpenXLCD (tipo de lcd de caracteres sin firmar);
Argumentos:	Lcdtype Una máscara de bits que se crea realizando una operación AND bit a bit ('&') con un valor de cada una de las categorías enumeradas a continuación. Estos valores se definen en el archivo xlcd.h.
Interfaz de datos:	CUATRO_BIT Modo de interfaz de datos de 4 bits OCHO_BIT Modo de interfaz de datos de 8 bits
Configuración LCD:	LINE_5X7 5x7 caracteres, visualización de una sola LINE_5X10 Línea 5x10 caracteres, visualización de 5x7 LÍNEAS_5X7 caracteres, visualización de varias líneas
Observaciones:	Esta función configura los pines de E/S PIC18 utilizados para controlar el controlador LCD Hitachi HD44780. También inicializa este controlador.
Nombre del archivo:	openxlcd.c
Ejemplo de código:	OpenXLCD(OCHO_BITS Y LÍNEAS_5X7);

putcXLCD

Consulte [WriteDataXLCD](#).

Bibliotecas del compilador MPLAB® C18 C

putsLCD putrsLCD

Función:	Escriba una cadena en el controlador LCD Hitachi HD44780.
Incluir:	xlcd.h
Prototipo:	void putsLCD(char *buffer); void putrsLCD(const rom char *buffer);
Argumentos:	búfer
	Puntero a los caracteres que se escribirán en el controlador LCD.
Observaciones:	Esta función escribe una cadena de caracteres ubicada en el búfer en el controlador LCD Hitachi HD44780. Detiene la transmisión cuando se encuentra un carácter nulo. El carácter nulo no se transmite. Las cadenas ubicadas en la memoria de datos deben usarse con las versiones "puts" de estas funciones. Las cadenas ubicadas en la memoria del programa, incluidos los literales de cadena, deben usarse con las versiones "putrs" de estas funciones.
Nombre del archivo:	putsxlcd.c putrxlcd.c
Ejemplo de código:	char mybuff [20]; putrsLCD("Hola mundo"); putsLCD(mybuff);

ReadAddrLCD

Función:	Lea el byte de dirección del controlador LCD Hitachi HD44780.
Incluir:	xlcd.h
Prototipo:	carácter sin firmar ReadAddrLCD(void);
Observaciones:	Esta función lee el byte de dirección del controlador LCD Hitachi HD44780. El controlador LCD no debe estar ocupado cuando se realiza esta operación; esto se puede verificar usando la función BusyLCD. La dirección leída del controlador es para la RAM del generador de caracteres o la RAM de datos de visualización, dependiendo de la función Set??RamAddr anterior que se haya llamado.
Valor de retorno:	Esta función devuelve una cantidad de 8 bits. La dirección está contenida en los 7 bits de orden inferior y el indicador de estado OCUPADO en el bit más significativo.
Nombre del archivo:	readaddr.c
Ejemplo de código:	dirección de caracteres; mientras (BusyLCD()); dir = LeerAddrLCD();

ReadDataXLCD

Función:	Lea un byte de datos del controlador LCD Hitachi HD44780.
Incluir:	xlcd.h
Prototipo:	char ReadDataXLCD(vacío);
Observaciones:	Esta función lee un byte de datos del controlador LCD Hitachi HD44780. El controlador LCD no debe estar ocupado cuando se realiza esta operación; esto se puede verificar usando la función BusyXLCD. Los datos leídos del controlador son para la RAM del generador de caracteres o la RAM de datos de visualización, dependiendo de la función Set??RamAddr anterior que se haya llamado.
Valor de retorno:	Esta función devuelve el valor de datos de 8 bits.
Nombre del archivo:	leerdatos.c
Ejemplo de código:	datos de caracteres; mientras (BusyXLCD()); datos = ReadAddrXLCD();

EstablecerCGRamAddr

Función:	Configure la dirección del generador de caracteres.
Incluir:	xlcd.h
Prototipo:	void SetCGRAMAddr (dirección de caracteres sin firmar) ;
Argumentos:	dirección Dirección del generador de caracteres.
Observaciones:	Esta función establece la dirección del generador de caracteres del controlador LCD Hitachi HD44780. El controlador LCD no debe estar ocupado cuando se realiza esta operación; esto se puede verificar usando la función BusyXLCD.
Nombre del archivo:	setcgram.c
Ejemplo de código:	char cgaddr = 0x1F; while(BusyXLCD()); SetCGRamAddr(cgaddr);

EstablecerDDRamAddr

Función:	Configure la dirección de datos de visualización.
Incluir:	xlcd.h
Prototipo:	void SetDDRamAddr (dirección de caracteres sin firmar) ;
Argumentos:	addr Muestra la dirección de datos.
Observaciones:	Esta función establece la dirección de datos de visualización del controlador LCD Hitachi HD44780. El controlador LCD no debe estar ocupado cuando se realiza esta operación; esto se puede verificar usando la función BusyXLCD.
Nombre del archivo:	setddram.c
Ejemplo de código:	char ddaddr = 0x10; while(BusyXLCD()); SetDDRamAddr(ddaddr);

Bibliotecas del compilador MPLAB® C18 C

EscribirCmdXLCD

Función:	Escribe un comando en el controlador LCD Hitachi HD44780.			
Incluir:	xlcld.h			
Prototipo:	void WriteCmdXLCD (cmd de caracteres sin firmar);			
Argumentos:	cmd			
Especifica el comando a ejecutar. El comando puede ser uno de los siguientes valores definidos en xlcld.h: Apagar la pantalla Habilitar la pantalla sin cursor Habilitar la pantalla con el cursor parpadeante Habilitar la pantalla con el cursor sin parpadear				
QUITARSE CURSOR_DESACTIVADO BLINK_ON BLINK_OFF				
SHIFT_CUR_IZQUIERDA SHIFT_CUR_RIGHT SHIFT_DISP_LEFT SHIFT_DISP_RIGHT				
El cursor se desplaza a la izquierda El cursor se desplaza a la derecha La pantalla se desplaza a la izquierda La visualización se desplaza a la derecha				
Alternativamente, el comando puede ser una máscara de bits que se crea realizando una operación AND bit a bit ('&') con un valor de cada una de las categorías enumeradas a continuación. Estos valores se definen en el archivo xlcld.h.				
Modo de transferencia de datos:				
CUATRO_BIT OCHO_BIT				
Modo de interfaz de datos de 4 bits Modo de interfaz de datos de 8 bits				
Tipo de visualización:				
LINE_5X7 LINE_5X10 LÍNEAS_5X7				
5x7 caracteres, una sola línea 5x10 caracteres en pantalla 5x7 caracteres, varias líneas Esta función				
Observaciones:	escribe el byte de comando en el controlador LCD Hitachi HD44780. El controlador LCD no debe estar ocupado cuando se realiza esta operación; esto se puede verificar usando la función BusyXLCD.			
Nombre del archivo:	wcmdxlcd.c			
Ejemplo de código:	<pre>while(BusyXLCD()); WriteCmdXLCD(OCHO_BITS Y LÍNEAS_5X7); EscribirCmdXLCD(BLINK_ON); WriteCmdXLCD(SHIFT_DISP_LEFT);</pre>			

putcXLCD WriteDataXLCD

Función:	Escribe un byte en el controlador LCD Hitachi HD44780.	
Incluir:	xlcld.h	
Prototipo:	void WriteDataXLCD (datos de caracteres);	
Argumentos:	datos	
El valor de los datos puede ser cualquier valor de 8 bits, pero debe corresponder a la tabla de RAM de caracteres del controlador LCD HD44780.		
Observaciones:	Esta función escribe un byte de datos en el controlador LCD Hitachi HD44780. El controlador LCD no debe estar ocupado cuando se realiza esta operación; esto se puede verificar usando la función BusyXLCD. Los datos leídos del controlador son para la RAM del generador de caracteres o la RAM de datos de visualización, dependiendo de la función Set??RamAddr anterior que se haya llamado.	
Nombre del archivo:	escribirdatos.c	

3.2.2 Ejemplo de uso

```
#incluir <p18C452.h> #incluir
<xlcd.h> #incluir <retrasos.h>
#incluir <uart.h>

void DelayFor18TCY ( void ) {

    nop();
    }

void DelayPORXLCD (void) {

    Retardo1KTCYx(60); // Retraso de 15ms
    // Ciclos = (TimeDelay * Fosc) / 4
    // Ciclos = (15ms * 16MHz) / 4
    // Ciclos = 60.000
    retorno;
}

vacío DelayXLCD (vacío) {

    Retardo1KTCYx(20); // Retraso de 5ms
    // Ciclos = (TimeDelay * Fosc) / 4
    // Ciclos = (5ms * 16MHz) / 4
    // Ciclos = 20.000
    retorno;
}

} vacío principal (vacío) {

    datos de caracteres;

    // configurar LCD externo
    OpenXLCD( OCHO_BITS Y LÍNEAS_5X7 );

    // configurar USART
    Abrir USART( USART_TX_INT_OFF & USART_RX_INT_OFF &
                USART_ASYNCH_MODE & USART_EIGHT_BIT &
                USART_CONT_RX,
                25);

    mientras(1)
    {
        while(!DataRdyUSART()); //esperar datos data = ReadUSART(); //
        leer datos //escribir en LCD WriteDataXLCD(datos); si (datos == 'Q')

    }

    CerrarUSART();
}
```

Bibliotecas del compilador MPLAB® C18 C

3.3 FUNCIONES EXTERNAS CAN2510

Esta sección documenta las funciones de la biblioteca de periféricos externos del MCP2510.

Se proporcionan las siguientes funciones:

TABLA 3-5: FUNCIONES EXTERNAS CAN2510

Función	Descripción
CAN2510BitModificar	Modifica los bits especificados en un registro a los nuevos valores.
CAN2510ByteRead	Lee el registro MCP2510 especificado por la dirección.
CAN2510ByteWrite	Escribe un valor en el registro MCP2510 especificado por la dirección.
CAN2510Lectura de datos	Lee un mensaje del búfer de recepción especificado.
CAN2510DataReady	Determina si los datos están esperando en el búfer de recepción especificado.
CAN2510Deshabilitar	Lleva el pin de E/S PIC18CXXX seleccionado a nivel alto para desactivar la selección de chip del MCP2510.(1)
CAN2510Habilitar	Conduce el pin de E/S PIC18CXXX seleccionado a nivel bajo para seleccionar el chip MCP2510.(1)
CAN2510ErrorState	Lee el estado de error actual del bus CAN.
CAN2510Inic	Inicialice el puerto SPI PIC18CXXX para comunicaciones con el MCP2510 y luego configure los registros del MCP2510 para interactuar con el bus CAN.
CAN2510 Habilitar interrupción	Modifica los bits de habilitación de interrupción CAN2510 (registro CANINTE) a los nuevos valores.
CAN2510 Estado de interrupción	Indica la fuente de la interrupción CAN2510.
CAN2510LoadBufferStd	Carga un marco de datos estándar en el búfer de transferencia especificado.
CAN2510LoadBufferXtd	Carga un marco de datos extendidos en el búfer de transferencia especificado.
CAN2510CargarRTRStd	Carga una trama remota estándar en el búfer de transferencia especificado.
CAN2510CargarRTRXtd	Carga una trama remota extendida en el búfer de transferencia especificado.
Modo de lectura CAN2510	Lee el modo de funcionamiento actual del MCP2510.
CAN2510Estado de lectura	Lee el estado de los búferes de transmisión y recepción del MCP2510.
CAN2510Reiniciar	Restablece el MCP2510.
CAN2510Búfer de envío	Solicita la transmisión de mensajes para los búferes de transmisión especificados.
CAN2510Lectura secuencial	Lee la cantidad de bytes especificados en el MCP2510, comenzando en la dirección especificada. Estos valores se almacenarán en dataArray.
CAN2510Escritura secuencial	Escribe la cantidad de bytes especificados en el MCP2510, comenzando en la dirección especificada. Estos valores se escribirán desde dataArray.
CAN2510SetBufferPriority	Carga la prioridad especificada para la transmisión especificada buffer.
Modo de configuración CAN2510	Configura el modo de funcionamiento del MCP2510.
CAN2510Establecer filtro de mensajes estándar	Configura TODOS los valores de filtro y máscara del búfer de recepción específico para un mensaje estándar.

TABLA 3-5: FUNCIONES EXTERNAS DEL CAN2510 (CONTINUACIÓN)

Función	Descripción
CAN2510SetMsgFilterXtd	Configura TODOS los valores de filtro y máscara del búfer de recepción específico para un mensaje extendido.
CAN2510SetSingleFilterStd	Configura el filtro de recepción especificado con un valor de filtro para un mensaje estándar (Std).
CAN2510SetSingleFilterXtd	Configura el filtro de recepción especificado con un valor de filtro para un mensaje extendido (Xtd).
CAN2510SetSingleMaskStd	Configura la máscara de búfer de recepción especificada con un valor de máscara para un mensaje de formato estándar (Std).
CAN2510SetSingleMaskXtd	Configura la máscara de búfer de recepción especificada con una máscara valor para un mensaje extendido (Xtd).
CAN2510Búfer de escritura	Inicia la transmisión del mensaje CAN del búfer seleccionado.
CAN2510Escritura estándar	Escribe un mensaje de formato estándar en el bus CAN utilizando el primer búfer de transmisión disponible.
CAN2510WriteXtd	Escribe un mensaje de formato extendido en el bus CAN utilizando el primer búfer de transmisión disponible.

Nota 1: Las funciones CAN2510Enable y CAN2510Disable deberán volver a compilarse si: - la asignación de PICmicro MCU del pin CS se modifica desde RC2 - el archivo de encabezado del dispositivo debe cambiarse

3.3.1 Descripciones de funciones

CAN2510BitModificar

Función:	Modifica los bits especificados en un registro a los nuevos valores.
CAN requerido	
Modo(s):	Todas
Incluir:	can2510.h
Prototipo:	void CAN2510BitModify(dirección de caracteres sin firmar máscara de caracteres sin firmar datos de caracteres sin firmar);
Argumentos:	<p>dirección El valor de addr especifica la dirección del registro MCP2510 a modificar.</p> <p>mask El valor de mask especifica los bits que serán modificados.</p> <p>datos El valor de data especifica el nuevo estado de los bits.</p>
Observaciones:	Esta función modifica el contenido del registro especificado por dirección, la máscara especifica qué bits se modificarán y los datos especifican el nuevo valor para cargar en esos bits. Solo se pueden modificar registros específicos con el comando Bit Modify.
Nombre del archivo:	canbmod.c

Bibliotecas del compilador MPLAB® C18 C

CAN2510ByteRead

Función:	Lee el registro MCP2510 especificado por la dirección.
CAN requerido	
Modo(s):	Todas
Incluir:	can2510.h
Prototipo:	carácter sin firmar CAN2510ByteRead (dirección de carácter sin firmar);
Argumentos:	dirección La dirección del MCP2510 que se va a leer.
Observaciones:	Esta función lee un solo byte del MCP2510 en la dirección especificada.
Valor de retorno:	El contenido de la dirección especificada.
Nombre del archivo:	leerbyte.c

CAN2510ByteWrite

Función:	Escribe un valor en el registro MCP2510 especificado por la dirección.
CAN requerido	
Modo(s):	Todas
Incluir:	can2510.h
Prototipo:	vacio CAN2510ByteWrite (dirección de caracteres sin firmar , valor de caracteres sin firmar);
Argumentos:	dirección La dirección del MCP2510 que se va a escribir. valor El valor que se va a escribir.
Observaciones:	Esta función escribe un solo byte desde el MCP2510 en la dirección especificada.
Nombre del archivo:	wrtbyte.c

CAN2510Lectura de datos

Función:	Lee un mensaje del búfer de recepción especificado.
CAN requerido	
Modo(s):	Todos (excepto el modo de configuración)
Incluir:	can2510.h
Prototipo:	carácter sin firmar CAN2510DataRead(carácter sin firmar bufferNum , largo sin firmar * msgId , carácter sin firmar * numBytes , carácter sin firmar * datos);
Argumentos:	bufferNum Recibir búfer desde el que leer el mensaje. Uno de los siguientes valores: CAN2510_RXB0 Leer búfer de recepción 0 CAN2510_RXB1 Leer búfer de recepción 1
msgId	
numBytes	
datos	
Observaciones:	Apunta a una ubicación que será modificada por la función para contener el identificador de mensaje estándar CAN.

CAN2510DataRead (continuación)

numBytes

Apunta a una ubicación que será modificada por la función para contener el número de bytes en este mensaje.

datos

Apunta a una matriz que será modificada por la función para contener los datos del mensaje. Esta matriz debe tener al menos 8 bytes de largo, ya que esa es la longitud máxima de datos del mensaje.

Observaciones:

Esta función determina si el mensaje es un mensaje estándar o extendido, decodifica la ID y la longitud del mensaje y completa las ubicaciones proporcionadas por el usuario con la información adecuada. La función CAN2510DataReady debe usarse para determinar si un búfer específico tiene datos para leer.

Valor de retorno:

La función devuelve uno de los siguientes valores:

- CAN2510_XTDMMSG Mensaje de formato extendido
- CAN2510_STDMMSG Mensaje de formato estándar
- CAN2510_XTDRTR Solicitud de transmisión remota
(Mensaje XTD)
- CAN2510_STDRTR Solicitud de transmisión remota
(mensaje STD)

Nombre del archivo:

puedeleer.c

CAN2510DataReady

Función: Determina si los datos están esperando en el búfer de recepción especificado.

CAN requerido

Modo(s): Todos (excepto el modo de configuración)

Incluir:

can2510.h

Prototipo:

carácter sin firmar CAN2510DataReady(carácter sin
firmar **bufferNum**);

Argumentos:**bufferNum**

Recibir búfer para comprobar si hay un mensaje en espera. Uno de los siguientes valores: CAN2510_RXB0 CAN2510_RXB1 CAN2510_RXBX Esta función prueba el bit RXnIF apropiado en el registro **CNTR**.

Comprobar el búfer de recepción 0

Compruebe el búfer de recepción 1

Compruebe el búfer de recepción 0 y el búfer de recepción 1

Observaciones:**Valor de retorno:**

Devuelve cero si no se detecta ningún mensaje o un valor distinto de cero si se detectó un mensaje. 1 = búfer0 2 = búfer1 3 = ambos

Nombre del archivo: canready.c

Bibliotecas del compilador MPLAB® C18 C

CAN2510Deshabilitar

Función: Lleva el pin de E/S PIC18CXXX seleccionado a nivel alto para desactivar la selección de chip del MCP2510.

CAN requerido**Modo(s):**

Todas

Incluir:

canenabl.h

Nota: Este archivo de inclusión deberá modificarse si la señal Chip Select no está asociada con el pin RC2 de la MCU PICmicro.

Prototipo:

vacío CAN2510Desactivar (vacío);

Argumentos:

Ninguno

Observaciones:

Esta función requiere que el usuario modifique el archivo para especificar el pin de E/S PIC18CXXX (y el puerto) que se usará para conectarse al pin MCP2510 CS. El pin predeterminado es RC2.

Nota: El archivo fuente que contiene esta función (y el

CAN2510Enable function) debe tener las definiciones modificadas para especificar correctamente el puerto (A, B, C, ...) y el número de pin (1, 2, 3, ...) que se utiliza para controlar el pin MCP2510 CS. Después de la modificación, se debe reconstruir la biblioteca específica del procesador. Consulte la Sección 1.5.3 "Reconstrucción" para obtener información sobre la reconstrucción.

Nombre del archivo:

canenabl.c

CAN2510Habilitar

Función: Conduce el pin de E/S PIC18CXXX seleccionado a nivel bajo para Chip Seleccione el MCP2510.

CAN requerido**Modo(s):**

Todas

Incluir:

canenabl.h

Nota: Este archivo de inclusión deberá modificarse si la señal Chip Select no está asociada con el pin RC2 de la MCU PICmicro.

Prototipo:

vacío CAN2510Enable (vacío);

Observaciones:

Esta función requiere que el usuario modifique el archivo para especificar el pin de E/S PIC18CXXX (y el puerto) que se usará para conectarse al pin MCP2510 CS. El pin predeterminado es RC2.

Nota: El archivo fuente que contiene esta función (y el

CAN2510Disable function) debe tener las definiciones modificadas para especificar correctamente el puerto (A, B, C, ...) y el número de pin (1, 2, 3, ...) que se utiliza para controlar el pin MCP2510 CS. Después de la modificación, se debe reconstruir la biblioteca específica del procesador. Consulte la Sección 1.5.3 "Reconstrucción" para obtener información sobre la reconstrucción.

Nombre del archivo:

canenabl.c

CAN2510ErrorState

Función:	Lee el estado de error actual del bus CAN.
CAN requerido	Modo normal, modo de bucle invertido, modo de solo escucha
Modo(s):	(Los contadores de errores se restablecen en el modo de configuración)
Incluir:	can2510.h
Prototipo:	carácter sin firmar CAN2510ErrorState (vacío); Esta función devuelve
Observaciones:	el estado de error del bus CAN. El estado de error depende de los valores en los registros TEC y REC.
Valor de retorno:	La función devuelve uno de los siguientes valores: CAN2510_BUS_OFF CAN2510_ERROR_PASSIVE_TX TEC > 255 CAN2510_ERROR_PASSIVE_RX TEC > 127 CAN2510_ERROR_ACTIVE_WITH_TXWARN TEC > REC > 127 95 CAN2510_ERROR_ACTIVE_WITH_RXWARN REC > 95 CAN2510_ERROR_ACTIVE TEC & REC > 95 TEC & REC & 95
Nombre del archivo:	canerrst.c

CAN2510Inic

Función:	Inicialice el puerto SPI PIC18CXXX para comunicaciones con el MCP2510 y luego configure los registros del MCP2510 para interactuar con el bus CAN.
CAN requerido	
Modo(s):	Modo de configuración
Incluir:	can2510.h
Prototipo:	carácter sin firmar CAN2510Init (bufferconfig corto y largo sin firmar , BitTimeConfig corto y largo sin firmar , interruptEnables de carácter sin firmar , SPI_syncMode de carácter sin firmar, SPI_busMode de carácter sin firmar, SPI_smpPhase de carácter sin firmar); Los valores de los siguientes parámetros se definen en el archivo de inclusión can2510.h.

Argumentos:**BufferConfig** El

valor de BufferConfig se construye a través de la operación AND (&) bit a bit de las siguientes opciones. Solo se puede seleccionar una opción por función de grupo. La opción en **negrita** es el valor predeterminado.

Restablecer dispositivo

MCP2510 Especifica si se debe enviar el comando Restablecer MCP2510. Esto no corresponde a un bit en los registros MCP2510.

CAN2510_NORESET Resetear el MCP2510. **No resetear el MCP2510**
RXB0M1:RXB0M0 (registro RXB0CTRL) Filtrado Controlado por los bits

CAN2510_RXB0_USEFILT Recibir solo mensajes filtrados por RXB0CTRL.
CAN2510_RXB0_XTDMMSG Recibir todos los mensajes, independientemente de RXB0_NOFILT
RXB1M1:RXB1M0 (registro RXB1CTRL) Buffer 1 Filtrado controlado por los bits

CAN2510_RXB1_USEFILT Recibir solo mensajes filtrados por RXB1CTRL.
CAN2510_RXB1_XTDMMSG Recibir todos los mensajes, independientemente de RXB1_NOFILT

Bibliotecas del compilador MPLAB® C18 C

CAN2510Init (continuación)

Transferencia de recepción de búfer 0 a recepción de búfer 1

Controlado por el bit BUKT (registro RXB0CTRL)

CAN2510_RXB0_ROLLO

Si el búfer de recepción 0 está lleno, el

mensaje va al búfer de recepción 1

CAN2510_RXB0_NOROLL

Rollover deshabilitado

Configuración de pines RX1BF

Controlada por los bits B1BFS:B1BFE:B1BFM (registro BFPCTRL)

CAN2510_RX1BF_OFF El pin RX1BF es una ~~salida RX1BF no se aplica impedancia~~ que se cargó el búfer de recepción 1. Se

CAN2510_RX1BF_GPOUTH

El pin RX1BF es una salida digital de uso general, salida alta El pin RX1BF es una salida digital de uso general, salida baja

CAN2510_RX1BF_GPOUTL

Configuración del pin

RX0BF controlado por los bits B0BFS:B0BFE:B0BFM (registro BFPCTRL)

CAN2510_RX0BF_OFF El pin RX0BF es un ~~El pin RX0BF tiene de alta impedancia~~ utilizarse como una señal de interrupción. de recepción 0 CAN2510_RX0BF_INT. Se puede

CAN2510_RX0BF_GPOUTH El pin RX0BF es una salida digital de uso general, salida alta

CAN2510_RX0BF_GPOUTL El pin RX0BF es una

salida digital de uso general, salida baja

Configuración de clavijas TX2

Controlado por el bit B2RTSM (registro TXRTSCTRL)

CAN2510_TX2_GPIN

El pin TX2RTS es una entrada digital

CAN2510_TX2_RTS

El pin TX2RTS es una entrada utilizada para iniciar un Solicitud de envío de trama desde TXBUF2

Configuración de clavijas TX1

Controlado por el bit B1RTSM (registro TXRTSCTRL)

CAN2510_TX1_GPIN

El pin TX1RTS es una entrada digital

CAN2510_TX1_RTS

El pin TX1RTS es una entrada utilizada para iniciar un Solicitud de envío de trama desde TXBUF1

Configuración de clavijas TX0

Controlado por el bit B0RTSM (registro TXRTSCTRL)

CAN2510_TX0_GPIN

El pin TX0RTS es una entrada digital

CAN2510_TX0_RTS

El pin TX0RTS es una entrada utilizada para iniciar un Solicitud de envío de trama desde TXBUF0

Solicitar modo de operación

Controlado por los bits REQOP2:REQOP0 (registro CANCTRL)

CAN2510_REQ_CONFIG

Modo de configuración

CAN2510_REQ_NORMAL

Modo de funcionamiento normal

CAN2510_REQ_SLEEP

Modo de sueño

CAN2510_REQ_LOOPBACK Modo de bucle inverso

CAN2510_REQ_LISTEN

Modo de solo escuchar

Configuración de clavijas CLKOUT

Controlado por los bits CLKEN:CLKPRE1:CLKPRE0 (registro CANCTRL)

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 está deshabilitado

CAN2510Init (continuación)

BitTimeConfig El valor de BitTimeConfig se construye a través de la operación AND (&) bit a bit de las siguientes opciones. Solo se puede seleccionar una opción por función de grupo. La opción en **negrita** es el valor predeterminado.

Prescaler de tasa de baudios (BRP)

Controlado por los bits BRP5:BRP0 (registro CNF1)

CAN2510_BRG_1X **TQ = 1 x (2TOSC)**

:

CAN2510_BRG_64X TQ = 64 x (2TOSC)

Ancho de salto de sincronización

Controlado por los bits SJW1:SJW0 (registro CNF1)

CAN2510_SJW_1TQ **SJW longitud = 1 TQ**

CAN2510_SJW_2TQ SJW longitud = 2 TQ

CAN2510_SJW_3TQ SJW longitud = 3 TQ

CAN2510_SJW_4TQ SJW longitud = 4 TQ

Ancho del segmento de la fase 2

Controlado por los bits PH2SEG2:PH2SEG0 (registro CNF3)

CAN2510_PH2SEG_2TQ **Longitud = 2 TQ**

CAN2510_PH2SEG_3TQ Longitud = 3 TQ

CAN2510_PH2SEG_4TQ Longitud = 4 TQ

CAN2510_PH2SEG_5TQ Longitud = 5 TQ

CAN2510_PH2SEG_6TQ Longitud = 6 TQ

CAN2510_PH2SEG_7TQ Longitud = 7 TQ

CAN2510_PH2SEG_8TQ Longitud = 8 TQ

Ancho de segmento de fase 1

Controlado por los bits PH1SEG2:PH1SEG0 (registro CNF2)

CAN2510_PH1SEG_1TQ **Longitud = 1 TQ**

CAN2510_PH1SEG_2TQ Longitud = 2 TQ

CAN2510_PH1SEG_3TQ Longitud = 3 TQ

CAN2510_PH1SEG_4TQ Longitud = 4 TQ

CAN2510_PH1SEG_5TQ Longitud = 5 TQ

CAN2510_PH1SEG_6TQ Longitud = 6 TQ

CAN2510_PH1SEG_7TQ Longitud = 7 TQ

CAN2510_PH1SEG_8TQ Longitud = 8 TQ

Ancho del segmento de propagación

controlado por los bits PRSEG2:PRSEG0 (registro CNF2)

CAN2510_PROPSEG_1TQ **Longitud = 1 TQ**

CAN2510_PROPSEG_2TQ Longitud = 2 TQ

CAN2510_PROPSEG_3TQ Longitud = 3 TQ

CAN2510_PROPSEG_4TQ Longitud = 4 TQ

CAN2510_PROPSEG_5TQ Longitud = 5 TQ

CAN2510_PROPSEG_6TQ Longitud = 6 TQ

CAN2510_PROPSEG_7TQ Longitud = 7 TQ

CAN2510_PROPSEG_8TQ Fuente Longitud = 8 TQ

de fase 2 Controlada por el bit

BTLMODE (registro CNF2). Esto determina si la longitud de la Fase 2 está determinada por los bits PH2SEG2:PH2SEG0 o la mayor longitud de bits PH1SEG2:PH1SEG0 y (2TQ).

CAN2510_PH2SOURCE_PH2 Longitud = mayor ~~Longitud 25 PH2SEG2:PH2SEG0~~
PH1SEG2:PH1SEG0 y 2TQ

Frecuencia de punto de muestra de

bit Controlada por el bit SAM (registro CNF2). Esto determina si el bit se muestrea 1 o 3 veces en el punto de muestra.

CAN2510_SAMPLE_1x El bit se muestrea tres veces ~~bit SAM~~
~~bit SAM~~
~~bit SAM~~

CAN2510Init (continuación)Filtro de ruido del pin RX en modo de

suspensión controlado por el bit WAKFIL (registro CNF3). Esto determina si el pin RX utilizará un filtro para rechazar el ruido cuando el dispositivo está en modo de suspensión.

CAN2510_RX_FILTRO**Filtrado en el pin RX cuando está en modo de suspensión****CAN2510_RX_SIN_FILTRO**

Sin filtrado en el pin RX cuando está en modo de suspensión

interruptEnables El valor

de interruptEnables puede ser una combinación de los siguientes valores, combinados mediante una operación AND (&) bit a bit. La opción en **negrita** es el valor predeterminado. Controlado por todos los bits del registro CANINTE.

CAN2510_NONE_ES**No hay interrupciones habilitadas****CAN2510_MSGERR_ES**

Interrupción por error durante la recepción o transmisión del mensaje

CAN2510_WAKEUP_ES

Interrupción en la actividad del bus CAN
Interrupción por cambio de condición de error de EFLG

CAN2510_ERROR_ES**CAN2510_TXB2_ES**

Interrupción en el búfer de transmisión 2 quedando vacío

CAN2510_TXB1_ES

Interrupción en el búfer de transmisión 1 quedando vacío

CAN2510_TXB0_ES

Interrupción en el búfer de transmisión 0 quedando vacío

CAN2510_RXB1_ES

Interrumpir cuando se recibe un mensaje en el búfer de recepción 1

CAN2510_RXB0_ES

Interrumpir cuando se recibe un mensaje en el búfer de recepción 0

Modo SPI_sync

Especifica la frecuencia de sincronización de PIC18CXXX SPI:

CAN2510_SPI_FOSC4

Se comunica en FOSC/4

CAN2510_SPI_FOSC16

Comunica en FOSC/16

CAN2510_SPI_FOSC64

Se comunica en FOSC/64

CAN2510_SPI_FOSCTMR2

Se comunica en TMR2/2

SPI_busMode

Especifica el modo de bus PIC18CXXX SPI:

CAN2510_SPI_MODE00

Comunicarse usando el modo SPI 00

CAN2510_SPI_MODE01

Comunicarse usando el modo SPI 01

SPI_smpPhase

Especifica el punto de muestra PIC18CXXX SPI: **Muestras**

CAN2510_SPI_SMPMID Muestras al final del bit ~~SPSMODE00~~ **SPSMODE01** Esta SPI, restablece el dispositivo MCP2510 (si se solía inicializar el modo PIC18CXXX

Observaciones:

MCP2510.

Nota: Cuando se completa esta función, el MCP2510 se deja en el Modo de configuración.

Valor de retorno:

Indica si se pudo inicializar el MCP2510. 0 si la inicialización se completó -1 si la inicialización no se completó

Nombre del archivo:

caninit.c

CAN2510 Habilitar interrupción

Función: Modifica los bits de habilitación de interrupción CAN2510 (registro CANINTE) a los nuevos valores.

CAN requerido**Modo(s):**

Todas

Incluir:can2510.h,
spi_can.h**Prototipo:**void CAN2510InterruptEnable(unsigned char
 interruptEnables);**Argumentos:****interruptEnables** El valorde **interruptEnables** puede ser una combinación de los siguientes valores, combinados mediante una operación AND (&) bit a bit. La opción en **negrita** es el valor predeterminado. Controlado por todos los bits del registro CANINTE.

CAN2510_NONE_EN	Sin interrupciones habilitadas (00000000)
CAN2510_MSGERR_ES	Interrupción por error durante la recepción o transmisión del mensaje (10000000)

CAN2510_WAKEUP_ES	Interrupción en la actividad del bus CAN (01000000)
CAN2510_ERROR_ES	Interrupción por cambio de condición de error de EFLG (00100000)

CAN2510_TXB2_ES	Interrupción en el búfer de transmisión 2 quedando vacío (00010000)
-----------------	---

CAN2510_TXB1_ES	Interrupción en el búfer de transmisión 1 quedando vacío (00001000)
-----------------	---

CAN2510_RXB0_ES	Interrupción en el búfer de transmisión 0 quedando vacío (00000100)
-----------------	---

CAN2510_RXB1_EN	Interrumpir cuando se recibe un mensaje en el búfer de recepción 1 (00000010)
-----------------	---

CAN2510_RXB0_EN	Interrumpir cuando se recibe un mensaje en el búfer de recepción 0 (00000001)
-----------------	---

Observaciones: Esta función actualiza el registro CANINTE con el valor que se determina mediante la operación AND de las fuentes de interrupción deseadas.

Nombre del archivo: caninte.c

Bibliotecas del compilador MPLAB® C18 C

CAN2510 Estado de interrupción

Función:	Indica la fuente de la interrupción CAN2510.
CAN requerido	
Modo(s):	Todas
Incluir:	can2510.h, spi_can.h
Prototipo:	carácter sin firmar CAN2510InterruptStatus(vacío);
Observaciones:	Esta función lee el registro CANSTAT y especifica un código dependiendo del estado de los bits ICODE2:ICODE0.
Valor de retorno:	La función devuelve uno de los siguientes valores: CAN2510_NO_INTS No se produjeron interrupciones CAN2510_WAKEUP_INT Interrupción en la actividad del bus CAN CAN2510_ERROR_INT Interrupción por cambio de condición de error de EFLG CAN2510_TXB2_INT Interrupción en el búfer de transmisión 2 quedando vacío CAN2510_TXB1_INT Interrupción en el búfer de transmisión 1 quedando vacío CAN2510_RXB0_INT Interrupción en el búfer de transmisión 0 quedando vacío CAN2510_RXB1_INT Interrumpir cuando se recibe un mensaje en el búfer de recepción 1 CAN2510_RXB0_INT Interrumpir cuando se recibe un mensaje en el búfer de recepción 0
Nombre del archivo:	canints.c

CAN2510LoadBufferStd

Función:	Carga un marco de datos estándar en el búfer de transferencia especificado.
CAN requerido	
Modo(s):	Todas
Incluir:	can2510.h
Prototipo:	vacio CAN2510LoadBufferStd(unsigned char bufferNum , unsigned int msgId , unsigned char numBytes , unsigned char * data);
Argumentos:	bufferNum Especifica el búfer en el que cargar el mensaje. Uno de los siguientes valores: CAN2510_RXB0 CAN2510_TXB1 CAN2510_TXB2 Búfer de transmisión 0 Búfer de transmisión 1 Búfer de transmisión 2 msgId Identificador de mensaje CAN, hasta 11 bits para un mensaje estándar. numBytes Número de bytes de datos a transmitir, de 0 a 8. Si el valor es mayor que 8, solo se almacenarán los primeros 8 bytes de datos. datos Matriz de valores de datos que se van a cargar. La matriz debe ser al menos tan grande como el valor especificado en numBytes .

CAN2510LoadBufferStd (continuación)

Observaciones: Esta función carga la información del mensaje, pero no transmite el mensaje. Utilice la función CAN2510WriteBuffer() para escribir el mensaje en el bus CAN.

Esta función no establece la prioridad del búfer. Utilice la función CAN2510SetBufferPriority() para establecer la prioridad del búfer.

Nombre del archivo: canloads.c

CAN2510LoadBufferXtd

Función: Carga un marco de datos extendidos en el búfer de transferencia especificado.

CAN requerido

Todas

Modo(s):

Incluir:

can2510.h

Prototipo:

vacio CAN2510LoadBufferXtd(

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

Argumentos:

bufferNum

Especifica el búfer en el que cargar el mensaje. Uno de los siguientes valores:

CAN2510_TXB0

Búfer de transmisión 0

CAN2510_TXB1

Búfer de transmisión 1

CAN2510_TXB2

Búfer de transmisión 2

msgId

Identificador de mensaje CAN, hasta 29 bits para un mensaje extendido.

numBytes

Número de bytes de datos a transmitir, de 0 a 8. Si el valor es mayor que 8, solo se almacenarán los primeros 8 bytes de datos.

data

Matriz de valores de datos a cargar. La matriz debe ser al menos tan grande como el valor especificado en **numBytes**.

Observaciones:

Esta función carga la información del mensaje, pero no transmite el mensaje. Utilice la función CAN2510WriteBuffer() para escribir el mensaje en el bus CAN.

Esta función no establece la prioridad del búfer. Utilice la función CAN2510SetBufferPriority() para establecer la prioridad del búfer.

Nombre del archivo:

canloadx.c

Bibliotecas del compilador MPLAB® C18 C

CAN2510CargarRTRStd

Función:	Carga una trama remota estándar en el búfer de transferencia especificado.	
CAN requerido		
Modo(s):	Todas	
Incluir:	can2510.h	
Prototipo:	vacío CAN2510LoadBufferStd(bufferNum char sin firmar, msgId int sin firmar , numBytes char sin firmar);	
Argumentos:	bufferNum	Especifica el búfer en el que cargar el mensaje. Uno de los siguientes valores:
	CAN2510_TXB0	Búfer de transmisión 0
	CAN2510_TXB1	Búfer de transmisión 1
	CAN2510_TXB2	Búfer de transmisión 2
	msgId	Identificador de mensaje CAN, hasta 11 bits para un mensaje estándar.
	numBytes	Número de bytes de datos a transmitir, de 0 a 8. Si el valor es mayor que 8, solo se almacenarán los primeros 8 bytes de datos.
Observaciones:	Esta función carga la información del mensaje, pero no transmite el mensaje. Utilice la función CAN2510WriteBuffer() para escribir el mensaje en el bus CAN.	
	Esta función no establece la prioridad del búfer. Utilice la función CAN2510SetBufferPriority() para establecer la prioridad del búfer.	
Nombre del archivo:	canlrrts.c	

CAN2510CargarRTRXtd

Función:	Carga una trama remota extendida en el búfer de transferencia especificado.	
CAN requerido		
Modo(s):	Todas	
Incluir:	can2510.h	
Prototipo:	vacío CAN2510LoadBufferXtd(bufferNum de caracteres sin firmar , msgId largo sin firmar , numBytes de caracteres sin firmar);	
Argumentos:	bufferNum	Especifica el búfer en el que cargar el mensaje. Uno de los siguientes valores: CAN2510_TXB0 CAN2510_TXB1 CAN2510_TXB2
		Búfer de transmisión 0
		Búfer de transmisión 1
		Búfer de transmisión 2
	msgId	Identificador de mensaje CAN, hasta 29 bits para un mensaje extendido.
	numBytes	Número de bytes de datos a transmitir, de 0 a 8. Si el valor es mayor que 8, solo se almacenarán los primeros 8 bytes de datos.
Observaciones:	Esta función carga la información del mensaje, pero no transmite el mensaje. Utilice la función CAN2510WriteBuffer() para escribir el mensaje en el bus CAN.	
	Esta función no establece la prioridad del búfer. Utilice la función CAN2510SetBufferPriority() para establecer la prioridad del búfer.	
Nombre del archivo:	canlrtrx.c	

Modo de lectura CAN2510

Función:	Lee el modo de funcionamiento actual del MCP2510.
CAN requerido	
Modo(s):	Todas
Incluir:	can2510.h
Prototipo:	carácter sin firmar CAN2510ReadMode(void);
Observaciones:	Esta función lee el modo de funcionamiento actual. El modo puede tener una solicitud pendiente para un nuevo modo.
Valor de retorno:	modo
	El valor de modo puede ser uno de los siguientes valores (definido en can2510.h). Especificado por los bits OPMODE2:OPMODE0 (registro CANSTAT). Uno de los siguientes valores: CAN2510_MODE_CONFIG
	Los registros de configuración se pueden modificar.
	CAN2510_MODE_NORMAL Normal (enviar y recibir mensajes)
	CAN2510_MODE_SLEEP Esperar por interrupción
	CAN2510_MODE_LISTEN Solo escucha, no envíes
	CAN2510_MODE_LOOPBACK Usado para pruebas, los mensajes permanecen internos
Nombre del archivo:	canmoder.c

CAN2510Estado de lectura

Función:	Lee el estado de los búferes de transmisión y recepción del MCP2510.
CAN requerido	
Modo(s):	Todas
Incluir:	can2510.h
Prototipo:	carácter sin firmar CAN2510ReadStatus(void);
Observaciones:	Esta función lee el estado actual de los búferes de transmisión y recepción.
Valor de retorno:	estado
	El valor de estado (un byte sin signo) tiene el siguiente formato: bit 7 bit 6 bit 5 bit 4 bit 3 bit 2 bit 1 bit 0 TXB2IF TXB2REQ TXB1IF TXB1REQ TXB0IF TXB0REQ RXB1IF RXB0IF
Nombre del archivo:	canstats.c

Bibliotecas del compilador MPLAB® C18 C

CAN2510Reiniciar

Función: Restablece el MCP2510.

CAN requerido

Modo(s): Todas

Incluir: can2510.h
spi_can.h spi.h

Prototipo: vacío CAN2510Reset (vacío);

Observaciones: Esta función reinicia el MCP2510.

Nombre del archivo: canreset.c

CAN2510Búfer de envío

Función: Solicita la transmisión de mensajes para los búferes de transmisión especificados.

CAN requerido

Modo(s): Modo normal

Incluir: can2510.h

Prototipo: vacío CAN2510WriteBuffer
(Número de búfer de caracteres sin signo);

Argumentos: **bufferNum**
Especifica el búfer para solicitar la transmisión. Uno de los siguientes valores:

CAN2510_TXB0	Búfer de transmisión 0
CAN2510_TXB1	Búfer de transmisión 1
CAN2510_TXB2	Búfer de transmisión 2
CAN2510_TXB0_B1	Transmitir el búfer 0 y el búfer 1
CAN2510_TXB0_B2	Transmitir el búfer 0 y el búfer 2
CAN2510_TXB1_B2	Transmitir el búfer 1 y el búfer 2
CAN2510_TXB0_B1_B2	Transmisión de búfer 0, búfer 1 y búfer 2 Esta función solicita la transmisión de un mensaje previamente cargado almacenado en los búferes especificados. Para cargar un mensaje, utilice las rutinas CAN2510LoadBufferStd() o CAN2510LoadBufferXtd().

Observaciones:

Nombre del archivo: cansend.c

CAN2510Lectura secuencial

Función: Lee la cantidad de bytes especificados en el MCP2510, comenzando en la dirección especificada. Estos valores se almacenarán en **DataArray**.

CAN requerido

Modo(s): Todas

Incluir: can2510.h

Prototipo: void CAN2510SequentialRead (caracter sin
firmar ***DataArray** char sin firmar
CAN2510addr char sin firmar **numbytes**);

Argumentos: **DataArray La**
dirección de inicio de la matriz de datos que almacena los datos de lectura secuencial.
CAN2510addr La
dirección del MCP2510 desde donde comienzan las lecturas secuenciales.
numbytes El
número de bytes para leer secuencialmente.

CAN2510Lectura secuencial (continuación)

Observaciones: Esta función lee bytes secuenciales del MCP2510 comenzando en la dirección especificada. Estos valores se cargan a partir de la primera dirección de la matriz que se especifica.

Nombre del archivo: readseq.c

CAN2510Escritura secuencial

Función: Escribe la cantidad de bytes especificados en el MCP2510, comenzando en la dirección especificada. Estos valores se escribirán desde **DataArray**.

CAN requerido

Modo(s): Todas

Incluir: can2510.h

Prototipo: void CAN2510SequentialWrite(caracter sin firmar
 ***DataArray** char sin firmar **CAN2510addr**
 char sin firmar **numbytes**);

Argumentos: **DataArray** La dirección de inicio de la matriz de datos que contiene los datos de escritura secuencial.

CAN2510addr La dirección del MCP2510 desde donde comienzan las escrituras secuenciales.

numbytes El número de bytes para escribir secuencialmente.

Observaciones: Esta función escribe bytes secuenciales en el MCP2510 comenzando en la dirección especificada. Estos valores están contenidos a partir de la primera dirección de la matriz que se especifica.

Nombre del archivo: wrtseq.c

CAN2510Establecer prioridad de búfer

Función: Carga la prioridad especificada para el búfer de transmisión especificado.

CAN requerido

Modo(s): Todas

Incluir: can2510.h

Prototipo: void CAN2510SetBufferPriority (número de búfer de caracteres sin firmar , **prioridad de búfer de caracteres sin firmar**);

Argumentos: **bufferNum**

Especifica el búfer para configurar la prioridad de. Uno de los siguientes valores:

CAN2510_TXB0	Búfer de transmisión 0
CAN2510_TXB1	Búfer de transmisión 1
CAN2510_TXB2	Búfer de transmisión 2

bufferPriority Prioridad

del búfer. Uno de los siguientes valores: Prioridad de

CAN2510_PRI_HIGHEST Prioridad de mensaje de CAN2510_PRI_HIGH

CAN2510_PRI_LOW Prioridad de mensaje de CAN2510_PRI_LOW

especificada de un búfer individual. Esta función carga la prioridad

Observaciones:

Nombre del archivo: cansetpr.c

Bibliotecas del compilador MPLAB® C18 C

Modo de configuración CAN2510

Función:	Configura el modo de funcionamiento del MCP2510.
CAN requerido	
Modo(s):	Todas
Incluir:	can2510.h
Prototipo:	void CAN2510SetMode (modo de caracteres sin firmar);
Argumentos:	modo El valor de modo puede ser uno de los siguientes valores (definido en can2510.h). Controlado por los bits REQOP2:REQOP0 (registro CANCTRL). Uno de los siguientes valores: CAN2510_MODE_CONFIG Los registros de configuración se pueden modificar. CAN2510_MODE_NORMAL Normal (enviar y recibir mensajes) CAN2510_MODE_SLEEP Esperar por interrupción CAN2510_MODE_LISTEN Solo escucha, no envíes CAN2510_MODE_LOOPBACK Usado para pruebas, los mensajes permanecen internos
Observaciones:	Esta función configura el modo especificado. El modo no cambiará hasta que se completen todas las transmisiones de mensajes pendientes.
Nombre del archivo:	camodes.c

CAN2510Establecer filtro de mensajes estándar

Función:	Configura TODOS los valores de filtro y máscara del búfer de recepción específico para un mensaje estándar.
CAN requerido	
Modo(s):	Todas
Incluir:	can2510.h
Prototipo:	carácter sin firmar CAN2510SetMsgFilterStd(número de búfer de caracteres sin firmar , máscara int sin firmar, int sin firmar * filtros);
Argumentos:	bufferNum Especifica el búfer de recepción para configurar la máscara y los filtros. Uno de los siguientes valores: CAN2510_RXB0 CAN2510_RXB1 Configurar RXM0, RXF0 y RXF1 Configurar RXM1, RXF2, RXF3, RXF4 y RXF5 máscara Valor a almacenar en la máscara correspondiente filtros Matriz de valores de filtro. Para mensajes de longitud estándar del búfer 0: matriz de 2 enteros sin signo Para mensajes de longitud estándar del búfer 1: matriz de 4 enteros sin signo Esta función configura el MCP2510 en el modo de configuración, Observaciones: luego escribe los valores de máscara y filtro en los registros apropiados. Antes de regresar, configura el MCP2510 al modo original.
Valor de retorno:	Indica si los modos del MCP2510 se pueden modificar correctamente. 0 si la inicialización y el restablecimiento del modo operativo se completaron -1 si la inicialización y el restablecimiento del modo operativo no se completaron
Nombre del archivo:	canfms.c

CAN2510SetMsgFilterXtd

Función: Configura TODOS los valores de filtro y máscara del búfer de recepción específico para un mensaje extendido.

CAN requerido**Modo(s):**

Todas

Incluir:

can2510.h

Prototipo:

carácter sin firmar CAN2510SetMsgFilterXtd(número de búfer
de caracteres sin firmar , **máscara** larga sin firmar ,
***filtros** largos sin firmar);

Argumentos:**bufferNum**

Especifica el búfer de recepción para configurar la máscara y los filtros para uno de los siguientes valores:

CAN2510_RXB0

Configurar RXM0, RXF0 y RXF1

CAN2510_RXB1

Configurar RXM1, RXF2, RXF3, RXF4 y

RXF5

máscara

Valor a almacenar en la máscara correspondiente

filtros

Matriz de valores de filtro.

Para búfer 0

Mensajes de longitud extendida: matriz de 2 enteros largos sin signo

Para tampón 1

Mensajes de longitud extendida: matriz de 4 enteros largos sin signo

Esta función configura el MCP2510 en el modo de configuración, luego escribe los valores de máscara y filtro en los registros apropiados. Antes de regresar, configura el MCP2510 al modo original.

Observaciones:

Indica si los modos del MCP2510 se pudieron modificar correctamente: 0 si se completó la inicialización y restauración del modo operativo -1 si no se completó la inicialización y restauración del modo operativo

Nombre del archivo:

canfmx.c

Bibliotecas del compilador MPLAB® C18 C

CAN2510Establecer filtro único estándar

Función:	Configura el filtro de recepción especificado con un valor de filtro para un mensaje estándar (Std).
CAN requerido	
Modo(s):	Modo de configuración
Incluir:	can2510.h
Prototipo:	<code>void CAN2510SetSingleFilterStd(filtro de caracteres sin firmarNum, filtro int sin firmar);</code>
Argumentos:	<p>filterNum</p> <p>Especifica el filtro de aceptación a configurar. Uno de los siguientes valores: CAN2510_RXF0 Configurar RXF0 (para RXB0) CAN2510_RXF1 Configurar RXF1 (para RXB0) CAN2510_RXF4 Configurar RXF4 (para RXB1) CAN2510_RXF5 Configurar RXF5 (para RXB1)</p> <p>filtro</p> <p>Valor para almacenar en el filtro correspondiente</p>
Observaciones:	Esta función escribe el valor del filtro en los registros apropiados. El MCP2510 debe estar en modo Configuración antes de ejecutar esta función.
Nombre del archivo:	canfilt.c

CAN2510SetSingleFilterXtd

Función:	Configura el filtro de recepción especificado con un valor de filtro para un mensaje extendido (Xtd).
CAN requerido	
Modo(s):	Modo de configuración
Incluir:	can2510.h
Prototipo:	<code>void CAN2510SetSingleFilterXtd(filtro de caracteres sin firmarNum, filtro largo sin firmar);</code>
Argumentos:	<p>filterNum</p> <p>Especifica el filtro de aceptación a configurar. Uno de los siguientes valores: CAN2510_RXF0 Configurar RXF0 (para RXB0) CAN2510_RXF1 Configurar RXF1 (para RXB0) CAN2510_RXF2 Configurar RXF2 (para RXB1) CAN2510_RXF3 Configurar RXF3 (para RXB1) CAN2510_RXF4 Configurar RXF4 (para RXB1) CAN2510_RXF5 Configurar RXF5 (para RXB1)</p> <p>filtro</p> <p>Valor para almacenar en el filtro correspondiente</p>
Observaciones:	Esta función escribe el valor del filtro en los registros correspondientes. El MCP2510 debe estar en modo Configuración antes de ejecutar esta función.
Nombre del archivo:	canfiltx.c

CAN2510SetSingleMaskStd

Función:	Configura la máscara de búfer de recepción especificada con un valor de máscara para un mensaje de formato estándar (Std).				
CAN requerido					
Modo(s):	Modo de configuración				
Incluir:	can2510.h				
Prototipo:	carácter sin firmar CAN2510SetSingleMaskStd(carácter sin firmar maskNum , máscara int sin firmar);				
Argumentos:	<p>número de máscara</p> <p>Especifica la máscara de aceptación a configurar. Uno de los siguientes valores:</p> <table border="0"> <tr> <td>CAN2510_RXM0</td> <td>Configurar RXM0 (para RXB0)</td> </tr> <tr> <td>CAN2510_RXM1</td> <td>Configurar RXM1 (para RXB1)</td> </tr> </table>	CAN2510_RXM0	Configurar RXM0 (para RXB0)	CAN2510_RXM1	Configurar RXM1 (para RXB1)
CAN2510_RXM0	Configurar RXM0 (para RXB0)				
CAN2510_RXM1	Configurar RXM1 (para RXB1)				
Observaciones:	máscara Valor para almacenar en la máscara correspondiente Esta función escribe el valor de la máscara en los registros apropiados. El MCP2510 debe estar en modo Configuración antes de ejecutar esta función.				
Nombre del archivo:	canmasks.c				

CAN2510SetSingleMaskXtd

Función:	Configura la máscara de búfer de recepción especificada con un valor de máscara para un mensaje extendido (Xtd).						
CAN requerido							
Modo(s):	Modo de configuración						
Incluir:	can2510.h						
Prototipo:	carácter sin firmar CAN2510SetSingleMaskXtd(carácter sin firmar maskNum , máscara larga sin firmar);						
Argumentos:	<p>número de máscara</p> <p>Especifica la máscara de aceptación a configurar. Uno de los siguientes valores:</p> <table border="0"> <tr> <td>CAN2510_RXM0</td> <td>Configurar RXM0</td> <td>(para RXB0)</td> </tr> <tr> <td>CAN2510_RXM1</td> <td>Configurar RXM1</td> <td>(para RXB1)</td> </tr> </table> <p>máscara Valor para almacenar en la máscara correspondiente Esta función escribe el valor de la máscara en los registros apropiados. El MCP2510 debe estar en modo Configuración antes de ejecutar esta función.</p>	CAN2510_RXM0	Configurar RXM0	(para RXB0)	CAN2510_RXM1	Configurar RXM1	(para RXB1)
CAN2510_RXM0	Configurar RXM0	(para RXB0)					
CAN2510_RXM1	Configurar RXM1	(para RXB1)					
Observaciones:							
Nombre del archivo:	canmaskx.c						

Bibliotecas del compilador MPLAB® C18 C

CAN2510Búfer de escritura

Función:	Inicia la transmisión del mensaje CAN del búfer seleccionado.
CAN requerido	
Modo(s):	Todas
Incluir:	can2510.h
Prototipo:	carácter sin firmar CAN2510WriteBuffer (número de búfer de caracteres sin fírmar)
Argumentos:	bufferNum Especifica el búfer en el que cargar el mensaje. Uno de los siguientes valores: CAN2510_TXB0 Búfer de transmisión 0 CAN2510_TXB1 Búfer de transmisión 1 CAN2510_TXB2 Búfer de transmisión 2
Observaciones:	Esta función inicia la transmisión del búfer de transmisión seleccionado.
Nombre del archivo:	canwrbuf.c

CAN2510Escritura estándar

Función:	Escribe un mensaje de formato estándar en el bus CAN utilizando el primer búfer de transmisión disponible.
CAN requerido	
Modo(s):	Modo normal
Incluir:	can2510.h
Prototipo:	carácter sin firmar CAN2510WriteStd(int sin firmar msgId, carácter sin firmar msgPriority , carácter sin firmar numBytes , carácter sin firmar * datos);
Argumentos:	<p>msgId Identificador de mensaje CAN, 11 bits para un mensaje estándar. Este identificador de 11 bits se almacena en los 11 bits inferiores de msgId (un número entero sin signo).</p> <p>msgPriority Prioridad del búfer. Uno de los siguientes valores: CAN2510_PRI_HIGHEST Mensaje de Mensaje de prioridad más alta intermedia baja CAN2510_PRI_LOW Mensaje de PRI_MITIGATION Mensaje de prioridad CAN2510_PRI_LOWEST</p> <p>numBytes Número de bytes de datos a transmitir, de 0 a 8. Si el valor es mayor a 8, solo se enviarán los primeros 8 bytes de datos.</p> <p>datos Matriz de valores de datos que se van a escribir. Debe ser al menos tan grande como el valor especificado en numBytes.</p>
Observaciones:	Esta función consultará cada búfer de transmisión en busca de un mensaje pendiente y publicará el mensaje especificado en el primer búfer disponible.
Valor de retorno:	El valor indica qué búfer se utilizó para transmitir el mensaje (0, 1 o 2). -1 indica que no se envió ningún mensaje.
Nombre del archivo:	canwrits.c

Bibliotecas del compilador MPLAB® C18 C

CAN2510WriteXtd

Función:	Escribe un mensaje de formato extendido en el bus CAN utilizando el primer búfer de transmisión disponible.
CAN requerido	
Modo(s):	Modo normal
Incluir:	can2510.h
Prototipo:	carácter sin firmar CAN2510WriteXtd(mensaje largo sin firmar , msgPrioridad de carácter sin firmar, numBytes de carácter sin firmar, * datos de carácter sin firmar);
Argumentos:	msgId Identificador de mensaje CAN, 29 bits para un mensaje extendido. Este identificador de 29 bits se almacena en los 29 bits inferiores de msgId (un largo sin firmar). msgPriority Prioridad del búfer. Uno de los siguientes valores: alta CAN2510_PRI_HIGHEST Mensaje de prioridad más alta prioridad intermedia CAN2510_PRI_CAN25Mensajes de prioridad media CAN2510_PRI_NORMAL Mensaje de prioridad media CAN2510_PRI_LOWEST Mensaje de prioridad más baja
numBytes	Número de bytes de datos a transmitir, de 0 a 8. Si el valor es mayor a 8, solo se enviarán los primeros 8 bytes de datos.
datos	Matriz de valores de datos que se van a escribir. Debe ser al menos tan grande como el valor especificado en numBytes.
Observaciones:	Esta función consultará cada búfer de transmisión en busca de un mensaje pendiente y publicará el mensaje especificado en el primer búfer disponible.
Valor de retorno:	El valor indica qué búfer se utilizó para transmitir el mensaje (0, 1 o 2). -1 indica que no se envió ningún mensaje.
Nombre del archivo:	canwritx.c

3.4 FUNCIONES DEL SOFTWARE I2C

Estas funciones están diseñadas para permitir la implementación de un bus I2C utilizando pines de E/S de un microcontrolador PIC18. Se proporcionan las siguientes funciones:

TABLA 3-6: FUNCIONES DEL SOFTWARE I2C

Función	Descripción
Reloj_prueba	Genera un retraso para el estiramiento del reloj esclavo.
SWAckI2C	Genere una condición de reconocimiento de bus I2C .
SWGetI2C	Lee un byte del bus I2C .
SWGetsI2C	Leer una cadena de datos.
SWNotAckI2C	Genere una condición de no reconocimiento del bus I2C .
SWPutClI2C	Escriba un solo byte en el bus I2C .
SWPutslI2C	Escriba una cadena en el bus I2C .
SWReadI2C	Lee un byte del bus I2C .
SWRestartI2C	Genere una condición de reinicio del bus I2C .
Inicio SWI2C	Genere una condición de inicio de bus I2C .
SWStopI2C	Genere una condición de parada de bus I2C .
SWEscribirI2C	Escriba un solo byte en el bus I2C .

Las versiones precompiladas de estas funciones utilizan asignaciones de pines predeterminadas que se pueden cambiar redefiniendo las asignaciones de macros en el archivo sw_i2c.h, que se encuentra en el subdirectorio h de la instalación del compilador:

TABLA 3-7: MACROS PARA SELECCIONAR ASIGNACIONES DE PIN I2C

Línea I2C	macros	Valor por defecto	Utilizar
Pin de DATOS	DATA_PIN	PUERTOBBits.RB4	Pin utilizado para la línea de DATOS.
	DATA_LAT	LATBBits.RB4	Pestillo asociado con el pin DATA.
	DATOS_BAJOS	TRISBbits.TRISB4 = 0;	Declaración para configurar el pin DATA como salida.
	DATA_HI	TRISBbits.TRISB4 = 1;	Declaración para configurar el pin DATA como entrada.
Pin RELOJ SCLK_PIN	SCLK_PIN	PUERTOBBits.RB3	Pin utilizado para la línea CLOCK.
	SCLK_LAT	LATBBits.LATB3	Pestillo asociado al pin CLOCK.
	RELOJ_BAJO	TRISBbits.TRISB3 = 0;	Declaración para configurar el pin CLOCK como salida.
	RELOJ_HI	TRISBbits.TRISB3 = 1;	Declaración para configurar el pin CLOCK como entrada.

Una vez realizadas estas definiciones, el usuario debe volver a compilar las rutinas I2C y luego utilizar los archivos actualizados en el proyecto. Esto se puede lograr agregando los archivos fuente de la biblioteca al proyecto o volviendo a compilar los archivos de la biblioteca utilizando los archivos por lotes proporcionados.

Bibliotecas del compilador MPLAB® C18 C

3.4.1 Descripciones de funciones

Reloj_prueba

Función:	Genera un retraso para el estiramiento del reloj esclavo.
Incluir:	sw_i2c.h
Prototipo:	char prueba_reloj (vacío); Esta función
Observaciones:	se llama para permitir el estiramiento del reloj esclavo. Es posible que sea necesario ajustar el tiempo de demora según los requisitos de la aplicación. Si al final del período de retraso la línea de reloj es baja, se devuelve un valor que indica reloj error.
Valor de retorno:	Se devuelve 0 si no se produjo ningún error de reloj. Se devuelve 2 si se produjo un error de reloj.
Nombre del archivo:	swctti2c.c

SWAckI2C

SWNotAckI2C

Función:	Genere una condición de reconocimiento/no reconocimiento del bus I2C .
Incluir:	sw_i2c.h
Prototipo:	char SWAckI2C(vacío); char SWNotAckI2C(vacío);
Observaciones:	Esta función se llama para generar una secuencia de reconocimiento de bus I2C . 0
Valor de retorno:	si el esclavo reconoce -1 si el esclavo no reconoce
Nombre del archivo:	swacki2c.c

SWGetI2C

Consulte **SWReadI2C**.

SWGetsI2C

Función:	Ler una cadena del bus I2C .
Incluir:	sw_i2c.h char SWGetsI2C(unsigned
Prototipo:	char *rdptr, unsigned char length);
Argumentos:	rdptr Ubicación para almacenar los datos leídos del bus I2C . longitud Número de bytes a leer.
Observaciones:	Esta función lee una cadena de longitud predeterminada. -1 si el maestro generó una condición de bus NOT ACK antes de que se hayan recibido todos los bytes 0 de lo contrario
Valor de retorno:	maestro generó una condición de bus NOT ACK antes de que se hayan recibido todos los bytes 0 de lo contrario
Nombre del archivo:	swgtsi2c.c
Ejemplo de código:	caract.x[10]; SWGetsI2C(x,5);

SWNotAckI2C

Ver **SWAckI2C**.

SWPutclI2C

Consulte **SWWriteI2C**.

SWPutsI2C

Función:	Escriba una cadena en el bus
Incluir:	I2C . sw_i2c.h
Prototipo:	char SWPutsI2C(carácter sin firmar *wrdptr);
Argumentos:	wrdptr Puntero a los datos que se escribirán en el bus I2C .
Observaciones:	Esta función escribe una cadena de datos hasta (pero sin incluir) un carácter nulo.
Valor de retorno:	-1 si hubo un error al escribir en el bus I2C 0 de lo contrario
Nombre del archivo:	swptsi2c.c
Ejemplo de código:	char mybuff [] = "Hola"; SPutsI2C(mybuff);

SWReadlI2C

SWGtclI2C

Función:	Lee un byte del bus I2C . sw_i2c.h
Incluir:	
Prototipo:	char SWReadlI2C(vacío);
Observaciones:	Esta función lee en un solo byte de datos generando el apropiado señales en la línea de reloj I2C predefinida.
Valor de retorno:	Esta función devuelve el byte de datos I2C adquirido. -1 si hubo un error en esta función.
Nombre del archivo:	swgtci2c.c

SWRestartI2C

Función:	Genere una condición de bus de reinicio I2C .
Incluir:	sw_i2c.h
Prototipo:	vacío SWRestartI2C (vacío);
Observaciones:	Se llama a esta función para generar una condición de reinicio del bus I2C .

Bibliotecas del compilador MPLAB® C18 C

Inicio SWI2C

Función: Genere una condición de inicio de bus I2C .
Incluir: sw_i2c.h
Prototipo: vacío SWStartI2C (vacío);
Observaciones: Esta función se llama para generar una condición de inicio de bus I2C .
Nombre del archivo: swstri2c.c

SWStopI2C

Función: Genere una condición de parada de bus I2C . sw_i2c.h vacío SWStopI2C (vacío);
Incluir: I2C . sw_i2c.h vacío SWStopI2C (vacío);
Prototipo: Esta función se llama para generar una condición de parada de bus I2C . swstpi2c.c
Observaciones:
Nombre del archivo:

SWEscribirI2C

SWPutclI2C

Función: Escribe un byte en el bus I2C .
Incluir: sw_i2c.h
Prototipo: char SWEscribirI2C(
 data_out de caracteres sin firmar);
Argumentos: salida_datos
Byte de datos único que se escribirá en el bus I2C .
Observaciones: Esta función escribe un solo byte de datos en el pin de datos predefinido.
Valor de retorno: 0 si la escritura es exitosa
-1 si hubo una condición de error
Nombre del archivo: swptci2c.c
Ejemplo de código si (SW Writel2C (0x80))
{
 manejador de errores();
}

3.4.2 Ejemplo de uso El

siguiente es un ejemplo de código simple que ilustra una implementación de software I2C que se comunica con un dispositivo de memoria Microchip 24LC01B I2C EE.

```
#incluye <p18cxx.h> #incluye
<sw_i2c.h> #incluye <retrasos.h>
```

```
// FUNCIÓN Prototipo void
main(void); byte_write vacío
(vacío); página vacía_escribir
(vacío); void dirección_actual(void);
vacío random_read (vacío); vacío
secuencial_read (vacío); vacío ack_poll
(vacío); char sin firmar warr[] =
{8,7,6,5,4,3,2,1,0}; char sin firmar rarr[15];
char sin firmar lejano *rdptr = rarr; char sin firmar lejos *wrptr = warr;
var char sin firmar;
```

```
#define W_CS PORTA.2
//***** vacío principal (vacío) {
```

```
byte_write();
ack_poll();
pagina_escribir();
ack_poll();
nop();
lectura_secuencial();
nop();
mientras (1); // Bucle indefinidamente
}
```

```
byte_write vacío (vacío) {
    SWIInicI2C(); var =
    SWPutI2C(0xA0); // byte de control
    SWAckI2C(); var
    = SWPutI2C(0x10); // dirección de la palabra
    SWAckI2C(); var
    = SWPutI2C(0x66); // datos
    SWAckI2C();
    SWStopI2C();
}
```

```
página vacía_escribir (vacío) {
    SWIInicI2C(); var =
    SWPutI2C(0xA0); // byte de control
    SWAckI2C(); var
    = SWPutI2C(0x20); // dirección de la palabra
    SWAckI2C(); var
    = SWPutI2C(wrptr); // datos
    SWStopI2C();
}
```

Bibliotecas del compilador MPLAB® C18 C

```
void lectura_secuencial( void ) {  
  
    SWIniciarI2C(); var =  
    SWPutcl2C( 0xA0 ); // byte de control  
    SWAckl2C(); var  
    = SWPutcl2C( 0x00 ); // dirección para leer  
    SWAckl2C();  
    SWReiniciarI2C(); var =  
    SWPutcl2C( 0xA1 );  
    SWAckl2C(); var  
    = SWGetsl2C(rptr, 9);  
    SWStopl2C();  
}  
  
void dirección_actual( void ) {  
  
    SWIniciarI2C();  
    SWPutcl2C( 0xA1 ); // byte de control  
    SWAckl2C();  
    SWGetcl2C();           // dirección de la palabra  
    SWNotAckl2C();  
    SWStopl2C();  
}  
  
nulo ack_poll (vacío) {  
  
    SWIniciarI2C(); var =  
    SWPutcl2C( 0xA0 ); // byte de control while( SWAckl2C() ) {  
  
        SWReiniciarI2C(); var =  
        SWPutcl2C(0xA0); // datos  
    }  
    SWStopl2C();  
}
```

3.5 FUNCIONES DEL SOFTWARE SPI®

Estas funciones están diseñadas para permitir la implementación de un SPI utilizando pines de E/S de un microcontrolador PIC18. Se proporcionan las siguientes funciones:

TABLA 3-8: FUNCIONES DEL SOFTWARE SPI

Función	Descripción
BorrarCSSWSPI	Borre el pin Chip Select (CS).
AbrirSWSPI	Configure los pines de E/S para usarlos como SPI.
putcSWSPI	Escriba un byte de datos en el software SPI.
EstablecerCSSWSPI	Configure el pin Chip Select (CS).
EscribirSWSPI	Escriba un byte de datos en el bus SPI del software.

Las versiones precompiladas de estas funciones usan asignaciones de pines predeterminadas que se pueden cambiar redefiniendo las asignaciones de macros en el archivo sw_spi.h, que se encuentra en el subdirectorio h de la instalación del compilador:

TABLA 3-9: MACROS PARA SELECCIONAR ASIGNACIONES DE PIN SPI

LCD Controlador Línea	macros	Valor por defecto	Utilizar
Pasador CS	SW_CS_PIN TRIS_SW_CS_PIN	PUERTOBBits.RB2 TRISBbits.TRISB2	Pin utilizado para la línea Chip Select (CS). Bit que controla la dirección del pin _____ asociado a la línea CS.
Pasador DIN SW_DIN_PIN	SW_DIN_PIN TRIS_SW_DIN_PIN	PUERTOBBits.RB3 TRISBbits.TRISB3	Pin utilizado para la línea DIN. Bit que controla la dirección del pin asociado a la línea DIN.
PIN DOUT SW_DOUT_PIN	SW_DOUT_PIN TRIS_SW_DOUT_PIN	PUERTOBBits.RB7 TRISBbits.TRISB7	Pin utilizado para la línea DOUT. Bit que controla la dirección del pin asociado a la línea DOUT.
Pasador SCK SW_SCK_PIN	SW_SCK_PIN TRIS_SW_SCK_PIN	PUERTOBBits.RB6 TRISBbits.TRISB6	Pin utilizado para la línea SCK. Bit que controla la dirección del pin asociado a la línea SCK.

Bibliotecas del compilador MPLAB® C18 C

Las bibliotecas que se proporcionan pueden operar en uno de cuatro modos. La siguiente tabla enumera las macros utilizadas para seleccionar entre estos modos. Exactamente uno de estos debe definirse al reconstruir las bibliotecas SPI del software.

TABLA 3-10: MACROS PARA SELECCIÓN DE MODOS

Macro	Valor por defecto	Sentido
MODO0	definido	CKP = 0 CKE = 0
MODO1	no definida	CKP = 1 CKE = 0
MODO2	no definida	CKP = 0 CKE = 1
MODO3	no definida	CKP = 1 CKE = 1

Una vez realizadas estas definiciones, el usuario debe volver a compilar las rutinas SPI del software y luego incluir los archivos actualizados en el proyecto. Esto se puede lograr agregando los archivos fuente SPI del software al proyecto o recompilando los archivos de la biblioteca utilizando los archivos por lotes proporcionados.

3.5.1 Descripciones de funciones

BorrarCSSWSPI

Función:	Borre el pin Chip Select (CS) que se especifica en el archivo de encabezado sw_spi.h.
Incluir:	sw_spi.h
Prototipo:	vacio ClearCSSWSPI (vacío);
Observaciones:	Esta función borra el pin de E/S que se especifica en sw_spi.h para que sea el pin Chip Select (CS) para el software SPI.
Nombre del archivo:	clrcsspi.c

AbrirSWSPI

Función:	Configure los pines de E/S para el software SPI.
Incluir:	sw_spi.h
Prototipo:	vacio OpenSWSPI (vacío);
Observaciones:	Esta función configura los pines de E/S utilizados para el software SPI en el estado de entrada o salida y el nivel lógico correctos.
Nombre del archivo:	abrespi.c

putcSWSPI

Consulte [Escribir SWSPI](#).

EstablecerCSSWSPI

Función: Configure el pin Chip Select (CS) que se especifica en el archivo de encabezado sw_spi.h.

Incluir: sw_spi.h

Prototipo: vacío SetCSSWSPI (vacío);

Observaciones: Esta función configura el pin de E/S que se especifica en sw_spi.h para que sea el pin Chip Select (CS) para el software SPI.

Nombre del archivo: secsspi.c

Escribir SWSPI**putcSWSPI**

Función: Escriba un byte en el software SPI.

Incluir: sw_spi.h

Prototipo: char WriteSWSPI (datos **de** char);

Argumentos: **datos**
Datos a escribir en el software SPI.

Observaciones: Esta función escribe el byte de datos especificado en el software SPI y devuelve el byte de datos que se leyó. Esta función no proporciona ningún control del pin Chip Select (CS).

Valor de retorno: Esta función devuelve el byte de datos que se leyó de los datos en el pin (DIN) del software SPI.

Nombre del archivo: wrtsspi.c

Ejemplo de código: dirección de caracteres =
0x10; resultado de
caracteres; resultado = WriteSWSPI(dirección);

3.5.2 Ejemplo de uso

```
#incluye <cp18C452.h> #incluye
<sw_spi.h> #incluye <retasos.h>

vacío principal (vacío) {

    dirección de caracteres;

    // configurar software SPI
    AbrirSWSPI();

    for( dirección=0; dirección<0x10; dirección++ ) {

        BorrarCSSWSPI();           //borrar pin CS //enviar
        Escribir SWSPI (0x02);     escribir cmd
        WriteSWSPI(dirección);    //enviar direccion hola
        WriteSWSPI(dirección);    //enviar direccion baja
        SetCSSWSPI();              //establecer pin
        Retardo10KTCYx( 50 );      CS //esperar 5000,000TCY
    }
}
```

Bibliotecas del compilador MPLAB® C18 C

3.6 FUNCIONES DEL SOFTWARE UART

Estas funciones están diseñadas para permitir la implementación de un UART utilizando pines de E/S de un microcontrolador PIC18. Se proporcionan las siguientes funciones:

TABLA 3-11: FUNCIONES DEL SOFTWARE UART

Función	Descripción
obtenerUART	Lea un byte del software UART.
obtieneUART	Lea una cadena del software UART.
OpenUART	Configure los pines de E/S para usarlos como UART.
putcUART	Escriba un byte en el software UART.
pone UART	Escriba una cadena en el software UART.
LeerUART	Lea un byte del software UART.
escribirUART	Escriba un byte en el software UART.

Las versiones precompiladas de estas funciones usan asignaciones de pines predeterminadas que se pueden cambiar redefiniendo las declaraciones de equivalencia (equ) en los archivos wriuart.asm, readuart.asm y openuart.asm, que se encuentran en src/tradicional/pmc/sw_uart o scr/ subdirectorio extended/pmc/sw_uart de la instalación del compilador:

TABLA 3-12: MACROS PARA SELECCIONAR ASIGNACIONES DE PIN DE UART

LCD Controlador Línea	Definición	Valor por defecto	Utilizar
Pasador TX	SWTXD	PUERTOB	Puerto utilizado para la línea de transmisión.
	PIN SWTXD	4	Bit en el puerto SWTXD utilizado para la línea TX.
	TRIS_SWTXD	TRIB	Registro de dirección de datos asociado con el puerto utilizado para la línea TX.
Pasador RX	SWRXD	PUERTOB	Puerto utilizado para la línea de recepción.
	PIN SWRXD	5	Bit en el puerto SWRXD utilizado para la línea RX.
	TRIS_SWRXD	TRIB	Registro de dirección de datos asociado al puerto utilizado para la línea RX.

Si se realizan cambios en estas definiciones, el usuario debe volver a compilar las rutinas UART del software y luego incluir los archivos actualizados en el proyecto. Esto se puede lograr agregando los archivos de origen del software UART al proyecto o volviendo a compilar los archivos de la biblioteca utilizando los archivos por lotes provistos con la instalación del compilador MPLAB C18.

Las bibliotecas UART también requieren que el usuario defina las siguientes funciones para proporcionar los retrasos apropiados:

TABLA 3-13: FUNCIONES DE RETARDO UART DEL SOFTWARE

Función	Conducta
RetrasoTXBitUART	Retraso para: ((((2*FOSC) / (4*baudios)) + 1) / 2) - 12 ciclos
RetardoRXHalfBitUART	Retraso para: ((((2*FOSC) / (8*baudios)) + 1) / 2) - 9 ciclos
RetrasoRXBitUART	Retraso para: ((((2*FOSC) / (4*baudios)) + 1) / 2) - 14 ciclos

3.6.1 Descripciones de funciones**obtenercUART**Consulte **ReadUART**.**obtieneUART****Función:** Lea una cadena del software UART.**Incluir:** sw_uart.h**Prototipo:** void getsUART(char * **buffer**, char **len sin firmar**);**Argumentos:** **buffer**Puntero a la cadena de caracteres leída del software UART. **len** Número de caracteres que se leerán desde el software UART.**Observaciones:** Esta función lee los caracteres **len** del software UART y los coloca en **el búfer**.**Nombre del archivo:** getsuart.c**Ejemplo de código:** caract.x[10];
obtiene UART(x, 5);**OpenUART****Función:** Configure los pines de E/S para el software UART.**Incluir:** sw_uart.h**Prototipo:** vacío OpenUART (vacío);**Observaciones:** Esta función configura los pines de E/S utilizados para el software UART en el estado de entrada o salida y el nivel lógico correctos.**Nombre del archivo:** openuart.asm**Ejemplo de código:** OpenUART();**putcUART**Consulte **WriteUART**.**pone UART****Función:** Escriba una cadena en el software UART.**Incluir:** sw_uart.h void putsUART(char * **buffer**);**Prototipo:****Argumentos:** **buffer**

Cadena que se escribirá en el software UART.

Observaciones: Esta función escribe una cadena de caracteres en el UART del software. La cadena completa, incluido el valor nulo, se envía a la UART.**Nombre del archivo:** putsuart.c**Ejemplo de código:** char mybuff [] = "Hola";
putsUART(mybuff);

Bibliotecas del compilador MPLAB® C18 C

LeerUART

getcUART

Función:	Lea un byte del software UART. sw_uart.h
Incluir:	
Prototipo:	char ReadUART (vacío);
Observaciones:	Esta función lee un byte de datos del software UART.
Valor de retorno:	Devuelve el byte de datos que se leyó desde el pin de recepción de datos (RXD) del software UART.
Nombre del archivo:	readuart.asm
Ejemplo de código:	carácter x; x = LeerUART();

Escribir

UART putcUART

Función:	Escriba un byte en el software UART.
Incluir:	sw_uart.h void WriteUART (datos de caracteres);
Prototipo:	caracteres);
Argumentos:	data Byte de datos que se escribirán en el software UART.
Observaciones:	Esta función escribe el byte de datos especificado en el software UART.
Nombre del archivo:	escribir.asm
Ejemplo de código:	carácter x = 'H'; WriteUART(x);

3.6.2 Ejemplo de uso

```
#incluir <p18C452.h> #incluir
<sw_uart.h>

vacío principal (vacío) {

    datos de caracteres;

    // configurar software UART
    OpenUART();

    mientras ( 1 ) {

        datos = LeerUART(); //leer un byte
        WriteUART (datos); // rebotarlo
    }
}
```



Capítulo 4. Biblioteca de software general

4.1 INTRODUCCIÓN

Este capítulo documenta las funciones generales de la biblioteca de software que se encuentran en el archivo de biblioteca C estándar precompilado. El código fuente de todas estas funciones se incluye con MPLAB C18 en los siguientes subdirectorios de la instalación del compilador:

- src\tradicional\stdlib
- src\extended\stdlib
- src\tradicional\retrasos
- src\extended\delays

Las siguientes categorías de rutinas son compatibles con la biblioteca MPLAB C18:

- Funciones de clasificación de caracteres
- Funciones de conversión de datos
- Funciones de manipulación de cadenas y memoria •
- Funciones de retardo • Funciones de reinicio
- Funciones de salida de caracteres

4.2 FUNCIONES DE CLASIFICACIÓN DE CARACTERES

Estas funciones son coherentes con las funciones de la biblioteca C estándar ANSI 1989 del mismo nombre. Se proporcionan las siguientes funciones:

TABLA 4-1: FUNCIONES DE CLASIFICACIÓN DE CARACTERES

Función	Descripción
isalnum	Determinar si un carácter es alfanumérico.
isalfa	Determinar si un carácter es alfabético.
escntrl	Determinar si un personaje es un personaje de control.
esdigito	Determinar si un carácter es un dígito decimal.
isgrafó	Determinar si un carácter es un carácter gráfico.
es bajo	Determinar si un carácter es un carácter alfabético en minúsculas.
esimprimir	Determinar si un carácter es un carácter imprimible.
es puntito	Determinar si un carácter es un carácter de puntuación.
es espacio	Determine si un carácter es un carácter de espacio en blanco.
essuperior	Determine si un carácter es un carácter alfabético en mayúsculas.
esxdigito	Determinar si un carácter es un dígito hexadecimal.

Bibliotecas del compilador MPLAB® C18 C

4.2.1 Descripciones de funciones

isalnum

Función:	Determinar si un carácter es alfanumérico.
Incluir:	tipoc.h
Prototipo:	isalnum char sin firmar(char sin firmar ch);
Argumentos:	ch Carácter a comprobar.
Observaciones:	Se considera que un carácter es alfanumérico si está en el rango de 'A' a 'Z', 'a' a 'z' o '0' a '9'.
Valor de retorno:	Distinto de cero si el carácter es alfanumérico Cero de lo contrario
Nombre del archivo:	isalnum.c

isalpha

Función:	Determinar si un carácter es alfabetico.
Incluir:	tipoc.h
Prototipo:	isalpha del carácter sin firmar (caracter sin firmar ch);
Argumentos:	ch Carácter a comprobar.
Observaciones:	Se considera que un carácter es alfabetico si está en el rango de 'A' a 'Z' o 'a' a 'z'.
Valor de retorno:	Distinto de cero si el carácter es alfabetico cero en caso contrario
Nombre del archivo:	isalpha.c

escntrl

Función:	Determinar si un personaje es un personaje de control.
Incluir:	tipoc.h
Prototipo:	iscntrl de caracteres sin firmar (ch de caracteres sin firmar);
Argumentos:	ch Carácter a comprobar.
Observaciones:	Se considera que un carácter es un carácter de control si no es un carácter imprimible según lo definido por isprint().
Valor de retorno:	Distinto de cero si el carácter es un carácter de control cero en caso contrario
Nombre del archivo:	iscntrl.c

esdigito

Función:	Determinar si un carácter es un dígito decimal.
Incluir:	tipoc.h
Prototipo:	isdigit del carácter sin signo (caracter sin signo ch);
Argumentos:	ch Carácter a comprobar.
Observaciones:	Se considera que un carácter es un carácter de dígito si está en el rango de '0' a '9'.
Valor de retorno:	Distinto de cero si el carácter es un carácter de dígito cero en caso contrario
Nombre del archivo:	isdigit.c

isgrafo

Función:	Determinar si un carácter es un carácter gráfico.
Incluir:	tipoc.h
Prototipo:	carisgrafo sin firmar(char sin firmar ch);
Argumentos:	ch Carácter a comprobar.
Observaciones:	Se considera que un carácter es un carácter alfabético de caso gráfico si es cualquier carácter imprimible excepto el espacio.
Valor de retorno:	Distinto de cero si el carácter es un carácter gráfico Cero de lo contrario
Nombre del archivo:	isgraph.c

es bajo

Función:	Determinar si un carácter es un carácter alfabético en minúsculas.
Incluir:	tipoc.h
Prototipo:	islower de caracteres sin firmar(char sin firmar ch);
Argumentos:	ch Carácter a comprobar.
Observaciones:	Se considera que un carácter es un carácter alfabético en minúsculas si está en el rango de 'a' a 'z'.
Valor de retorno:	Distinto de cero si el carácter es un carácter alfabético en minúscula Cero de lo contrario
Nombre del archivo:	islower.c

Bibliotecas del compilador MPLAB® C18 C

esimprimir

Función:	Determinar si un carácter es un carácter imprimible.
Incluir:	tipoc.h
Prototipo:	impresión de caracteres sin firmar (ch de caracteres sin firmar);
Argumentos:	ch Carácter a comprobar.
Observaciones:	Se considera que un carácter es un carácter imprimible si está en el rango de 0x20 a 0x7e, inclusive.
Valor de retorno:	Distinto de cero si el carácter es un carácter imprimible cero en caso contrario
Nombre del archivo:	ispprint.c

es punto

Función:	Determinar si un carácter es un carácter de puntuación.
Incluir:	tipoc.h
Prototipo:	ispunct char sin firmar(char sin firmar ch);
Argumentos:	ch Carácter a comprobar.
Observaciones:	Un carácter se considera un carácter de puntuación si es un carácter imprimible que no es un espacio ni un carácter alfanumérico.
Valor de retorno:	Distinto de cero si el carácter es un carácter de puntuación cero en caso contrario
Nombre del archivo:	ispunct.c

es espacio

Función:	Determine si un carácter es un carácter de espacio en blanco.
Incluir:	tipoc.h
Prototipo:	espacio de caracteres sin firmar (char sin firmar ch);
Argumentos:	ch Carácter a comprobar.
Observaciones:	Se considera que un carácter es un carácter de espacio en blanco si es uno de los siguientes: espacio (' '), tabulador ('\t'), retorno de carro ('\r'), nueva línea ('\n'), avance de página ('\f') o pestaña vertical ('\v').
Valor de retorno:	Distinto de cero si el carácter es un carácter de espacio en blanco Cero de lo contrario
Nombre del archivo:	isspace.c

essuperior

Función:	Determine si un carácter es un carácter alfabético en mayúsculas.
Incluir:	tipoc.h
Prototipo:	isupper de caracteres sin firmar (char sin firmar ch);
Argumentos:	ch Carácter a comprobar.
Observaciones:	Se considera que un carácter es un carácter alfabético en mayúsculas si está en el rango de 'A' a 'Z'.
Valor de retorno:	Distinto de cero si el carácter es un carácter alfabético en mayúscula Cero de lo contrario
Nombre del archivo:	isupper.c

esxdígito

Función:	Determinar si un carácter es un dígito hexadecimal.
Incluir:	tipoc.h
Prototipo:	isxdigit del carácter sin firmar (caracter sin firmar ch);
Argumentos:	ch Carácter a comprobar.
Observaciones:	Se considera que un carácter es un carácter de dígito hexadecimal si está en el rango de '0' a '9', 'a' a 'f' o 'A' a 'F'.
Valor de retorno:	Distinto de cero si el carácter es un carácter de dígito hexadecimal Cero de lo contrario
Nombre del archivo:	isxdig.c

Bibliotecas del compilador MPLAB® C18 C

4.3 FUNCIONES DE CONVERSIÓN DE DATOS

Salvo que se indique lo contrario en las descripciones de las funciones, estas funciones son coherentes con las funciones de la biblioteca C estándar ANSI 1989 del mismo nombre. Las funciones proporcionadas son:

TABLA 4-2: FUNCIONES DE CONVERSIÓN DE DATOS

Función	Descripción
de la A, a la B	Convierte una cadena en un byte firmado de 8 bits.
en	Convierte una cadena en un valor de punto flotante.
atoi	Convierte una cadena en un entero con signo de 16 bits.
atolón	Convierte una cadena en una representación de entero largo.
btoa	Convierte un byte con signo de 8 bits en una cadena.
itoa	Convierte un entero de 16 bits con signo en una cadena.
Itoa	Convierte un entero largo con signo en una cadena.
rand	Genera un entero pseudoaleatorio.
srand	Establezca la semilla inicial para el generador de números pseudoaleatorios.
reducir	Convierte un carácter en un carácter ASCII alfabético en minúsculas.
topper	Convierte un carácter en un carácter ASCII alfabético en mayúsculas.
ultoa	Convierte un entero largo sin signo en una cadena.

4.3.1 Descripciones de funciones

de la A, a la B

Función:	Convierte una cadena en un byte firmado de 8 bits.
Incluir:	stdlib.h
Prototipo:	char firmado atob(const char * s);
Argumentos:	s Puntero a la cadena ASCII que se va a convertir.
Observaciones:	Esta función convierte la cadena ASCII s en un byte con signo de 8 bits (-128 a 127). La cadena de entrada debe estar en base 10 (base decimal) y puede comenzar con un carácter que indique el signo ('+' o '-'). Los resultados de desbordamiento no están definidos. Esta función es una extensión MPLAB C18 de las bibliotecas estándar ANSI.
Valor de retorno:	Byte con signo de 8 bits para todas las cadenas en el rango (-128 a 127).
Nombre del archivo:	atob.asm

en

Función:	Convierte una cadena en un valor de punto flotante.
Incluir:	stdlib.h
Prototipo:	atof doble (const char * s);
Argumentos:	s Puntero a la cadena ASCII que se va a convertir.
Observaciones:	Esta función convierte las cadenas ASCII en un valor de punto flotante. Ejemplos de cadenas de punto flotante que se reconocen son: -3.1415 1.0E2 1.0E+2 1.0E-2

Valor de retorno: La función devuelve el valor convertido.

Nombre del archivo: atof.c

atoi

Función:	Convierta una cadena en un entero con signo de 16 bits.
Incluir:	stdlib.h
Prototipo:	int atoi(const char * s);
Argumentos:	s
	Puntero a la cadena ASCII que se va a convertir.
Observaciones:	Esta función convierte la cadena ASCII s en un entero con signo de 16 bits (-32768 a 32767). La cadena de entrada debe estar en base 10 (base decimal) y puede comenzar con un carácter que indique el signo ('+' o '-'). Los resultados de desbordamiento no están definidos. Esta función es una extensión MPLAB C18 de las bibliotecas estándar ANSI.
Valor de retorno:	Entero con signo de 16 bits para todas las cadenas en el rango (-32768 a 32767).
Nombre del archivo:	atoi.asm

atolón

Función:	Convierta una cadena en una representación de entero largo.
Incluir:	stdlib.h
Prototipo:	atol largo(const char * s);
Argumentos:	s
	Puntero a la cadena ASCII que se va a convertir.
Observaciones:	Esta función convierte la cadena ASCII s en un valor largo. La cadena de entrada debe estar en base 10 (base decimal) y puede comenzar con un carácter que indique un signo ('+' o '-'). Los resultados de desbordamiento no están definidos. Esta función es una extensión MPLAB C18 de las bibliotecas estándar ANSI.
Valor de retorno:	La función devuelve el valor convertido.
Nombre del archivo:	atol.asm

btoa

Función:	Convierta un byte con signo de 8 bits en una cadena.
Incluir:	stdlib.h
Prototipo:	char * btoa(valor de carácter firmado, carácter * cadena);
Argumentos:	valor
	Un byte firmado de 8 bits. string Puntero a la cadena ASCII que contendrá el resultado. La cadena debe ser lo suficientemente larga para contener la representación ASCII, incluido el carácter de signo para valores negativos y un carácter nulo final.
Observaciones:	Esta función convierte el byte con signo de 8 bits del valor del argumento en una representación de cadena ASCII. Esta función es una extensión MPLAB C18 de las bibliotecas requeridas por ANSI.
Valor de retorno:	Puntero a la cadena de resultado .
Nombre del archivo:	btoa.asm

Bibliotecas del compilador MPLAB® C18 C

itoa

Función:	Convierte un entero de 16 bits con signo en una cadena.
Incluir:	stdlib.h
Prototipo:	char * itoa (valor int, char * cadena);
Argumentos:	valor Un entero de 16 bits con signo. string Puntero a la cadena ASCII que contendrá el resultado. La cadena debe ser lo suficientemente larga para contener la representación ASCII, incluido el carácter de signo para valores negativos y un carácter nulo final.
Observaciones:	Esta función convierte el entero de 16 bits con signo del valor del argumento en una representación de cadena ASCII. Esta función es una extensión MPLAB C18 de las bibliotecas requeridas por ANSI.
Valor de retorno:	Puntero a la cadena de resultado .
Nombre del archivo:	itoa.asm

Itoa

Función:	Convierta un entero largo con signo en una cadena.
Incluir:	stdlib.h
Prototipo:	char * Itoa(valor largo, carácter * cadena);
Argumentos:	valor Un entero largo con signo que se va a convertir. string Puntero a la cadena ASCII que contendrá el resultado.
Observaciones:	Esta función convierte el entero largo con signo en el valor del argumento en una representación de cadena ASCII. La cadena debe ser lo suficientemente larga para contener la representación ASCII, incluido el carácter de signo para valores negativos y un carácter nulo final. Esta función es una extensión de MPLAB C18 para las bibliotecas requeridas por ANSI.
Valor de retorno:	Puntero a la cadena de resultado .
Nombre del archivo:	Itoa.asm

rand

Función:	Genera un entero pseudoaleatorio.
Incluir:	stdlib.h
Prototipo:	int rand (vacío);
Observaciones:	Las llamadas a esta función devuelven valores enteros pseudoaleatorios en el rango [0,32767]. Para usar esta función de manera efectiva, debe inicializar el generador de números aleatorios usando la función srand(). Esta función siempre devolverá la misma secuencia de enteros cuando se utilicen valores iniciales idénticos.
Valor de retorno:	Un valor entero pseudoaleatorio.
Nombre del archivo:	rand.asm

strand

Función:	Establezca la semilla inicial para la secuencia de números pseudoaleatorios.
Incluir:	stdlib.h
Prototipo:	void srand (semilla int sin firmar);
Argumentos:	semilla
	El valor inicial de la secuencia de números pseudoaleatorios.
Observaciones:	Esta función establece la semilla inicial para la secuencia de números pseudoaleatorios generada por la función rand(). La función rand() siempre devolverá la misma secuencia de enteros cuando se utilicen valores iniciales idénticos. Si se llama a rand() sin haber llamado primero a srand(), la secuencia de números generados será la misma que si se hubiera llamado a srand() con un valor inicial de 1.

Nombre del archivo: rand.asm

reducir

Función:	Convierta un carácter en un carácter ASCII alfabético en minúsculas.
Incluir:	tipoc.h
Prototipo:	char tolower(char ch);
Argumentos:	ch
	Personaje a convertir.
Observaciones:	Esta función convierte ch en un carácter ASCII alfabético en minúscula siempre que el argumento sea un carácter alfabético en mayúscula válido.
Valor de retorno:	Esta función devuelve un carácter en minúsculas si el argumento estaba en mayúsculas al principio; de lo contrario, se devuelve el carácter original.
Nombre del archivo:	tolower.c

topper

Función:	Convierta un carácter en un carácter ASCII alfabético en mayúsculas.
Incluir:	tipoc.h
Prototipo:	char toupper(char ch);
Argumentos:	ch
	Personaje a convertir.
Observaciones:	Esta función convierte ch en un carácter ASCII alfabético en mayúsculas siempre que el argumento sea un carácter alfabético en minúsculas válido.
Valor de retorno:	Esta función devuelve un carácter en mayúsculas si el argumento estaba en minúsculas al principio; de lo contrario, se devuelve el carácter original.
Nombre del archivo:	topper.c

Bibliotecas del compilador MPLAB® C18 C

ultoa

Función:	Convierta un entero largo sin signo en una cadena.
Incluir:	stdlib.h
Prototipo:	char * ultoa (valor largo sin signo, char * cadena);
Argumentos:	valor Un entero largo sin signo que se va a convertir. string Puntero a la cadena ASCII que contendrá el resultado.
Observaciones:	Esta función convierte el entero largo sin signo del valor del argumento en una representación de cadena ASCII. La cadena debe ser lo suficientemente larga para contener la representación ASCII, incluido un carácter nulo final. Esta función es una extensión de MPLAB C18 para las bibliotecas requeridas por ANSI.
Valor de retorno:	Puntero a la cadena de resultado .
Nombre del archivo:	ultoa.asm

4.4 FUNCIONES DE MANIPULACIÓN DE MEMORIA Y CADENA

Salvo que se indique lo contrario en las descripciones de las funciones, estas funciones son coherentes con las funciones de la biblioteca C estándar ANSI (1989) del mismo nombre. Se proporcionan las siguientes funciones:

TABLA 4-3: FUNCIONES DE MANIPULACIÓN DE MEMORIA Y CADENA

Función	Descripción
memchr memchrgm	Busque un valor en una región de memoria específica.
memcmp memcmpgmg memcmppgm2ram memcmpram2pgm	Compara el contenido de dos arreglos.
memcpy memcpypgm memcpypgm2ram memcpypgm2pgm	Copie un búfer.
recuerda memmovepgm memmovepgm2ram memmoveram2pgm	Copie un búfer, donde el origen y el destino pueden superponerse.
conjunto de miembros memsetpgm	Inicialice una matriz con un único valor repetido.
strcat strcatpgm strcatpgm2ram strcatram2pgm	Agregue una copia de la cadena de origen al final de la cadena de destino.
strchr strchrpgm	Busque la primera aparición de un valor en una cadena.
strcmp strcmppgm strcmppgm2ram strcmpram2pgm	Compara dos cadenas.
strcpy strcpypgm strcpypgm2ram strcpypgm2pgm	Copie una cadena de la memoria de datos o programa a la memoria de datos.

TABLA 4-3: FUNCIONES DE MANIPULACIÓN DE CADENA Y MEMORIA (CONTINUACIÓN)

strcspn strcspnpgm strcspnpgmram strcspnrampgm	Calcular el número de caracteres consecutivos al principio de una cadena que no están contenidos en un conjunto de caracteres.
estrellándose strlenpgm	Determinar la longitud de una cuerda.
strlwr strlwrgpm	Convierte todos los caracteres en mayúsculas de una cadena a minúsculas.
strncat strncatpgm strncatpgm2ram strncatram2pgm	Agregue un número específico de caracteres desde la cadena de origen hasta el final de la cadena de destino.
strcmp strncmpggm strncmpggm2ram strncmpram2pgm	Compara dos cadenas, hasta un número específico de caracteres.
strncpy strncpypgm strncpypgm2ram strncpyram2pgm	Copie caracteres de la cadena de origen en la cadena de destino, hasta el número de caracteres especificado.
strpbrk strpbrkpgm strpbrkpgmram strpbrkrampgm	Busque en una cadena la primera aparición de un carácter de un conjunto de caracteres.
strrchr strrchrpgm	Busque la última aparición de un carácter especificado en una cadena.
strspn strspnpgm strspnpgmram strspnrampgm	Calcular el número de caracteres consecutivos al principio de una cadena que están contenidos en un conjunto de caracteres.
callestr strstrpgm strstrpgmram strstrrampgm	Busque la primera aparición de una cadena dentro de otra cadena.
strtok strtokpgm strtokpgmram strtokrampgm	Dividir una cadena en subcademas o tokens insertando caracteres nulos en lugar de delimitadores especificados.
strupr struprpgm	Convierte todos los caracteres en minúsculas de una cadena a mayúsculas.

Bibliotecas del compilador MPLAB® C18 C

4.4.1 Descripciones de funciones

memchr

memchrgpm

Función:	Busque la primera aparición de un valor de byte en una región de memoria específica.
Incluir:	cadena.h
Prototipo:	<pre>void * memchr(const void *mem, unsigned char c, size_t n); rom char * memchrgpm(const rom char *mem, const unsigned char c, sizerom_t n);</pre>
Argumentos:	<p>memoria Puntero a una región de memoria.</p> <p>c Valor de byte a buscar.</p> <p>n Número máximo de bytes para buscar.</p>
Observaciones:	<p>Esta función busca hasta n bytes de la región mem para encontrar la primera aparición de c.</p> <p>Esta función difiere de la función especificada por ANSI en que c se define como un parámetro char sin signo en lugar de un parámetro int.</p>
Valor de retorno:	Si c aparece en los primeros n bytes de mem , esta función devuelve un puntero al carácter en mem . De lo contrario, devuelve un puntero nulo.
Nombres de archivos:	memchr.asm mchrgpm.asm

memcmp

memcmpgpm

memcmpgpm2ram

memcmpram2pgm

Función:	Compara el contenido de dos matrices de bytes.
Incluir:	cadena.h
Prototipo:	<pre>char firmado memcmp(const void * buf1, const void * buf2, size_t memsize); char firmado memcmpgpm(const rom void * buf1, const rom void * buf2, sizerom_t memsize); signado char memcmpgpm2ram(const void * buf1, const rom void * buf2, sizeram_t memsize); char firmado memcmpram2pgm (const rom void * buf1, const void * buf2, sizeram_t memsize);</pre>

memcmp memcmpppgm memcmpppgm2ram memcmppram2pgm (Continuación)

Argumentos:	buf1 Puntero a la primera matriz. buf2 Puntero a la segunda matriz. tamaño mem Número de elementos a comparar en matrices.
Observaciones:	Esta función compara el primer número de bytes de memsize en buf1 con el primer número de bytes de memsize en buf2 y devuelve un valor que indica si los búferes son menores, iguales o mayores entre sí.
Valor de retorno:	Devuelve un valor que es: <0 si buf1 es menor que buf2 ==0 si buf1 es igual a buf2 >0 si buf1 es mayor que buf2
Nombres de archivos:	memcmp.asm memcmppp2.asm memcmppp2r.asm memcmppr2p.asm

memcpy memcpypgm memcpypgm2ram memcpypyram2pgm

Función: Copie el contenido del búfer de origen en el búfer de destino.

Incluir: cadena.h

Prototipo:

```
void * memcpy( void * dest, const void * origin,
               size_t memsize );
rom void * memcpypgm( rom void * dest, const rom void * origin,
                      rom void * memsize );
void * memcpyram2ram( void * dest, const rom void * origin,
                      void * sizerom_t memsize );
void * memcpypgm2ram( void * dest, const rom void * origin,
                      void * sizeram_t memsize );
rom void * memcpypyram2pgm( rom void * dest, const void * origin,
                            void * sizeram_t memsize );
```

Argumentos:	dest Puntero a la matriz de destino. origin Puntero a la matriz de origen. memsize Número de bytes de la matriz src para copiar en dest .
Observaciones:	Esta función copia el primer número de bytes de memsize en src a la matriz dest . Si src y dest se superponen, el comportamiento no está definido.

Bibliotecas del compilador MPLAB® C18 C

memcpy**memcpypgm****memcpypgm2ram****memcpypyram2pgm (Continuación)**

Valor de retorno: Esta función devuelve el valor de **dest**.

Nombres de archivos:
memcpy.asm
memcpyp2p.asm
memcpyp2r.asm
memcpyr2p.asm

recuerda**memmovepgm****memmovepgm2ram****memmoveram2pgm**

Función: Copie el contenido del búfer de origen en el búfer de destino, incluso si las regiones se superponen. *cadena.h*

Incluir:

Prototipo:

```
void * memmove( void * dest, const void *  
                src, size_t memsize ); rom  
void * memmovepgm( rom  
                   void * dest, const rom void * sizerom_t memsize );  
void * memmovepgm2ram( void  
                      * dest, const rom void *      origen,  
                               sizeram_t memsize ); rom void  
* memmoveram2pgm(  
                  origen,
```

```
rom void * dest, const  
void * sizeram_t      origen,  
memsize );
```

Argumentos: **dest**

Puntero a la matriz de destino.

*origen*Puntero a la matriz de origen. **memsize** Númerode bytes de la matriz **src** para copiar en **dest**.

Observaciones: Esta función copia el primer número de bytes de **memsize** en **src** a la matriz **dest**. Esta función funciona correctamente incluso si **src** y **dest** se superponen.

Valor de retorno: Esta función devuelve el valor de **dest**.

Nombres de archivos:
memmove.asm
memmovp2p.asm
memmovp2r.asm
memmovr2p.asm

memset**memsetpgm**

Función: Copie el carácter especificado en la matriz de destino.**Incluir:** cadena.h**Prototipo:**
void * memset(void * **dest**, valor de carácter sin firmar , size_t
memsize); rom void *
memsetpgm(rom void * **dest**, valor de carácter sin
firmar , sizerom_t memsize);**Argumentos:** **dest**

Puntero a la matriz de destino.

valorValor de carácter a copiar. **tamaño****mem**Número de bytes de **dest** en los que se copia el **valor**.**Observaciones:** Esta función copia el **valor** del carácter en los primeros bytes de la matriz **dest**. Esta función difiere de la función especificada por ANSI en que el **valor** se define como un carácter sin firmar en lugar de un parámetro int.**Valor de retorno:** Esta función devuelve el valor de **dest**.**Nombre del archivo:**
memset.asm
memsetpgm.asm

strcat**strcatpgm****strcatpgm2ram****strcatram2pgm**

Función: Agregue una copia de la cadena de origen al final de la cadena de destino.**Incluir:** cadena.h**Prototipo:**
char * strcat(char * **destino**,
 const char * **src**);
rom char * strcatpgm(rom char *
 dest, const rom char *
 src);
char * strcatpgm2ram(char *
 dest, const rom
 char * **src**);
rom char * strcatram2pgm(rom char *
 dest, const char * **src**);**Argumentos:** **destino**

Puntero a la matriz de destino.

origen

Puntero a la matriz de origen.

Observaciones: Esta función copia la cadena en **src** al final de la cadena en **dest**.La cadena **src** comienza en el nulo en **dest**. Se agrega un carácter nulo al final de la cadena resultante en **dest**. Si **src** y **dest** se superponen, el comportamiento no está definido.**Valor de retorno:** Esta función devuelve el valor de **dest**.

Bibliotecas del compilador MPLAB® C18 C

strcat

strcatpgm

strcatpgm2ram

strcatram2pgm (Continuación)

Nombres de archivos:	strcat.asm scatp2p.asm scatp2r.asm scatp2p.asm
----------------------	---

strchr

strchrpgm

Función: Busque la primera aparición de un carácter especificado en una cadena.

Incluir: cadena.h

Prototipo: char * strchr(const char * str,
 carácter sin firmar c); rom
char * strchrgm(const rom char * str, unsigned char
 c);

Argumentos: calle

Puntero a una cadena que se va a buscar.

c

Personaje a buscar.

Observaciones: Esta función busca la cadena str para encontrar la primera aparición del carácter c.

Esta función difiere de la función especificada por ANSI en que c se define como un parámetro char sin signo en lugar de un parámetro int.

Valor de retorno: Si c aparece en str, esta función devuelve un puntero al carácter en str. De lo contrario, devuelve un puntero nulo.

Nombres de archivos: strchr.asm
schrgm.asm

strcmp

strcmppgm

strcmppgm2ram

strcmpram2pgm

Función: Compara dos cadenas.

Incluir: cadena.h

Prototipo: firmado char strcmp (const char
 * str1, const char * str2);
 char firmado strcmppgm(

 const rom char * str1, const rom char

 * str2); char firmado strcmppgm2ram(const

 char * str1, const rom char * str2); char firmado
 strcmpram2pgm (const rom char * str1,
 const char * str2);

strcmp
strcmppgm
strcmppgm2ram
strcmpram2pgm

Argumentos:	str1 Puntero a la primera cadena. str2 Puntero a la segunda cadena.
Observaciones:	Esta función compara la cadena en str1 con la cadena en str2 y devuelve un valor que indica si str1 es menor, igual o mayor que str2 .
Valor de retorno:	Devuelve un valor que es: <0 si str1 es menor que str2 ==0 si str1 es igual a str2 >0 si str1 es mayor que str2
Nombre del archivo:	strcmp.asm scmpp2p.asm scmpp2r.asm scmpr2p.asm

strcpy
strcpypgm
strcpypgm2ram
strcpyram2pgm

Función:	Copie la cadena de origen en la cadena de destino.
Incluir:	cadena.h
Prototipo:	char * strcpy(char * dest, const char * src); rom char * strcpypgm(rom char * dest, const rom char * src);char * char * strcpypgm2ram(char * dest, const rom char *src); rom char * strcpyram2pgm(rom char * dest, const char * src);

Argumentos:	dest Puntero a la cadena de destino. origen Puntero a la cadena de origen.
Observaciones:	Esta función copia la cadena en src a dest . Los caracteres en src se copian hasta el carácter nulo final incluido en src . Si src y dest se superponen, el comportamiento no está definido.
Valor de retorno:	Esta función devuelve el valor de dest .
Nombre del archivo:	strcpy.asm scpy2p.asm scpy2r.asm scpyr2p.asm

Bibliotecas del compilador MPLAB® C18 C

strcspn

strcspnpgm

strcspnpgmram

strcspnrampgm

Función: Calcular el número de caracteres consecutivos al principio de una cadena que no están contenidos en un conjunto de caracteres.

Incluir: cadena.h

Prototipo:

```
size_t strcspn(const char * str1,
                const char * str2 );
size_t strcspnpgm(
    const rom char * str1, const rom char
    * str2 );
size_t strcspnpgmram( const rom
    char * str1, const char * str2 );

size_t strcspnrampgm( const char *
    str1, const rom char * str2 );
```

Argumentos: **str1**

Puntero a una cadena que se va a buscar. **str2** Puntero a una cadena que se trata como un conjunto de caracteres.

Observaciones: Esta función determinará el número de caracteres consecutivos desde el principio de **str1** que no están contenidos en **str2**. Por ejemplo: **resultado str1 str2 "holo" "aeiou" 1** "antílope" "aeiou" 0 "antílope" "xyz" 8

Valor de retorno: Esta función devuelve el número de caracteres consecutivos de la

principio de **str1** que no están contenidos en **str2**, como se muestra en los ejemplos anteriores.

Nombres de archivos:

- strcspn.asm
- sclspnpp.asm
- sclspnpr.asm
- sclspnpr.asm

strlen

strlenpgm

Función: Devuelve la longitud de la cadena.

Incluir: cadena.h

Prototipo: size_t strlen(const char * str); size_t strlenpgm(const rom char * str);

Argumentos: **calle**

Puntero a cadena.

Observaciones: Esta función determina la longitud de la cadena, sin incluir el carácter nulo de terminación.

Valor de retorno: Esta función devuelve la longitud de la cadena.

Nombre del archivo:

- strlen.asm
- slenpgm.asm

strlwr**strlwrpgm**

Función:	Convierte todos los caracteres en mayúsculas de una cadena a minúsculas.
Incluir:	cadena.h
Prototipo:	char * strlwr(char * str); rom char * strlwrpgm(rom char * str);
Argumentos:	calle
	Puntero a cadena.
Observaciones:	Esta función convierte todos los caracteres en mayúsculas de str en caracteres en minúsculas. Todos los caracteres que no están en mayúsculas (A a Z) no se ven afectados.
Valor de retorno:	Esta función devuelve el valor de str .
Nombre del archivo:	strlwr.asm slwrpgm.asm

strncat**strncatpgm****strncatpgm2ram****strncatram2pgm**

Función:	Agreege un número específico de caracteres de la cadena de origen a la cadena de destino.
Incluir:	cadena.h
Prototipo:	char * strncat(char * dest, const char * size_t n); rom char * origen, strncatpgm(rom char * dest, const rom char * sizerom_t n); char * strncatpgm2ram(char * destino , origen, const rom char * origen, sizeram_t n); * strncatram2pgm(rom char * dest, carácter constante * origen, sizeram_t n);
Argumentos:	dest Puntero a la matriz de destino. origen Puntero a la matriz de origen. <small>nota</small> Número de caracteres a añadir.
Observaciones:	Esta función agrega exactamente n caracteres desde la cadena en src hasta el final de la cadena en dest . Si se copia un carácter nulo antes de que se hayan copiado n caracteres, los caracteres nulos se agregarán a dest hasta que se hayan agregado exactamente n caracteres. Si src y dest se superponen, el comportamiento no está definido. Si no se encuentra un carácter nulo, no se agrega un carácter nulo.
Valor de retorno:	Esta función devuelve el valor de dest .

Bibliotecas del compilador MPLAB® C18 C

strncat

strncatpgm

strncatpgm2ram

strncatram2pgm (Continuación)

Nombres de archivos:	strncat.asm sncatp2p.asm sncatp2r.asm sncatr2p.asm
----------------------	---

strcmp

strcmpppgm

strcmpppgm2ram

strcmpram2pgm

Función: Compara dos cadenas, hasta un número específico de caracteres.

Incluir: cadena.h

Prototipo: firmado char strcmp (const char * **str1**, const char * **str2**, size_t **n**);

```
char firmado strcmpppgm(  
    const rom char * str1, const rom  
    char * str2 , sizerom_t n);
```

```
char firmado strcmpppgm2ram(  
    const char * str1, const rom  
    char * str2 , sizeram_t n);
```

```
char firmado strcmpram2pgm (const rom  
    char * str1, const char * str2 ,  
    sizeram_t n);
```

Argumentos: **str1**

Puntero a la primera cadena.

str2

Puntero a la segunda cadena.

n Número máximo de caracteres para comparar.

Observaciones: Esta función compara la cadena en **str1** con la cadena en **str2** y devuelve un valor que indica si **str1** es menor, igual o mayor que **str2**. Si se comparan **n** caracteres y no se encuentran diferencias, esta función devolverá un valor que indica que las cadenas son equivalentes.

Valor de retorno: Devuelve un valor basado en el primer carácter que difiere entre **str1** y **str2**. Devuelve: <0 si **str1** es menor que **str2** ==0 si **str1** es igual a **str2** >0 si **str1** es mayor que **str2**

Nombre del archivo:	strcmp.asm sncmp2p.asm sncmp2r.asm sncmp2r.asm
---------------------	---

strncpy strncpypgm strncpypgm2ram strncpyram2pgm

Función: Copie caracteres de la cadena de origen en la cadena de destino, hasta el número de caracteres especificado.

Incluir: cadena.h

Prototipo:

```
char * strncpy( char * dest, const char *
                origen,
                tamaño_n ) ; rom
char * strncpypgm( rom char * dest, const rom
                   char * sizerom_t n ); char
                   *strncpypgm2ram( char * origen,
                                     dest, const rom char *
                                     sizeram_t n ); rom char * strncpyram2pgm( rom char *
                                                       dest, const char * src,
                                                       sizeram_t n );           origen,
```

Argumentos:

- destino**
Puntero a la cadena de destino.
- origen**
Puntero a la cadena de origen.

note
Número máximo de caracteres a copiar.

Observaciones: Esta función copia la cadena en **src** a **dest**. Los caracteres en **src** se copian en **dest** hasta que se copian el carácter nulo final o **n** caracteres. Si se copiaron **n** caracteres y no se encontró ningún carácter nulo, **dest** no terminará en nulo.

Si la copia tiene lugar entre objetos que se superponen, el comportamiento no está definido.

Valor de retorno: Esta función devuelve el valor de **dest**.

Nombre del archivo:

- strncpy.asm
- sncpyp2p.asm
- sncpyp2r.asm
- sncpyr2p.asm

Bibliotecas del compilador MPLAB® C18 C

struprbrk struprbrkpgm struprbrkpgmram struprbrkrampgm

Función:	Busque en una cadena la primera aparición de un carácter de un conjunto de caracteres especificado.
Incluir:	cadena.h
Prototipo:	<pre>char *struprbrk(const char *str1, const char *str2); rom char *struprbrkpgm(const rom char *str1, const rom char *str2); rom char *struprbrkpgmram(const rom char *str1, const char * str2); char *struprbrkrampgm(const char * str1, const rom char *str2);</pre>
Argumentos:	str1 Puntero a una cadena que se va a buscar. str2 Puntero a una cadena que se trata como un conjunto de caracteres.
Observaciones:	Esta función buscará en str1 la primera aparición de un carácter contenido en str2 .
Valor de retorno:	Si se encuentra un carácter en str2 , se devuelve un puntero a ese carácter en str1 . Si no se encuentra ningún carácter de str2 en str1 , se devuelve un puntero nulo.
Nombres de archivos:	struprbrk.asm spbrkpp.asm spbrkpr.asm spbrkrp.asm

strrchr

Función:	Busque la última aparición de un carácter especificado en una cadena.
Incluir:	cadena.h
Prototipo:	<code>char *strrchr(const char *str, const char c);</code>
Argumentos:	str Puntero a una cadena que se va a buscar. c Personaje a buscar.
Observaciones:	Esta función busca la cadena str , incluido el carácter nulo final, para encontrar la última aparición del carácter c . Esta función difiere de la función especificada por ANSI en que c se define como un parámetro char sin signo en lugar de un parámetro int.
Valor de retorno:	Si c aparece en str , esta función devuelve un puntero al carácter en str . De lo contrario, devuelve un puntero nulo.
Nombres de archivos:	strrchr.asm

**strspn
strspnpgm
strspnpgmram
strspnrampgm**

Función: Calcular el número de caracteres consecutivos al principio de una cadena que están contenidos en un conjunto de caracteres.

Incluir: cadena.h

Prototipo: size_t strspn(const char * str1, const char * str2);

```
sizerom_t strspnpgm( const rom
                      char * str1, const rom char * str2 );
```

```
tamañoram_t strspnpgmram(
               const rom char * str1, const char *
               str2 );
```

```
tamañoram_t strspnrampgm(
               const char * str1, const rom
               char * str2 );
```

Argumentos: str1

Puntero a una cadena que se va a buscar. str2 Puntero a una cadena que se trata como un conjunto de caracteres.

Observaciones: Esta función determinará el número de caracteres consecutivos desde el principio de str1 que están contenidos en str2. Por ejemplo: **resultado str1 "plátano" 2 "plátano"**
str2 6 "plátano" 0
"ab"
"abn"
"un"

Valor de retorno: Esta función devuelve el número de caracteres consecutivos de la principio de str1 que están contenidos en str2, como se muestra en los ejemplos anteriores.

Nombres de archivos:
strspn.asm
sspripp.asm
sspnpr.asm
sspnrp.asm

Bibliotecas del compilador MPLAB® C18 C

strstr

strstrpgm

strstrpgmram

strstrrampgm

Función: Busque la primera aparición de una cadena dentro de otra cadena.

Incluir: cadena.h

Prototipo:

```
char * strstr( const char * str,
                const char * substr );
rom char * strstrpgm( const rom
                      char * str, const rom char * substr );

char de rom * strstrpgmram(
    const rom char * str, const char *
    substr );
char * strstrrampgm( const char *
                     str, const rom char * substr );
```

Argumentos: **calle**

Puntero a una cadena que se va a buscar.

substr Puntero a un patrón de cadena que
buscar.

Observaciones: Esta función encontrará la primera aparición de la cadena **substr** (excluyendo
el terminador nulo) dentro de la cadena **str**.

Valor de retorno: Si se encuentra la cadena , se devolverá un puntero a esa cadena en **str** .
De lo contrario, se devuelve un puntero nulo.

Nombres de archivos:
strstr.asm
sstrpp.asm
sstrpr.asm
sstrrp.asm

strtok

strtokpgm

strtokpgmram

strtokrampgm

Función: Dividir una cadena en subcadenas o tokens insertando caracteres nulos en
lugar de delimitadores especificados.

Incluir: cadena.h

Prototipo:

```
char * strtok(char * str,
              const char * delim );
rom char * strtokpgm( rom char *
                      str, const rom char *
                      delim ); char * strtokpgmram( char * str,
                                         const rom char * delim );
```

```
rom char * strtokrampgm(
    rom char * str, const
    char * delim );
```

Argumentos: **calle**

Puntero a una cadena que se va a buscar.

delimitar

Puntero a un conjunto de caracteres que indican el final de un token.

strtok**strtokpgm****strtokpgmram****strtokrampgm (Continuación)**

Observaciones: Esta función se puede usar para dividir una cadena en subcadenas al reemplazar los caracteres especificados con caracteres nulos. La primera vez que se invoca esta función en una cadena en particular, esa cadena debe pasarse en **str**. Después de la primera vez, esta función puede continuar analizando la cadena desde el último delimitador invocándolo con un valor nulo pasado en **str**.

Cuando se invoca strtok con un parámetro no nulo para **str**, comienza a buscar **str** desde el principio. Omite todos los caracteres principales que aparecen en la cadena **delim**, luego omite todos los caracteres que no aparecen en **delim**, luego establece el siguiente carácter en nulo.

Cuando se invoca strtok con un parámetro nulo para **str**, busca la cadena que se examinó más recientemente, comenzando con el carácter posterior al que se estableció en nulo durante la llamada anterior. Omite todos los caracteres que no aparecen en **delim**, luego establece el siguiente carácter en nulo.

Si strtok encuentra el final de la cadena antes de encontrar un delimitador, no modifica la cadena.

El conjunto de caracteres que se pasa en **delim** no necesita ser el mismo para cada llamada a strtok.

Valor de retorno: Si se encontró un delimitador, esta función devuelve un puntero en **str** al primer carácter buscado que no aparecía en el conjunto de caracteres **delim**. Este carácter representa el primer carácter de un token creado por la llamada.

Si no se encuentra ningún delimitador antes del carácter nulo de terminación, la función devuelve un puntero nulo.

Nombres de archivos:
strtok.asm
stokpgm.asm
stokpr.asm
stokrp.asm

strupr
struprpgm

Función: Convierte todos los caracteres en minúsculas de una cadena a mayúsculas.

Incluir: `cadena.h`

Prototipo: `char * strupr(char * str); rom char *`
`struprpgm(rom char * str);`

Argumentos: **calle**

Puntero a cadena.

Observaciones: Esta función convierte todos los caracteres en minúsculas de **str** en caracteres en mayúsculas. Todos los caracteres que no están en minúsculas (de la a a la z) no se ven afectados.

Valor de retorno: Esta función devuelve el valor de **str**.

Nombre del archivo:
strupr.asm
suprpgm.asm

Bibliotecas del compilador MPLAB® C18 C

4.5 FUNCIONES DE RETARDO

Las funciones de retraso ejecutan código para un número específico de ciclos de instrucción del procesador. Para retrasos basados en el tiempo, se debe tener en cuenta la frecuencia de operación del procesador. Se proporcionan las siguientes rutinas:

TABLA 4-4: FUNCIONES DE RETARDO

Función	Descripción
Retraso1TCY	Retrasar un ciclo de instrucción.
Retraso10TCYx	Retardo en múltiplos de 10 ciclos de instrucción.
Retraso100TCYx	Retardo en múltiplos de 100 ciclos de instrucción.
Retraso1KTCYx	Retardo en múltiplos de 1000 ciclos de instrucción.
Retraso10KTCYx	Retardo en múltiplos de 10.000 ciclos de instrucción.

4.5.1 Descripciones de funciones

Retraso1TCY

Función:	Retardo 1 ciclo de instrucción (TCY).
Incluir:	retrasos.h
Prototipo:	vacío Delay1TCY (vacío); Esta función
Observaciones:	es en realidad un #define para la instrucción NOP. Cuando se encuentra en el código fuente, el compilador simplemente inserta un NOP.
Nombre del archivo:	#define en retrasos.h

Retraso10TCYx

Función:	Retardo en múltiplos de 10 ciclos de instrucción (TCY).
Incluir:	retrasos.h
Prototipo:	void Delay10TCYx (unidad de caracteres sin firmar);
Argumentos:	unidad
	El valor de la unidad puede ser cualquier valor de 8 bits. Un valor en el rango [1,255] retrasará (unidad * 10) ciclos. Un valor de 0 provoca un retraso de 2.560 ciclos.
Observaciones:	Esta función crea un retraso en múltiplos de 10 ciclos de instrucción.
Nombre del archivo:	d10tcyx.asm

Retraso100TCYx

Función:	Retardo en múltiplos de 100 ciclos de instrucción (TCY).
Incluir:	retrasos.h
Prototipo:	void Delay100TCYx (unidad de caracteres sin firmar);
Argumentos:	unidad
	El valor de la unidad puede ser cualquier valor de 8 bits. Un valor en el rango [1,255] retrasará (unidad * 100) ciclos. Un valor de 0 provoca un retraso de 25.600 ciclos.

Delay100TCYx (continuación)

Observaciones: Esta función crea un retraso en múltiplos de 100 ciclos de instrucción. Esta función utiliza la variable asignada globalmente, DelayCounter1. Si esta función se usa tanto en la interrupción como en el código principal, la variable DelayCounter1 debe guardarse y restaurarse en el controlador de interrupciones. Consulte la cláusula save= de las directivas #pragma interrupt o #pragma interruptlow para obtener más información. Tenga en cuenta que otras funciones de retardo también utilizan la variable DelayCounter1 asignada globalmente.

Nombre del archivo: d100tcyx.asm

Retraso1KTCYx

Función: Retardo en múltiplos de 1000 ciclos de instrucción (TCY).
Incluir: retrazos.h
Prototipo: void Delay1KTCYx (unidad de caracteres sin firmar);
Argumentos: **unidad**
 El valor de la **unidad** puede ser cualquier valor de 8 bits. Un valor en el rango [1,255] retrasará (**unidad** * 1000) ciclos. Un valor de 0 provoca un retraso de 256.000 ciclos.

Observaciones: Esta función crea un retraso en múltiplos de 1000 ciclos de instrucción. Esta función utiliza las variables asignadas globalmente, DelayCounter1 y DelayCounter2. Si esta función se usa tanto en el código de interrupción como en el de la línea principal, estas variables, DelayCounter1 y DelayCounter2, deben guardarse y restaurarse en el controlador de interrupción. Consulte la cláusula save= de las directivas #pragma interrupt y #pragma interruptlow para obtener más información. Tenga en cuenta que otras funciones de retardo también utilizan la variable DelayCounter1 asignada globalmente.

Nombre del archivo: d1ktcyx.asm

Retraso10KTCYx

Función: Retardo en múltiplos de 10.000 ciclos de instrucción (TCY).
Incluir: retrazos.h
Prototipo: void Delay10KTCYx (unidad de caracteres sin firmar);
Argumentos: **unidad**
 El valor de la **unidad** puede ser cualquier valor de 8 bits. Un valor en el rango [1,255] retrasará (**unidad** * 10000) ciclos. Un valor de 0 provoca un retraso de 2.560.000 ciclos.

Observaciones: Esta función crea un retraso en múltiplos de 10.000 ciclos de instrucción. Esta función utiliza la variable asignada globalmente, DelayCounter1. Si esta función se usa tanto en la interrupción como en el código principal, la variable DelayCounter1 debe guardarse y restaurarse en el controlador de interrupciones. Consulte la cláusula save= de las directivas #pragma interrupt o #pragma interruptlow para obtener más información. Tenga en cuenta que otras funciones de retardo también utilizan la variable DelayCounter1 asignada globalmente.

Nombre del archivo: d10ktcyx.asm

Bibliotecas del compilador MPLAB® C18 C

4.6 FUNCIONES DE RESTABLECIMIENTO

Las funciones de reinicio se pueden usar para ayudar a determinar la fuente de un evento de reinicio o activación y para reconfigurar el estado del procesador después de un reinicio. Se proporcionan las siguientes rutinas:

TABLA 4-5: FUNCIONES DE RESTABLECIMIENTO

Función	Descripción
esBOR	Determine si la causa de un restablecimiento fue el circuito de restablecimiento de Brown-out.
esLVD	Determine si la causa de un restablecimiento fue una condición de detección de bajo voltaje.
esMCLR	Determine si la causa de un restablecimiento fue el pin MCLR.
esPOR	Detectar una condición de reinicio de encendido.
esWDTTO	Determine si la causa de un restablecimiento fue un tiempo de espera del temporizador de vigilancia.
esWDTWU	Determine si la causa de la activación fue el temporizador de vigilancia.
esWU	Detecta si el microcontrolador acaba de despertarse de la suspensión desde el pin MCLR o una interrupción.
EstadoReiniciar	Establezca los bits POR y BOR.

Nota: Si está utilizando Brown-out Reset (BOR) o Watchdog Timer (WDT), debe definir las macros de habilitación (#define BOR_ENABLED y #define WDT_ENABLED, respectivamente) en el archivo de encabezado reset.h y recomilar el código fuente .

4.6.1 Descripciones de funciones

esBOR

Función:	Determine si la causa de un restablecimiento fue el circuito de restablecimiento de Brown-out.
Incluir:	restablecer.h
Prototipo:	char esBOR(vacío);
Observaciones:	Esta función detecta si el microcontrolador se reinició debido al circuito Brown-out Reset. Esta condición se indica mediante los siguientes bits de estado: POR = 1 BOR = 0

Valor de retorno: 1 si el reinicio se debió al circuito de reinicio Brown-out 0 de lo contrario

Nombre del archivo: isbor.c

esLVD

Función:	Determine si la causa de un restablecimiento fue una condición de detección de bajo voltaje.
Incluir:	restablecer.h
Prototipo:	char esLVD(vacío);
Observaciones:	Esta función detecta si el voltaje del dispositivo se ha vuelto más bajo que el valor especificado en el registro LVDCON (bits LVDL3:LVDL0).
Valor de retorno:	1 si un restablecimiento se debió a LVD durante el funcionamiento normal 0 de lo contrario
Nombre del archivo:	islvd.c

esMCLR

Función:	Determine si la causa de un restablecimiento fue el pin MCLR.
Incluir:	restablecer.h
Prototipo:	char esMCLR(vacio);
Observaciones:	Esta función detecta si el microcontrolador se restableció a través del pin MCLR durante el funcionamiento normal. Esta situación se indica mediante los siguientes bits de estado: POR = 1 Si Brown-out está habilitado, BOR = 1 Si WDT está habilitado, TO = 1 PD = 1

Valor de retorno:	1 si el restablecimiento se debió a MCLR durante el funcionamiento normal 0 de lo contrario
--------------------------	--

Nombre del archivo: ismclr.c

esPOR

Función:	Detectar una condición de reinicio de encendido.
Incluir:	restablecer.h
Prototipo:	char isPOR(vacío);
Observaciones:	Esta función detecta si el microcontrolador acaba de dejar un Power-on Reset. Esta condición se indica mediante los siguientes bits de estado: POR = 0 BOR = 0 TO = 1 PD = 1

Esta condición también puede ocurrir para MCLR durante la operación normal y cuando se ejecuta la instrucción CLRWDT.

Después de llamar a isPOR, se debe llamar a StatusReset para establecer los bits POR y BOR.

Valor de retorno:	1 si el dispositivo acaba de dejar un reinicio de encendido 0 de lo contrario
--------------------------	--

Nombre del archivo: ispor.c

esWDTTO

Función:	Determine si la causa de un restablecimiento fue un tiempo de espera del temporizador de vigilancia (WDT).
Incluir:	restablecer.h
Prototipo:	char esWDTTO(vacio);
Observaciones:	Esta función detecta si el microcontrolador se reinició debido al WDT durante el funcionamiento normal. Esta condición se indica mediante los siguientes bits de estado: POR = 1 BOR = 1 TO = 0 PD = 1

Valor de retorno:	1 si el reinicio se debió al WDT durante el funcionamiento normal 0 de lo contrario
--------------------------	---

Nombre del archivo: iswdtto.c

Bibliotecas del compilador MPLAB® C18 C

esWDTWU

Función:	Determine si la causa de una activación fue el temporizador de vigilancia (WDT).
Incluir:	restablecer.h
Prototipo:	char esWDTWU(vacío);
Observaciones:	Esta función detecta si el WDT sacó al microcontrolador de la suspensión. Esta condición se indica mediante los siguientes bits de estado: POR = 1 BOR = 1 TO = 0 PD = 0 _____

Valor de retorno:	1 si el WDT sacó el dispositivo de la suspensión 0 de lo contrario
Nombre del archivo:	iswdtwu.c

esWU

Función:	Detecta si el microcontrolador acaba de despertarse de la suspensión a través del pin MCLR o la interrupción.
Incluir:	restablecer.h
Prototipo:	char esWU(vacío);
Observaciones:	Esta función detecta si el pin MCLR o una interrupción sacaron al microcontrolador de la suspensión. Esta condición se indica mediante los siguientes bits de estado: POR = 1 BOR = 1 TO = 1 PD = 0 _____

Valor de retorno:	1 si el pin MCLR o una interrupción 0 sacaron el dispositivo de la suspensión; de lo contrario
Nombre del archivo:	iswu.c

EstadoReiniciar

Función:	Establezca los bits PÖR y BOR en el registro CPUSTA.
Incluir:	restablecer.h
Prototipo:	void StatusReset(void);
Observaciones:	Esta función establece los bits POR y BOR en el registro CPUSTA. Estos bits deben establecerse en el software después de que se haya producido un reinicio de encendido.
Nombre del archivo:	inicio.c

4.7 FUNCIONES DE SALIDA DE CARACTERES

Las funciones de salida de caracteres proporcionan una familia central de funciones para procesar la salida a periféricos, búferes de memoria y otros consumidores de datos de caracteres.

Cuando se procesa una llamada a fprintf, printf, sprintf, vfprintf, vprintf o vsprintf, MPLAB C18 siempre procesará la parte de longitud variable de la lista de argumentos con las promociones de enteros habilitadas (consulte la sección "Promociones de enteros" de la Guía del usuario del compilador MPLAB® C18 C). (DS51288) para obtener más información). Esto permite que la biblioteca estándar interactúe con el compilador de forma limpia y con un comportamiento coherente para el formato de la salida, como normalmente se esperaría de esas funciones.

4.7.1 Flujos de salida

La salida se basa en el uso de un flujo de destino. Un flujo puede ser un periférico, un búfer de memoria o cualquier otro consumidor de datos y se indica mediante un puntero a un objeto de tipo ARCHIVO . MPLAB C18 define dos flujos en la biblioteca estándar: salida _H_USER a través de la función de salida definida por el usuario _user_putc.

Salida _H_USART a través de la función de salida de la biblioteca _usart_putc.

La versión actual de la biblioteca solo admite estos dos flujos de salida. Ambos flujos siempre se consideran abiertos y no requieren el uso de funciones como fopen, fclose, etc.

Las variables globales stdout y stderr están definidas por la biblioteca y tienen un valor predeterminado de _H_USART. Para cambiar el destino a _H_USER, asigne ese valor a la variable. Por ejemplo, para cambiar la salida estándar para usar la función de salida definida por el usuario:

```
salida estándar = _H_USUARIO;
```

TABLA 4-6: FUNCIONES DE SALIDA DE CARACTERES

Función	Descripción
fprintf	Salida de cadena formateada a una secuencia.
entradas	Cadena de salida a una secuencia.
imprimir	Salida de cadena formateada a stdout.
poner	Salida de caracteres a una secuencia
pone	Salida de cadena a stdout.
correr	Salida de cadena formateada a un búfer de memoria de datos.
vfprintf	Salida de cadena formateada a una secuencia con los argumentos para procesar la cadena de formato proporcionada a través de la función stdarg.
vprintf	Salida de cadena formateada a stdout con los argumentos para procesar la cadena de formato proporcionada a través de la función stdarg.
vsprintf	Salida de cadena formateada a un búfer de memoria de datos con los argumentos para procesar la cadena de formato proporcionada a través de la función stdarg.
_usart_putc	Salida de un solo carácter al USART (USART1 para dispositivos que tienen más de un USART).
_usuario_putc	Salida de un solo carácter de una manera definida por la aplicación.

Bibliotecas del compilador MPLAB® C18 C

4.7.2 Descripciones de funciones

fprintf

Función:	Salida de cadena formateada a una secuencia.
Incluir:	stdio.h
Prototipo:	int fprintf (ARCHIVO *f, const rom char *fmt, ...);
Observaciones:	<p>La función fprintf da formato a la salida, pasando los caracteres al flujo especificado a través de la función putc. La cadena de formato se procesa un carácter a la vez y los caracteres se emiten tal como aparecen en la cadena de formato, a excepción de los especificadores de formato. Un especificador de formato se indica en la cadena de formato mediante un signo de porcentaje, %; a continuación, un especificador de formato bien formado tiene los siguientes componentes.¹ Excepto por la operación de conversión, todos los especificadores de formato son opcionales:</p> <ol style="list-style-type: none">1. Caracteres de bandera (el orden no importa), donde un carácter de bandera es uno de #, -, +, 0 o espacio.2. Un ancho de campo, que es un valor constante entero decimal un asterisco, *.3. Una precisión de campo, que es un punto (.), opcionalmente seguido de un entero decimal o un asterisco, *.4. Una especificación de tamaño, que es uno de los especificadores h, H, hh, j, z, Z, t, T o l.5. Una operación de conversión, que es una de c, b, B, d, i, n, o, p, P, s, S, u, x, X o %.

¹No todos los componentes son válidos para todas las operaciones de conversión. Los detalles se proporcionan en las descripciones de los operadores de conversión.

fprintf (continuación)

Caracteres de la

bandera # Se presentará la forma alternativa del resultado. Para la conversión de o, la forma alternativa es como si se aumentara la precisión de tal manera que el primer dígito del resultado se ve obligado a ser cero. Para la versión x con, un resultado distinto de cero tendrá un prefijo 0x agregado. Para la conversión X, un resultado distinto de cero tendrá un prefijo 0X agregado. Para la conversión b, un resultado distinto de cero tendrá un prefijo 0b agregado.

Para la conversión B, un resultado distinto de cero tendrá un prefijo 0B agregado. Para otras conversiones, la bandera se ignora.

- El resultado se justificará a la izquierda. Si no se especifica esta bandera, el resultado estará justificado a la derecha.

+ Para una conversión con signo, el resultado siempre comenzará con un signo + o -. De forma predeterminada, solo se agrega un carácter de signo al resultado si el resultado es negativo.

Para otras conversiones, la bandera se ignora. espacio Para una conversión firmada, si el resultado no es negativo o no tiene caracteres, se antepondrá un espacio al resultado. Si se especifican los indicadores de espacio y +, se ignorará el indicador de espacio. Para otras conversiones, la bandera se ignora.

0 Para las conversiones de enteros (d, i, o, u, b, B, x, X), los ceros iniciales se anteponen al resultado (después de cualquier signo y/o indicador de base) de modo que el resultado llene el ancho del campo. No se realiza relleno de espacio. Si también se especifica el indicador -, se ignorará el indicador 0. Si se especifica una precisión, se ignorará el indicador 0. Para otras conversiones, la bandera se ignora.

Ancho de campo

El ancho del campo especifica el número mínimo de caracteres para el valor convertido. Si el valor convertido es más corto que el ancho del campo, el valor se rellena para que el número de caracteres sea igual al ancho del campo. De forma predeterminada, los espacios iniciales se utilizan para el relleno; los caracteres de bandera se utilizan para modificar el carácter de relleno y la justificación del valor.

Si el ancho del campo es un carácter de asterisco, *, se lee un argumento int para especificar el ancho del campo. Si el valor es negativo, es como si se especificara el indicador -, seguido de un ancho de campo positivo.

Precisión de campo

La precisión de campo especifica el número mínimo de dígitos que estarán presentes en el valor convertido para la conversión ad, i, o, u, b, B, x o X, o el número máximo de caracteres en el valor convertido para una s conversión.

Si el ancho del campo es un carácter de asterisco, *, se lee un argumento int para especificar el ancho del campo. Si el valor es negativo, es como si la precisión no estuviera especificada.

Para los operadores de conversión d, i, o, u, b, B, x o X, la precisión predeterminada es 1. Para todos los demás operadores de conversión, el comportamiento cuando no se especifica la precisión se describe a continuación.

Bibliotecas del compilador MPLAB® C18 C

fprintf (continuación)

Especificaciones de

tamaño El carácter de especificación de tamaño se aplica a los especificadores de conversión de enteros, d, i, o, u, b, B, y los específicadores presentes para el operador de conversión, se ignora. hh Para los especificadores de conversión de enteros, el argumento a convertir es un argumento char con signo o un char sin signo.²

Para un especificador de conversión n, el especificador denota un puntero a un argumento char con signo.

h Para los especificadores de conversión de enteros, el argumento a convertir es un int corto o un int corto sin signo. Para un especificador de conversión n, el especificador denota un puntero a un argumento int corto. Como un int simple tiene el mismo tamaño que un int corto para MPLAB C18, esta opción no tiene ningún efecto real y está presente solo por motivos de compatibilidad. Para los especificadores de conversión de puntero, el argumento a convertir es un puntero de 16 bits.

H Para los especificadores de conversión de enteros, el argumento a convertir es un int largo corto o un int largo corto sin signo. Para un especificador de versión n conversor, el especificador denota un puntero a un argumento int corto y largo. Para los especificadores de conversión de puntero, el argumento a convertir es un puntero de 24 bits.³ Por ejemplo, cuando se genera un far rom char *, se debe usar el especificador de tamaño H (%HS). j Para los especificadores de conversión de enteros, el argumento a convertir es un argumento intmax_t o uintmax_t. Para un especificador de conversión n, el especificador denota un puntero a un argumento intmax_t. Para MPLAB C18, esto es equivalente al especificador de tamaño l. l Para los especificadores de conversión de enteros, el argumento que se va a convertir es un entero largo o un entero largo sin signo. Para un especificador de conversión n, el especificador denota un puntero a un argumento int largo. Para los especificadores de conversión de puntero, se ignora el especificador de tamaño. t Para los especificadores de conversión de enteros, el argumento a convertir es un argumento ptrdiff_t. Para un especificador de conversión n, el especificador denota un puntero a un tipo de entero con signo correspondiente al argumento ptrdiff_t. Para MPLAB C18, esto es equivalente al especificador de tamaño h.

T Para los especificadores de conversión de enteros, el argumento a convertir es un argumento ptrdifffrom_t. Para un especificador de conversión n, el especificador denota un puntero a un tipo de entero con signo correspondiente al argumento ptrdifffrom_t. Para MPLAB C18, esto es equivalente al especificador de tamaño H.⁴ z Para los especificadores de conversión de enteros, el argumento a convertir es un argumento size_t. Para un especificador de conversión n, el especificador denota un puntero a un tipo de entero con signo correspondiente al argumento size_t. Para MPLAB C18, esto es equivalente al especificador de tamaño h.

Z Para los especificadores de conversión de enteros, el argumento a convertir es un argumento sizerom_t. Para un especificador de conversión n, el especificador denota un puntero a un tipo de entero con signo correspondiente al argumento sizerom_t. Para MPLAB C18, esto es equivalente al especificador de tamaño H.⁵

²Tenga en cuenta que las promociones de enteros seguirán aplicándose cuando se pase el argumento. Este especificador hace que el argumento vuelva a tener un tamaño de 8 bits antes de que se use el valor.

³El especificador de tamaño H es una extensión específica de MPLAB C18 para ANSI C.

⁴El especificador de tamaño T es una extensión específica de MPLAB C18 para ANSI C.

⁵El especificador de tamaño Z es una extensión específica de MPLAB C18 para ANSI C.

fprintf (continuación)

Operadores de

conversión c El argumento int se convierte en un valor de carácter sin signo y se escribe el carácter representado por ese valor.

d, i El argumento int se formatea como decimal con signo con la

precisión que indica el número mínimo de dígitos que se escribirán.

Si el valor convertido tiene menos dígitos, se le anteponen ceros.

Si el valor convertido es cero y la precisión es cero, no se escribirán caracteres.

o El argumento int sin signo se convierte a octal sin signo con la precisión que indica el número mínimo de dígitos que se escribirán. Si el valor convertido tiene menos dígitos, se le anteponen ceros a la izquierda. Si el valor convertido es cero y la precisión es cero, no se escribirán caracteres.

u El argumento int sin signo se formatea como decimal sin signo con la precisión que indica el número mínimo de dígitos que se escribirán. Si el valor convertido tiene menos dígitos, se le anteponen ceros. Si el valor convertido es cero y la precisión es cero, no se escribirán caracteres.

b El argumento int sin signo se formatea como binario sin signo con la precisión que indica el número mínimo de dígitos que se escribirán. Si el valor convertido tiene menos dígitos, se le anteponen ceros. Si el valor convertido es cero y la precisión es cero, no se escribirán caracteres.⁶

B El argumento int sin signo se formatea como binario sin signo con la precisión que indica el número mínimo de dígitos que se escribirán. Si el valor convertido tiene menos dígitos, se le anteponen ceros. Si el valor convertido es cero y la precisión es cero, no se escribirán caracteres.⁷

x El argumento int sin signo se formatea como hexadecimal sin signo con la precisión que indica el número mínimo de dígitos que se escribirán. Los caracteres abcdef se utilizan para la representación de los números decimales del 10 al 15. Si el valor convertido tiene menos dígitos, se le anteponen ceros. Si el valor convertido es cero y la precisión es cero, no se escribirán caracteres.

X El argumento int sin signo se formatea como hexadecimal sin signo con la precisión que indica el número mínimo de dígitos que se escribirán. Los caracteres ABCDEF se utilizan para la representación de los números decimales del 10 al 15. Si el valor convertido tiene menos dígitos, se le anteponen ceros. Si el valor convertido es cero y la precisión es cero, no se escribirán caracteres. **s** Los caracteres de la matriz de memoria de datos del argumento char se escriben hasta que se ve un carácter de terminación '\0' (el carácter '\0' no se escribe) o el número de caracteres escritos es igual a la precisión especificada. Si se especifica que la precisión es mayor que el tamaño de la matriz o no se especifica, la matriz debe contener un carácter de terminación '\0'.

S Los caracteres de la matriz de la memoria del programa del argumento char se escriben hasta que se ve un carácter de terminación '\0' (el carácter '\0' no se escribe) o el número de caracteres escritos es igual a la precisión especificada. Si se especifica que la precisión es mayor que el tamaño de la matriz o no se especifica, la matriz debe contener un carácter de terminación '\0'. **.%HS**).

⁶El operador de conversión b es una extensión específica de MPLAB C18 para ANSI C.

⁷El operador de conversión B es una extensión específica de MPLAB C18 para ANSI C.

8El operador de conversión S es una extensión específica de MPLAB C18 para ANSI C.

Bibliotecas del compilador MPLAB® C18 C

fprintf (continuación)

p El puntero al argumento void (datos o memoria de programa) está configurado se convierte a un tipo de entero sin signo de tamaño equivalente y ese valor se procesa como si se hubiera especificado el operador de conversión x. Si el especificador de tamaño H está presente, el puntero es un puntero de 24 bits; de lo contrario, es un puntero de 16 bits.

P El puntero al argumento void (datos o memoria de programa) está configurado se convierte a un tipo de entero sin signo de tamaño equivalente y ese valor se procesa como si se hubiera especificado el operador de conversión X. Si el especificador de tamaño H está presente, el puntero es un puntero de 24 bits; de lo contrario, es un puntero de 16 bits.⁹

n El número de caracteres escritos hasta el momento se almacenará en la ubicación a la que hace referencia el argumento, que es un puntero a un tipo entero en la memoria de datos. El tamaño del tipo entero está determinado por el especificador de tamaño presente para la conversión, o un entero simple de 16 bits si no hay ningún especificador de tamaño presente.

% Se escribe un carácter % literal. La especificación de conversión será %% solamente, no pueden estar presentes banderas u otros especificadores.

Si un especificador de conversión no es válido (p. ej., un carácter indicador está presente para el especificador de conversión %%), el comportamiento no está definido. fprintf devuelve EOF si se produce un error; de lo contrario, devuelve el número de caracteres de salida.

Valor de retorno:

Nombre del archivo: fprintf.c

Ejemplo de código:

```
#incluir <stdio.h>
vacío principal (vacío) {

    far rom char * S = "¡Hola, mundo!"; entero n = 0x1234;
    fprintf (_H_USART, "salida de prueba a USART\n"); fprintf
    (_H_USER, "salida de prueba a la aplicación"

        "función definida\n"); fprintf (stdout,
    "salida hexadecimal: %#x", n); fprintf (stderr, "%HS\n", S);

}
```

⁹El operador de conversión P es una extensión específica de MPLAB C18 para ANSI C.

entradas

Función: Cadena de salida a una secuencia.

Incluir: stdio.h

Prototipo: int fputs (const rom char *s, FILE *f);

Observaciones: fputs genera una cadena terminada en nulo en el flujo de salida especificado, un carácter a la vez a través de putc. Se añade un carácter de nueva línea a la salida. El nulo de terminación no se emite.

Valor de retorno: fputs devuelve EOF si se produce un error; de lo contrario, devuelve un valor no negativo.

Nombre del archivo: fputs.c

imprimir

Función: Salida de cadena formateada a stdout.
Incluir: stdio.h
Prototipo: int printf (const rom char *fmt, ...);
Observaciones: La función printf da formato a la salida, pasando los caracteres a stdout a través de la función putc. La cadena de formato se procesa como se describe para la función fprintf. printf devuelve EOF si se produce un error; de lo contrario, devuelve el número de caracteres de salida.

Nombre del archivo: imprimirf.c

Ejemplo de código: #incluir <stdio.h>
vacío principal (vacío) {

 /* se generará a través de stdout (_H_USART por defecto) */ printf ("¡Hola,
mundo!\n");
}

poner

Función: Salida de caracteres a una secuencia.
Incluir: stdio.h
Prototipo: int putc (char c, ARCHIVO *f);
Observaciones: putc genera un solo carácter en el flujo de salida especificado.
Valor de retorno: putc devuelve EOF si se produce un error; de lo contrario, devuelve el carácter que se emitió.
Nombre del archivo: putc.c

pone

Función: Salida de cadena a stdout.
Incluir: stdio.h
Prototipo: int pone (const rom char *s);
Observaciones: puts genera una cadena terminada en nulo para stdout un carácter a la vez a través de putc. Se añade un carácter de nueva línea a la salida. El nulo de terminación no se emite.
Valor de retorno: puts devuelve EOF si se produce un error; de lo contrario, devuelve un valor no negativo.
Nombre del archivo: pone.c
Ejemplo de código: #incluir <stdio.h>
vacío principal (vacío) {

 pone ("mensaje de prueba");
}

Bibliotecas del compilador MPLAB® C18 C

correr

Función:	Salida de cadena formateada a un búfer de memoria de datos.
Incluir:	stdio.h
Prototipo:	int sprintf (char *buf, const rom char *fmt, ...);
Observaciones:	La función sprintf da formato a la salida, almacenando los caracteres en el búfer de memoria de datos de destino, buf. La cadena de formato, fmt, se procesa como se describe para la función fprintf. sprintf devuelve EOF si se produce un error; de lo contrario, se devuelve el número de caracteres de salida.
Valor de retorno:	produce un error; de lo contrario, se devuelve el número de caracteres de salida.
Nombre del archivo:	sprintf.c
Ejemplo de código:	#include <stdio.h> vacío principal (vacío) { int i = 0xA12; charbuf[20]; sprintf (buf, "%#010x", yo); /* buf contendrá la cadena "0x00000a12" }

vfprintf

Función:	Salida de cadena formateada a una secuencia con los argumentos para procesar la cadena de formato proporcionada a través de la función stdarg.
Incluir:	stdio.h
Prototipo:	int vfprintf (ARCHIVO *f, const rom char *fmt, va_list ap);
Observaciones:	La función vfprintf da formato a la salida, pasando los caracteres al flujo de salida especificado, f, a través de la función puts. La cadena de formato, fmt, se procesa como se describe para la función fprintf excepto que los argumentos consumidos al procesar la cadena de formato se recuperan a través de la función de argumento de longitud variable stdarg. vfprintf devuelve EOF si se produce un error; de lo contrario, se devuelve el número de caracteres de salida. vfprintf.c
Valor de retorno:	error; de lo contrario, se devuelve el número de caracteres de salida. vfprintf.c

Nombre del archivo:

vprintf

Función:	Salida de cadena formateada a stdout con los argumentos para procesar la cadena de formato proporcionada a través de la función stdarg.
Incluir:	stdio.h
Prototipo:	int vprintf (const rom char *fmt, va_list ap); La función vprintf da formato
Observaciones:	a la salida, pasando los caracteres a stdout a través de la función puts. La cadena de formato, fmt, se procesa como se describe para la función fprintf excepto que los argumentos consumidos al procesar la cadena de formato se recuperan a través de la función de argumento de longitud variable stdarg. vprintf devuelve EOF si se produce un error; de lo contrario, se devuelve el número de caracteres de salida.
Valor de retorno:	caracteres de salida.

Nombre del archivo: vprintf.c

vsprintf

Función:	Salida de cadena formateada a un búfer de memoria de datos con los argumentos para procesar la cadena de formato proporcionada a través de la función stdarg.
Incluir:	stdio.h
Prototipo:	int vsprintf (char *buf, const rom char *fmt, va_list ap); La función vsprintf da formato a la salida, almacenando los caracteres
Observaciones:	en el búfer de memoria de datos de destino, buf. La cadena de formato, fmt, se procesa como se describe para la función printf excepto que los argumentos consumidos al procesar la cadena de formato se recuperan a través de la función de argumento de longitud variable stdarg.
Valor de retorno:	vsprintf devuelve EOF si se produce un error; de lo contrario, se devuelve el número de caracteres de salida.
Nombre del archivo:	vsprintf.c

_uart_putc

Función:	Salida de un solo carácter al USART (USART1 para dispositivos que tienen más de un USART).
Incluir:	stdio.h
Prototipo:	int _uart_putc (carácter c); _uart_putc
Observaciones:	es la función de salida de biblioteca invocada por putc cuando _H_USART es el flujo de destino. El carácter que se va a emitir se asigna al registro de transmisión (TXREG) cuando el USART está listo para la salida (TRMT está configurado). Si el USART no está habilitado cuando se llama a _uart_putc (TXSTA bit TXEN está borrado), el USART se habilitará (se establecerán TXEN y SPEN) y se establecerá en la salida de tasa de baudios máxima (SPBRG se le asignará un valor de cero). Esta configuración permite que las funciones de la biblioteca de salida de caracteres se utilicen con el soporte de MPLAB IDE para la salida de depuración de USART sin una configuración periférica explícita. _uart_putc devuelve el valor del carácter que se emitió.
Valor de retorno:	devuelve el valor del carácter que se emitió.
Nombre del archivo:	_uart_putc.c

_usuario_putc

Función:	Salida de un solo carácter de una manera definida por la aplicación.
Incluir:	stdio.h
Prototipo:	int _user_putc (carácter c); _user_putc
Observaciones:	es una función definida por la aplicación. Será llamado por las funciones de salida de caracteres para que cada carácter sea emitido cuando el flujo de destino sea _H_USER. _user_putc devuelve el valor del carácter que se emitió.
Valor de retorno:	

Bibliotecas del compilador **MPLAB® C18 C**

NOTAS:



Capítulo 5. Bibliotecas matemáticas

5.1 INTRODUCCIÓN

Este capítulo documenta las funciones de la biblioteca matemática. Incluye dos secciones:

- Biblioteca matemática de coma flotante de 32 bits
- bits • Funciones matemáticas de la biblioteca estándar de C

5.2 BIBLIOTECA MATEMÁTICA DE PUNTO FLOTANTE DE 32 BITS

Las operaciones básicas de coma flotante (sumar, restar, multiplicar, dividir y conversiones entre números flotantes y enteros) cumplen con el estándar IEEE 754 para flotantes de precisión simple con dos excepciones. Las excepciones se discutirán en Subnormales (**Sección 5.2.1.2 "Subnormales"**) y Redondeo (**Sección 5.2.2 "Redondeo"**). El modo extendido y el modo tradicional usan las mismas representaciones flotantes y los resultados de las operaciones flotantes son los mismos.

El estándar IEEE para aritmética de punto flotante binario publicado en 1985 se conoció oficialmente como ANSI/IEEE Std 754-1985 [IEEE85]. La norma tiene tres requisitos importantes:

- representación consistente de números de punto flotante por todas las máquinas que adoptan el estándar;
- operaciones de coma flotante correctamente redondeadas, usando varios modos de redondeo; •
- tratamiento coherente de situaciones excepcionales como la división por cero.

5.2.1 Representación de coma flotante

La representación de números de punto flotante C18 sigue el estándar IEEE 754 de precisión simple. Un número de coma flotante consta de cuatro partes:

1. Un signo
2. Una significancia 3.
- Una base
4. Un exponente Estos

componentes son de la forma $x = \pm d_0.d_1.d_2.d_3 \dots d_{23} \times 2^E$

donde \pm es el signo, $d_0.d_1.d_2.d_3 \dots d_{23}$ es la mantisa, y E es el exponente al que se eleva la base 2. Cada d_i es un dígito (0 o 1). El exponente E es un número entero en el rango E_{\min} a E_{\max} donde $E_{\min} = -126$ y $E_{\max} = 127$.

Los números de formato único utilizan una palabra de 32 bits organizada como un signo de 1 bit, un exponente sesgado de 8 bits $e = E + 127$ y una fracción de 23 bits, que es la parte fraccionaria de la mantisa.

El bit más significativo de la mantisa (d_0) no se almacena. Esto es posible porque su valor se puede deducir del valor del exponente: si el valor del exponente sesgado es 0, entonces $d_0 = 0$, de lo contrario, $d_0 = 1$. El uso de esta convención permite almacenar 24 bits de precisión en 23 bits físicos.

Bibliotecas del compilador MPLAB® C18 C

Signo	Exponente E sesgado de 8 bits	Fracción sin signo de 23 bits f
±	e7e6e5e4e3e2e1e0	d0d1d2d3...d23 En la

implementación C18, los números d0 = 0 no se utilizan (consulte la Sección 5.2.1.2 "Subnormales").

5.2.1.1 NORMALES

Todas las líneas de la Tabla 5-1, excepto la primera y la última, se refieren a números normalizados. La cadena de bits del exponente e7e6e5...e0 utiliza una representación sesgada; la cadena de bits se almacena como la representación binaria de E+127, donde E es el exponente imparcial. El número 127, que se suma al exponente E, se denomina sesgo del exponente. Por ejemplo, el número 1=(1.000...0)2 20 se almacena como

0	01111111	0000000000000000000000000
---	----------	---------------------------

Aquí, la cadena de bits de exponente es la representación binaria de 0+127 y la cadena de bits de fracción es la representación binaria de 0 (la parte fraccionaria de 1,0).

El rango de cadenas de bits de campo de exponente para números normalizados es de 00000001 a 11111110 (los números decimales del 1 al 254), que representan exponentes reales de Emin = -126 a Emax = 127.

TABLA 5-1: FORMATO ÚNICO IEEE-754

Exponente sesgado	Número representado
(00000000)2 = (00)16 = (0)10	± (0.d1d2d3...d23)2 X 2-126
(00000001)2 = (01)16 = (1)10	± (1.d1d2d3...d23)2 X 2-126
(00000010)2 = (02)16 = (2)10	± (1.d1d2d3...d23)2 X 2-125
(00000011)2 = (03)16 = (3)10 ý	± (1.d1d2d3...d23)2 X 2-124
	ý
(01111110)2 = (7E)16 = (126)10	± (1.d1d2d3...d23)2 X 2-1
(01111111)2 = (7F)16 = (127)10	± (1.d1d2d3...d23)2 X 20
(10000000)2 = (80)16 = (128)10 ý	± (1.d1d2d3...d23)2 X 21
	ý ± (1.d1d2d3...d23)2 X
(11111100)2 = (FC)16 = (252)10	2125 ± (1.d1d2d3...d23)2 X
(11111101)2 = (FD)16 = (253)10	2126 ± (1.d1d2d3...d23)2 X
(11111110)2 = (FE)16 = (254)10	2127
(11111111)2 = (FF)16 = (255)10	±ý si d1...d23 = 0 NaN si d1...d23 ý 0

El número normalizado positivo distinto de cero más pequeño que se puede almacenar está representado por

0	00000001	0000000000000000000000000
---	----------	---------------------------

y esto se denota por

$$N_{\min} = (1.000\dots0)2 \times 2^{-126} = 2^{-126} \sim 1.2 \times 10^{-38}$$

Los programadores de C pueden acceder a la constante Nmin mediante la constante de manifiesto FLT_MIN definida en <float.h>.

El número normalizado más grande (equivalentemente, el número finito más grande) está representado por

0	11111110	1111111111111111111111111
---	----------	---------------------------

y esto se denota por

$$N_{\max} = (1.111\dots1)2 \times 2^{127} = (2 - 2^{-23}) \times 2^{127} \sim 3.4 \times 10^{38}$$

Los programadores de C pueden acceder a la constante Nmax mediante la constante de manifiesto FLT_MAX definida en <float.h>.

5.2.1.2 SUBNORMALES

El número normalizado más pequeño que se puede representar es 2-126. El estándar IEEE 754 utiliza la combinación de un exponente e sesgado a cero y una fracción f distinta de cero para representar números más pequeños llamados números subnormales. La estructura de los números subnormales se muestra en la línea 1 de la tabla 5-1. En la implementación flotante C18, los números subnormales siempre se convierten a cero con signo.

IEEE 754 utiliza dos representaciones cero diferentes: +0 y -0. El +0 está representado por todos los bits cero. El -0 está representado por todos los bits cero excepto por el bit de signo.

Si el resultado de una operación flotante es menor que el número normalizado más pequeño, el resultado se establece en un cero con signo antes de que se devuelva. Dado que, en la implementación C18, ninguna operación flotante puede crear una subnormal, una subnormal aparecerá solo si se construye explícitamente como un literal, o si se genera de alguna manera distinta a las operaciones estándar de flotación. Si se usa un valor subnormal en una operación flotante, se convierte automáticamente en un cero con signo antes de que se use en la operación.

5.2.1.3 NaN

Además de admitir infinitos con signo, ceros con signo y números finitos distintos de cero con signo, el formato de punto flotante IEEE especifica una codificación para patrones de error. Estos patrones no son números sino un registro del hecho de que se ha intentado una operación no válida. Cualquier patrón de este tipo es un indicador de error, no un número de punto flotante y, por lo tanto, se denomina No es un número o NaN. Las operaciones no válidas están definidas por el estándar IEEE para incluir:

- Resta de la magnitud de infinitos, como $(+\infty) + (-\infty)$
- Multiplicación de un cero por un infinito, como $(0) \times (+\infty)$
- División de un cero o infinito por cero o infinito, respectivamente, como $(+\infty)/(-\infty)$ o $(+\infty)/(+\infty)$

Los NaN tienen un exponente sesgado de 255, que también es el exponente utilizado para codificar infinitos. La interpretación cuando el exponente sesgado es 255 es: si la fracción es cero, la codificación representa un infinito; si la fracción no es cero, la codificación representa NaN (no un número). Ignorando el bit de signo, que el estándar no interpreta para NaN, hay por lo tanto 223 – 1 NaN posibles. La implementación de C18 devuelve el patrón NaN 7FFF FFFF16 en respuesta a una operación no válida. Es decir, el bit de signo es 0, el exponente es 255 y los bits de fracción son todos 1.

5.2.2 Redondeo

El estándar IEEE-754 exige que las operaciones se redondeen correctamente. El estándar define el valor correctamente redondeado de x , que se denota por $\text{round}(x)$, de la siguiente manera: si x es un número de punto flotante, entonces $\text{round}(x) = x$. De lo contrario, el valor redondeado correctamente depende de cuál de los cuatro modos de redondeo esté en vigor. La implementación flotante C18 utiliza el modo Redondear al más cercano con una ligera modificación al estándar IEEE 754.

El umbral para el redondeo es de aproximadamente 0,502 en lugar de exactamente 0,5. Esto da un ligero sesgo hacia el redondeo hacia cero. Esta modificación da como resultado un ahorro significativo en espacio de código y tiempo de ejecución prácticamente sin consecuencias para los cálculos del mundo real.

Bibliotecas del compilador MPLAB® C18 C

5.3 LAS FUNCIONES MATEMÁTICAS DE LA BIBLIOTECA ESTÁNDAR C

Todas las funciones matemáticas de la biblioteca C estándar devolverán NaN si uno o más de sus argumentos:

- es NaN.
- está fuera del rango de valores para los cuales la función tiene un valor real definido, por ejemplo la raíz cuadrada de un número negativo.

La tabla 5-2 enumera las funciones matemáticas.

TABLA 5-2: FUNCIONES DE LA BIBLIOTECA MATEMÁTICA

Función	Descripción
acos	Calcule el coseno inverso (arcoseno).
como en	Calcule el seno inverso (arcoseno).
un bronceado	Calcule la tangente inversa (arcotangente).
atan2	Calcular la tangente inversa (arcotangente) de una razón.
hacer techo	Calcule el techo (menor número entero).
porque	Calcula el coseno.
aporrear	Calcule el coseno hiperbólico.
Exp	Calcule la exponencial ej.
fabulosos	Calcular el valor absoluto.
piso	Calcule el piso (mayor entero).
fmod	Calcular el resto.
frenético	Dividir en fracción y exponente.
ieetomchp	Convierta un valor de coma flotante de 32 bits con formato IEEE-754 al formato de coma flotante de 32 bits de Microchip.
Idexp	Cargue el exponente: calcule $x * 2^n$.
Iniciar sesión	Calcule el logaritmo natural.
registro10	Calcule el logaritmo común (base 10).
mchptoieee	Convierta un valor de coma flotante de 32 bits en formato Microchip al formato de coma flotante de 32 bits IEEE-754.
modelo	Calcule el módulo.
pow	Calcule la exponencial xy .
pecado	Calcula el seno.
pecado	Calcule el seno hiperbólico.
sqrt	Calcula la raíz cuadrada.
broncearse	Calcular la tangente.
bronceado	Calcular la tangente hiperbólica.

5.3.1 Descripciones de funciones

acos

Función:	Calcular el coseno inverso (arcoseno)
Incluir:	matemáticas.h
Prototipo:	flotar acos(flotar x);
Observaciones:	Esta función calcula el coseno inverso (arcoseno) del argumento x, que debe estar entre -1 y +1. Los argumentos fuera del rango permitido producen errores de dominio y el resultado es NaN.
Valor de retorno:	El valor devuelto es el arcocoseno en radianes y está entre 0 y π .
Nombre del archivo:	acos.c

como en

Función:	Calcule el seno inverso (arcoseno).
Incluir:	matemáticas.h
Prototipo:	flotante asen(flotante x);
Observaciones:	Esta función calcula el seno inverso (arcoseno) del argumento x, que debe estar entre -1 y +1. Los argumentos fuera del rango permitido producen errores de dominio y el resultado es NaN.
Valor de retorno:	El valor devuelto es el arcoseno en radianes y está entre $-\frac{\pi}{2}$ y $\frac{\pi}{2}$.
Nombre del archivo:	asin.c

un bronceado

Función:	Calcule la tangente inversa (arcotangente).
Incluir:	matemáticas.h
Prototipo:	flotar atan(flotar x);
Observaciones:	Esta función calcula la tangente inversa (arcotangente) del argumento x. Si x es NaN, se produce un error de dominio y el valor devuelto es NaN.
Valor de retorno:	El valor devuelto está en radianes y entre $-\frac{\pi}{2}$ y $\frac{\pi}{2}$.
Nombre del archivo:	atan.c

atan2

Función	Calcular la tangente inversa (arcotangente) de una razón.
Incluir:	matemáticas.h
Prototipo:	flotar atan2(flotar y, flotar x);
Observaciones:	Esta función calcula la tangente inversa (arcotangente) de y/x. Si x o y es NaN, se produce un dominio y el valor devuelto es NaN. Si x es un NaN, o si x = y = 0, o si x = y = ∞ , se produce un error de dominio y el valor devuelto es NaN.
Valor de retorno:	El valor devuelto está en radianes y entre $-\pi$ y π .
Nombre del archivo:	atan2.c

Bibliotecas del compilador MPLAB® C18 C

hacer techo

Función:	Calcule el techo (menor número entero).
Incluir:	matemáticas.h
Prototipo:	techo flotante (flotante x);
Observaciones:	Ninguno.
Valor de retorno:	El entero más pequeño mayor o igual que x.
Nombre del archivo:	techo.c

porque

Función:	Calcula el coseno.
Incluir:	matemáticas.h
Prototipo:	flotante cos (flotante x);
Observaciones:	Calcula el coseno de x (en radianes). Un error de dominio resulta de un argumento que es infinito o NaN. Ambos casos devuelven NaN.
Valor de retorno:	El coseno del argumento x.
Nombre del archivo:	porque c

aporrear

Función:	Calcule el coseno hiperbólico.
Incluir:	matemáticas.h
Prototipo:	flotante cosh (flotante x);
Observaciones:	Ninguno.
Valor de retorno:	El coseno hiperbólico del argumento x.
Nombre del archivo:	cosh.c

Exp

Función:	Calcule la exponencial ej.
Incluir:	matemáticas.h
Prototipo:	flotante exp (flotante x);
Observaciones:	Se produce un error de rango si la magnitud de x es demasiado grande. El rango de esta función está limitado a valores para el exponente entre aproximadamente -103.2789 y 88.722283. El valor mínimo del resultado es 2-149 y el máximo es 2127.
Valor de retorno:	El valor de la exponencial ex .
Nombre del archivo:	exp.c

fabulosos

Función:	Calcular el valor absoluto.
Incluir:	matemáticas.h
Prototipo:	flotar fabs(float x);
Observaciones:	Para los argumentos de coma flotante que son ceros e infinitos, el valor devuelto es el argumento con el bit de signo borrado.
Valor de retorno:	El valor absoluto de x.
Nombre del archivo:	fabulosos.c

piso

Función: Calcule el piso (mayor entero).
Incluir: matemáticas.h
Prototipo: piso flotante (flotante x);
Observaciones: Ninguno.
Valor de retorno: El entero más grande menor o igual que x.
Nombre del archivo: piso.c

fmod

Función: Calcular el resto.
Incluir: matemáticas.h
Prototipo: float fmod(float x, float y);
Observaciones: Ninguno.
Valor de retorno: El resto para x módulo y.
Nombre del archivo: fmod.c

frenético

Función: Dividir en fracción y exponente.
Incluir: matemáticas.h
Prototipo: float frexp(float x, int *pexp); Separa el argumento x en
Observaciones: dos partes que se ajustan a esta fórmula: $x = \text{frexp}(x, *pexp) \times 2^{*\text{pexp}}$ El valor entero, que se almacena en la ubicación pexp, se elige de modo que la parte fraccionaria del resultado esté entre $\frac{1}{2}$ y 1.
Valor de retorno: Resultado fraccionario que satisface las condiciones enumeradas anteriormente.
Nombre del archivo: frex.c

ieeetomchp

Función: Convierta un valor de coma flotante de 32 bits con formato IEEE-754 al formato de coma flotante de 32 bits de Microchip.
Incluir: matemáticas.h
Prototipo: ieeetomchp largo sin firmar (float v);
Observaciones: Esta función ajusta el bit de signo de la representación de punto flotante para que se ubique según lo requiera el formato de Microchip:

	eb	f0	f1	f2
IEEE-754 de 32 bits	s eeee	xxxx xxxx	xxxx xxxx xxxx	xxxx
Microchip de 32 bits	e eee	eeee xxxx xxxx	xxxx xxxx xxxx	xxxx

s = bit de signo e=exponente x=significando

Valor de retorno: El valor convertido de 32 bits.
Nombre del archivo: ieeetomchp.c

Bibliotecas del compilador MPLAB® C18 C

Idexp

Función: Cargar exponente - calcular x^{*2n} .

Incluir: matemáticas.h

Prototipo: float Idexp(float x, int n);

Observaciones: Ninguno.

Valor de retorno: Devuelve el valor de x^{*2n} .

Nombre del archivo: Idexp.c

Iniciar sesión

Función: Calcule el logaritmo natural.

Incluir: matemáticas.h

Prototipo: registro flotante (floatante x);

Observaciones: Se produce un error de dominio si el argumento no está en el intervalo [0, +∞].

Valor de retorno: Logaritmo natural de x.

Nombre del archivo: log.c

registro10

Función: Calcule el logaritmo común (base 10).

Incluir: matemáticas.h

Prototipo: float log10 (float x); Se produce un

Observaciones: error de dominio si el argumento no está en el intervalo [0, +∞]. log10x.

Valor de retorno:

Nombre del archivo: log10.c

mchptoiieee

Función: Convierta un valor de coma flotante de 32 bits en formato Microchip al formato de coma flotante de 32 bits IEEE-754.

Incluir: matemáticas.h

Prototipo: float ieeetomchp (v larga sin firmar);

Observaciones: Esta función ajusta el bit de signo de la representación de punto flotante para ubicarlo según lo requiera el formato IEEE:

eb	f0	f1	f2
IEEE-754 de 32 bits seee eeee xxxx xxxx xxxx xxxx xxxx xxxx			
Microchip de 32 bits eeee eeee sxxx xxxx xxxx xxxx xxxx xxxx			

s = bit de signo

e=exponente

x=significando

Valor de retorno: El valor de punto flotante convertido.

Nombre del archivo: mchptoiieee.c

modelo

Función:	Calcule el módulo.
Incluir:	matemáticas.h
Prototipo:	modf flotante (x flotante, flotante * ipart);
Observaciones:	Esta función separa el argumento x en partes enteras y fraccionarias. Se devuelve la parte fraccionaria y la parte entera se almacena en la ubicación ipart. Si el argumento es NaN, los resultados tanto para la parte fraccionaria como para la entera también serán NaN.
Valor de retorno:	Porción fraccionaria de x.
Nombre del archivo:	modf.c

pow

Función:	Calcule la exponencial xy .
Incluir:	matemáticas.h
Prototipo:	flotar pow(flotar x, flotar y);
Observaciones:	Los errores de dominio ocurren si x es finito y negativo e y es finito y no un número entero; también si x es cero y y es menor o igual a cero. Se produce un error de rango si xy es demasiado grande o demasiado pequeño para ser representado. En tal caso, se devuelve un infinito o cero correctamente firmado y se señala un error de rango. xy .
Valor de retorno:	
Nombre del archivo:	pow.c

pecado

Función:	Calcula el seno.
Incluir:	matemáticas.h
Prototipo:	flotante sin(flotante x);
Observaciones:	Calcula el seno de x (en radianes). Un error de dominio resulta de un argumento que es infinito o NaN. Ambos casos devuelven NaN.
Valor de retorno:	El seno de x.
Nombre del archivo:	pecado.c

pecado

Función:	Calcule el seno hiperbólico.
Incluir:	matemáticas.h
Prototipo:	flotante sinh(flotante x);
Observaciones:	Ninguno.
Valor de retorno:	El seno hiperbólico del argumento x.
Nombre del archivo:	sinh.c

Bibliotecas del compilador MPLAB® C18 C

sqrt

Función: Calcula la raíz cuadrada.
Incluir: matemáticas.h
Prototipo: float sqrt(float x);
Observaciones: Se produce un error de dominio si el argumento x es estrictamente negativo. La raíz cuadrada principal existe y es computable para cada número x de punto flotante no negativo.
Valor de retorno: La raíz cuadrada de x.
Nombre del archivo: sqrt.c

broncearse

Función: Calcular la tangente.
Incluir: matemáticas.h
Prototipo: float bronceado(float x);
Observaciones: Calcula la tangente de x (en radianes). Se produce un error de dominio si el argumento es infinito o NaN. Ambos casos devuelven NaN.
Valor de retorno: La tangente de x.
Nombre del archivo: bronceado.c

bronceado

Función: Calcular la tangente hiperbólica.
Incluir: matemáticas.h
Prototipo: float tanh(float x);
Observaciones: Si el argumento es NaN, el valor devuelto es NaN.
Valor de retorno: La tangente hiperbólica de x.
Nombre del archivo: tanh.c



COMPILEADOR MPLAB® C18 C BIBLIOTECAS

Glosario

A

Sección absoluta

Una sección con una dirección fija que el enlazador no puede cambiar.

Acceda a la memoria

Registros especiales de propósito general (GPR) en los microcontroladores PIC18 PICmicro que permiten el acceso independientemente de la configuración del Registro de selección de banco (BSR).

Dirección

El código que identifica dónde se almacena una parte de la información en la memoria.

Estructura anónima Un

objeto sin nombre.

ANSI

Instituto Americano de Estándares Nacionales

Ensamblador

Una herramienta de lenguaje que traduce el código fuente ensamblador en código de máquina.

Asamblea

Un lenguaje simbólico que describe el código de máquina binario en una forma legible.

Sección asignada Una

sección que se ha asignado a un bloque de memoria de destino en el archivo de comandos del enlazador.

Asíncronamente

Múltiples eventos que no ocurren al mismo tiempo. Esto generalmente se usa para referirse a las interrupciones que pueden ocurrir en cualquier momento durante la ejecución del procesador.

B

Binario

El sistema de numeración de base dos que usa los dígitos 0-1. El dígito más a la derecha cuenta unidades, el siguiente cuenta múltiplos de 2, luego $2^2 = 4$, etc.

C

Unidad central de procesamiento

La parte de un dispositivo que es responsable de obtener la instrucción correcta para su ejecución, decodificar esa instrucción y luego ejecutarla. Cuando es necesario, trabaja en conjunto con la unidad aritmético lógica (ALU) para completar la ejecución de la instrucción. Controla el bus de direcciones de la memoria de programa, el bus de direcciones de la memoria de datos y los accesos a la pila.

Compilador

Un programa que traduce un archivo fuente escrito en un lenguaje de alto nivel a código de máquina.

Bibliotecas del compilador MPLAB® C18 C

Compilación condicional El

acto de compilar un fragmento de programa solo si cierta expresión constante, especificada por una directiva de preprocesador, es verdadera.

UPC

Unidad Central de procesamiento

mi

endianidad

El orden de los bytes en un objeto de varios bytes.

Archivo de errores

Un archivo que contiene los diagnósticos generados por el compilador MPLAB C18.

Modo extendido

En el modo extendido, el compilador utilizará las instrucciones extendidas (es decir, ADDFSR, ADDULNK, CALLW, MOVSF, MOVSS, PUSHL, SUBFSR y SUBULNK) y el direccionamiento indexado con desplazamiento literal.

F

Error fatal

Un error que detendrá la compilación inmediatamente. No se producirán más mensajes.

Puntero de cuadro

Un puntero que hace referencia a la ubicación en la pila que separa los argumentos basados en la pila de las variables locales basadas en la pila.

De pie

Una implementación que acepta cualquier programa estrictamente conforme que no utiliza tipos complejos y en el que el uso de las funciones especificadas en la cláusula de la biblioteca (ANSI '89 estándar cláusula 7) se limita al contenido de los encabezados estándar <float.h>, <iso646.h>, <limits.h>, <stdarg.h>, <stdbool.h>, <stddef.h> y <stdint.h>.

H

hexadecimal

El sistema de numeración de base 16 que usa los dígitos 0-9 más las letras AF (o af).

Los dígitos AF representan valores decimales de 10 a 15. El dígito más a la derecha cuenta unidades, el siguiente cuenta múltiplos de 16, luego 162 = 256, etc.

Lenguaje de alto nivel

Un lenguaje para escribir programas que está más alejado del procesador que el ensamblador.

CIE

Depurador en circuito

HIELO

Emulador en circuito

IDE

Entorno de desarrollo integrado

IEEE

Instituto de Ingenieros Eléctricos y Electrónicos **Interrupción**

Una señal a la CPU que suspende la ejecución de una aplicación en ejecución y transfiere el control a un ISR para que el evento pueda ser procesado. Una vez completada la ISR, se reanuda la ejecución normal de la aplicación.

Rutina de servicio de interrupción

Una función que maneja una interrupción.

YO ASI

Organización Internacional de Normalización

ISR

Rutina de servicio de interrupción

L**Latencia**

El tiempo entre el momento en que ocurre un evento y la respuesta al mismo.

bibliotecario

Un programa que crea y manipula bibliotecas.

Biblioteca

Una colección de módulos de objetos reubicables.

Enlazador

Un programa que combina archivos de objetos y bibliotecas para crear código ejecutable.

LittleEndian

Dentro de un objeto dado, el byte menos significativo se almacena en direcciones más bajas.

METRO**Modelo de memoria**

Una descripción que especifica el tamaño de los punteros que apuntan a la memoria del programa.

microcontrolador

Un chip altamente integrado que contiene una CPU, RAM, algún tipo de ROM, puertos de E/S y temporizadores.

Ensamblador MPASM

Macro ensamblador reubicable de Microchip Technology para familias de microcontroladores PICmicro.

MPLIB Object Librarian

Bibliotecario de Microchip Technology para familias de microcontroladores PICmicro.

MPLINK Object Linker Linker

de Microchip Technology para familias de microcontroladores PICmicro.

norte**Modo no extendido**

En el modo no extendido, el compilador no utilizará las instrucciones extendidas ni las indexadas con direccionamiento de desplazamiento literal.

Bibliotecas del compilador MPLAB® C18 C

O

Archivo de

objeto Un archivo que contiene código de objeto. Puede ser inmediatamente ejecutable o puede requerir vinculación con otros archivos de código de objeto (por ejemplo, bibliotecas) para producir un programa ejecutable completo.

Código de objeto

El código de máquina generado por un ensamblador o compilador.

octales

El sistema numérico de base 8 que solo usa los dígitos 0-7. El dígito más a la derecha cuenta unidades, el siguiente dígito cuenta múltiplos de 8, luego 82 = 64, etc.

PAG

pragma

Una directiva que tiene significado para un compilador específico.

R

RAM

Memoria de acceso aleatorio

Memoria de acceso aleatorio

Un dispositivo de memoria en el que se puede acceder a la información en cualquier orden.

Memoria de sólo lectura

Hardware de memoria que permite un acceso rápido a los datos almacenados permanentemente pero evita la adición o modificación de los datos.

ROM

Memoria de sólo lectura

recursivo

Autorreferencial (por ejemplo, una función que se llama a sí misma).

reentrante

Una función que puede tener varias instancias activas simultáneamente. Esto puede suceder debido a la recursividad directa o indirecta oa través de la ejecución durante el procesamiento de interrupciones.

reubicable

Un objeto cuya dirección no ha sido asignada a una ubicación de memoria fija.

Modelo de tiempo de ejecución

Conjunto de supuestos bajo los cuales opera el compilador.

S

Sección

Una porción de una aplicación ubicada en una dirección específica de memoria.

Atributo de sección

Una característica adscrita a una sección (por ejemplo, una sección de acceso).

Registro de función especial

Registros que controlan las funciones del procesador de E/S, el estado de E/S, los temporizadores u otros modos o periféricos.

Clase de

almacenamiento Determina la vida útil de la memoria asociada con el objeto identificado.

Calificador de

almacenamiento Indica propiedades especiales de los objetos que se declaran (p. ej., const).

V**Vector**

Las ubicaciones de memoria a las que saltará una aplicación cuando se produzca un reinicio o una interrupción. ocurre.

Bibliotecas del compilador **MPLAB® C18 C**

NOTAS:



COMPILEADOR MPLAB® C18 C

BIBLIOTECAS

Índice

Símbolos

_uart_putc	157
_usuario_putc	157

A

Convertidor A/D	9
Ocupado.....	10
Cerrar.....	10
Convertir	10
Ejemplo de uso	16
Abierto	10, 12, 14
Leer	15
Establecer canal	16
Absoluto Valor.....	164
AckI2C	22
acos	163
Caracteres alfabéticos	120
Caracteres alfanuméricos	120
ANSI.....	5
Arcocoseno	163
Arcoseno	163
Arcotangente.....	163
asin	163
Modo asíncrono	69
atan	163
atán2	163
atob	124
de	124
atoi	125
atoles	125

B

baudioUSART.....	73
Restablecimiento de caída de tensión	
146 btoa	125
construir.bat	
6 BusyADC	10
BusyUSART.....	67
BusyLCD	77

C

c018.o	5
c018_e.o	
5 c018i.o	5
c018i_e.o	5
c018iz.o.....	5
c018iz_e.o.....	5
CAN2510, Externo	
Modificación de 82 bits	
Lectura de 83 bytes	
84 Escritura de bytes	
84 Lectura de datos	84

Listo para datos	
85 Deshabilitar	
86 Habilitar	
86 Estado de error	
87 Inicializar..	
87 Habilitar interrupción	
91 Estado de interrupción	
92 Carga extendida al búfer	93
Carga extendida a RTR	94
Carga estándar a tampón.	92
Cargar estándar a RTR.....	94
Modo de lectura	95
Leer estado	95
Reiniciar.....	
96 Búfer de envío	
96 Lectura secuencial	
96 Escritura secuencial	
97 Establecer prioridad de búfer	
97 Establecer el filtro de mensajes en Extendido	99
Extendido	99
Establecer filtro de mensajes en estándar	98
modo	98
Establezca el filtro único en Extendido	100
Establezca el filtro único en Estándar	100
Establezca Máscara única en Extendida	
101 Establecer máscara única en estándar	
101 Escribir mensaje extendido	
104 Escribir mensaje estándar	102, 103
CAN2510BitModify.....	83
CAN2510Lectura de bytes.....	84
CAN2510Escritura de bytes.....	84
CAN2510 lectura de datos	
84 CAN2510DataReady.....	85
CAN2510 Desactivar.....	86
Habilitar CAN2510	86
Estado de error CAN2510	87
CAN2510Inic.....	87
CAN2510 Habilitar interrupción	
91 CAN2510 Estado de interrupción	
92 CAN2510LoadBufferStd.....	92
CAN2510LoadBufferXtd.....	93
CAN2510CargarRTRStd.....	94
CAN2510LoadRTRXtd.....	94
CAN2510Modo de lectura	
95 CAN2510Lectura de estado	
95 CAN2510Reiniciar	
96 CAN2510Búfer de envío	96
CAN2510Lectura secuencial	96
CAN2510Escritura secuencial	
97 CAN2510Establecer prioridad de búfer	
97 Modo de configuración CAN2510	

Bibliotecas del compilador MPLAB® C18 C

CAN2510SetMsgFilterXtd	99	Largo a cadena	126
CAN2510SetSingleFilterStd	100	Cadena a Byte.....
CAN2510Establecer filtro único Xtd	100	124 Cuerda a flotar.....	124
CAN2510Establecer máscara única		De cadena a entero	125
estándar	101	Cuerda a larga	125 Largo
CAN2510SetSingleMaskXtd	101	sin signo a cadena	128
CAN2510WriteStd	102, 103	Inicialización de datos	5
CAN2510WriteXtd.....	104	DataRdyMwire	38
Captura		DataRdySPI	49
18 Cerrar.....		DataRdyUSART	68
17 Ejemplo de uso	20	Retraso	144
Abierto	18	1 unidad	144
Leer	19	Múltiplos de 1,000 Tcy	145
techo	164	Múltiplos de 10 Tcy	144
Techo	164	10 Múltiplos de ,000 Tcy	145
Carácter er Clasificación		Múltiplos de 100 Tcy	144
Alfabético	120	Retardo 100TCYx	144
Alfanumérico	120	Retardo 10KTCYx	145
Mando	120	Retardo 10TCYx	144
Decimales	121	Retardo 1KTCYx.....	145
Gráfica.....	121 Hexadecimales	Retraso1TCY.....	123
Letras alfábéticas en minúsculas	121	144	
Imprimible	122	Directorios	
Puntuación	122	h.....	75, 105, 111
Alfabéticamente en mayúsculas.....	123	librerías.....	5, 6
Espacio en blanco.....	122	pmc	9 ,
Funciones de clasificación de caracteres	119	75 origen	
Funciones de salida de caracteres	149	5 puesta en marcha	
Salida de caracteres	155, 157	6 Deshabilitar Pullups.....	
Salida con formato	150, 155, 156, 157	35 Convenciones de documentación	2
Salida sin formato	154, 155	mi	
BorrarSWCSSPI	112	ECapture	
clib.lib	6	Cerrar.....	17
clib_e.lib	6	Abierto	18 EE
Prueba_reloj	106	Funciones de la interfaz del dispositivo de memoria	
Cerrar ADC	10	29 EEAckPolling	29
CerrarCapture	17	EEByteWrite	29
CerrarECapture	17	EEActualAñadirLectura.....	30
Cerrar I2C	22	EEPageWrite.....	31 EELectura
CerrarCableM.....	37	aleatoria	32
CerrarPORTB.....	35	EESequentialRead.....	33
Cerrar PWM	44	Habilitar pullups	35
CerrarRBxINT.....	35	Ejemplos Convertidor A/D	
Cerrar SPI	49	16 Capturar	20 I
Temporizador de cierre..		2C, Hardware	34 I 2C,
57 CerrarUSART.....	67	Software..	109
Logaritmo común	166	LCD.....	81
Carácter de control	120	Microhilo.....	42 SPI,
ConvertADC	10	Hardware.....	54 SPI,
cos.....	164	software	113
cosh.....	164	Temporizadores	
Coseno	164 Servicio	65 UART, software	116
de notificación al cliente	4 Atención al	USART, Hardware	74
cliente	4	exp.....	164
D		Exponente	159
Funciones de conversión de datos	124	Sesgo de exponente
Byte a cadena	125	160 Exponential.....	164, 167
Convertir caracteres a minúsculas	127	F	
Convertir carácter a mayúsculas	127 De	fabulosas	164
entero a cadena	126		

flotar.h	160	
Punto flotante	159	
Bibliotecas	159	
piso	165	
FLT_MAX.....	160	
FLT_MIN	160	
mod	165	
fprintf	150	
entradas	154	
frex	165	
 GAMMA		
obtenerI2C.....	23	
getcMwire.....	38	
getcSPI	49	
obtener UART		
115 obtenerUSART		
68 obtenerI2C.....		
23 obtieneMcable.....	38	
obtiene SPI	50	
obtiene UART	115	
obtiene USART		
68 Caracteres gráficos	121	
 H		
h directorio	75, 105, 111	
Coseno Hiperbólico.....	164	
Seno Hiperbólico		
167 Tangente Hiperbólica.....	168	
 I		
Funciones de los puertos de E/S Consulte Puerto		
B.....	34 Macros de software I	
2C.....	105 I 2C,	
Herrajes	21	
Confirmar	22	
Cerrar.....	22	
Sondeo de reconocimiento de EEPROM		
29 Escritura de bytes de EEPROM		
29 Lectura de la dirección actual de la EEPROM		
30 Escritura de la página de la		
EEPROM	31 Lectura aleatoria	
de EEPROM	32 Lectura	
secuencial de EEPROM	33 Ejemplo	
de uso	34 Obtener	
carácter	23 Obtener	
cadena	23	
Inactivo	24	
Sin reconocimiento e.....	24	
Abrir	25	
Poner carácter	25	
Poner cadena	26	
Leer	26	
Reiniciar	27	
Inicio	27 Detener	
I 2C, Software	105	
Confirmar	106	
Prueba de reloj		
106 Ejemplo de uso		
109 Obtener carácter		
106 Obtener cadena	106	
 Sin reconocimiento		106, 107
Poner carácter	107	
Poner cadena		
107 Leer	107	
Reiniciar	107	
Inicio	108	
Detener..	108	
Escribir.....	108	
IdleI2C	24 IEEE	
754.....	159	
IEEE-754.....	165, 166	
ieetomchp.....	165 Datos	
Inicializados.....	5	
Captura de entrada	17	
Rutina de servicio de interrupción		
171 rutina de servicio de interrupción		
171 Coseno Inverso		
163 Seno inverso		
163 Tangente inversa	163	
isalnum.....	120	
isalfa	120	
esBOR.....	146	
escntrl.....	120 es	
un dígito	121	
isgrafe	121 es	
inferior	121	
esLVD.....	146	
esMCLR	147	
esPOR.....	147	
esimprimir.....	122	
espuntual.....	122	
esespacio	122	
essuperior	123	
esWDTTO	147	
esWDTWU	148	
esWU.....	148	
esxdigit	123	
itoa	126	
 L		
Retardos externos de la pantalla		
LCD	77 Macros	
externas	76 Pantalla LCD,	
externa.....	75	
Ocupado	77	
Ejemplo de uso	81	
Abrir	77	
Poner carácter.....	77, 80	
Poner cadena de ROM.....	78	
Poner cadena	78	
Leer dirección	78 Leer	
datos	79 Configurar	
la dirección del generador de caracteres		
79 Establecer dirección de datos de pantalla		
79 Comando de escritura		
80 Escribir datos	80	
Idxp.....	166	
directorio lib.....	5, 6 Bibliotecas	
Independiente del procesador	6	
Específico del procesador	7	

Bibliotecas del compilador MPLAB® C18 C

Reconstruyendo	6—
5–7 Código fuente	6—
7 Descripción general de la biblioteca	
5 Little Endian	171
Exponente de carga	
166 registro	
166 log10	166
Caracteres en minúsculas	121, 127, 137
Itoa	126
METRO	
principal	5
makeclib.bat	6
makeplib.bat...	7
Bibliotecas de Matemáticas	
159 Valor absoluto	
164 Techo	164
Logaritmo común	166
Coseno	164
Exponencial	164
Piso....	165
Fracción y Exponente	
165 Coseno Hiperbólico.....	
164 Seno Hiperbólico.....	
167 Tangente Hiperbólica.....	
168 IEEE-75 4 Conversión	165,
166 Coseno inverso	
163 Seno inverso	
163 Tangente inversa	
163 Exponente de carga	
166 Módulo	167
Logaritmo natural	166
Potencia..	167
Resto	165
Seno.....	167
Raíz cuadrada	168
Tangente.....	168
mchptoieee	166
RCML	147
memoria	130
memcmp	130
memcmppgm.....	130
memcmppgm2ram	130
memcmpram2pgm	130
memcpy.....	
131 memcpypgm2ram.....	131
movimiento de memoria	
132 memmovepgm2ram	132
Funciones de manipulación de la memoria	128
Comparar	130
Copiar...	131
Mover	132
Buscar	130
Establecer.....	
133 conjunto de	
memoria	133
Sitio Web de Microchip	3
Microhilo.....	37
Cerrar.....	37
Listo para datos	
38 Ejemplo de uso.....	42

Obtener cadena	
38 Abierto	39
39 Poner carácter.....	
Leer.....	40
Escribir.....	
41 modelo	
167 Módulo	
167 Ensamblador MPASM	
6, 7 Bibliotecario de MPLIB	6, 7
norte	
NaN	161
Logaritmo natural.....	166
Números normalizados	160
Normales	160
NotAckI2C24
o	
OpenADC.....	10, 12, 14
Captura abierta.....	18
OpenECapture	18
OpenI2C	25
OpenMwire	39
AbrirPORTB	36
OpenPWM	45
OpenRBxINT	36
OpenSPI.....	50
Open SWSPI	112
Temporizador Abierto	
58–62 OpenUART	
115 Abrir USART	
69 OpenXLCD	77
PAG	
Bibliotecas Periféricas	7
directorio pmc	9, 75
PUERTO Cerrar.....	35
Desactivar interrupciones	
35 Deshabilitar pull-ups	
35 Habilitar interrupciones	
36 Habilitar pull-ups	35
Abrir	36
potencia.....	
167 imprimir f	
155 Funciones de modulación de ancho de pulso	44
putc.....	155
putcI2C.....	25
ponercMwire.....	39
putcSPI.....	51
putcSWSPI	112
putcUART.....	115
putcUSART	70
putcXLCD.....	77, 80
putrsUSART	70
putrsXLCD.....	78
pone.....	155
poneI2C.....	26
pone SPI.....	51
poneUART	115
pone USART	70
38 Ejemplo de uso.....	38

poneXLCD.....	78	Escribe.....	53
PWM.....	44	SPI, Software	111
Cerrar	44	Borrar selección de chip	112
Abrir	45	Ejemplo de uso	113
Establecer el ciclo de trabajo		Macros	111
46 Establecer salida ECCP	47	Abierto	112
R		Poner carácter	112
rand.....	126	Establecer selección de chip	
LecturaADC.....	15	113 Escribir	
ReadAddrXLCD	78	113 sprintf	156
Captura de lectura.....		pies cuadrados	
19 ReadDataXLCD	79	168 Raíz cuadrada	
LeerI2C.....	26	168 hilos	127
ReadMwire.....	40	directorio src	5
Leer SPI.....	52	SSP	21, 22
Temporizador de lectura		Pila, Software	5
63 LeerUART.....	116	Biblioteca C estándar.....	
ReadUSART	71	6 InicioI2C	27
Reconstrucción de bibliotecas Procesador		Código de inicio	5
independiente	6 Específico	directorio de inicio	6
del procesador	7 Código	Restablecimiento de estado	
de inicio...	6	148 DetenerI2C	
Referencias	2	28 strcat	133
Resto	165 Funciones	strcatpgm2ram	133
de reinicio	146	strchr	134
Apagón	146	strcmp	134
Baja Tensión Detectar	146	strcmppgm2ram	134
Borrar Maestro	147	tiras	135
Encendido	147	strcpypgm2ram	135
Estado	148	strcspn	136
Despertar	148	Funciones de manipulación de cadenas	
Tiempo de espera del temporizador de		128 Anexar	133, 137
vigilancia	147	Comparar	134, 138
Activación del		Convertir a minúsculas	137
temporizador de vigilancia	148	Convertir a mayúsculas	143
ReiniclarI2C.....		Copiar	135, 139
27 Redondeo.....	159, 161	Longitud	159, 136
S		Buscar	134, 140, 142
EstablecerCGRamAddr		Tokenizar	142
79 EstablecerChanADC		estrellas	136
16 Establecer DCPWM		strlwr	137
46 EstablecerDDRamAddr		strncat	137
79 Establecer salida PWM		strncatpgm2ram	137
47 EstablecerSWCSSPI		strncmp	138
113 Definiciones SFR		strncpy	139
7 Significado	159	strncpypgm2ram	139
pecado		strpbrk	140
167 Seno		strrchr	140
167 sen		hilos	calle
167 Dormir.....		141	142
148 SPI, hardware.....	48	strok	142
Cerrar	49	instrumentos	
Datos listos.....	49	143 Números subnormales	
Ejemplo de uso	54	161 Subnormales	159, 161
Obtener carácter	49	SWAk I2C.....	106, 107
Obtener cadena		SWGetI2C	106
50 Abrir	50	SWGetsI2C	106
Poner carácter	51	SWNotAckI2C	106
Poner cadena		SWPutI2C	107
51 Leer	52	SWPutsl2C	107

Bibliotecas del compilador MPLAB® C18 C

SWReadI2C	107
SWReiniciarI2C	107
SWStartI2C	108
SWStopI2C	108
SWWriteI2C.....	108
Modo síncrono	69
T	
bronceado	
168 Tangente	
168 tanh	168
Temporizadores	
57 Cerrar.....	
57 Ejemplo de uso	65
Abrir	58–62
Leer	63
Escribir.....	
64 hacia abajo	
127 topper	127
tu	
UART, Software	114
Retrasos	114
Ejemplo de uso.....	116
Obtener carácter	115
Obtener cadena	
115 Macros	
114 Abrir	
115 Poner carácter	
115 Poner cadena	
115 Leer	
116 Escribir.....	
.116 ultoa	
128 Caracteres en mayúsculas	123, 127,
137 USART, hardware.....	
66b aud	73
Ocupado	
67 Cerrar.....	
67 Datos listos	68
Ejemplo de uso	
74 Obtener carácter	68
Obtener cadena	
68 Abrir	
69 Poner carácter..	
70 Poner cadena	
70 Leer	
71 Escribir.....	72
V	
vfprintf	156
vprintf	156
vsprintf.....	157

W	
Temporizador de vigilancia (WDT)	147,
148 WriteCmdLCD	80
WriteDataLCD	80
Escribir I2C	28
EscribirMcable	41
Escribir SPI	
53 Escribir SWSPI	
113 Temporizador de escritura	64
Escribir UART	116
Escribir USART.....	72

NOTAS:



MICROCHIP

VENTAS Y SERVICIO EN TODO EL MUNDO

AMÉRICAS

Oficina corporativa
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Teléfono: 480-792-7200
Fax: 480-792-7277
Asistencia técnica: <http://support.microchip.com> Dirección web:
www.microchip.com

Atlanta Alpharetta, GA
Tel: 770-640-0034 Fax: 770-640-0307

Bostón

Westborough, MA
Teléfono: 774-760-0087
Fax: 774-760-0088

chicago

Itasca, IL
Teléfono: 630-285-0071
Fax: 630-285-0075

dallas

Addison, Texas
Teléfono: 972-818-7423
Fax: 972-818-2924

detroit

Farmington Hills, Michigan
Teléfono: 248-538-2250
Fax: 248-538-2260

kokomo

Kokomo, IN
Teléfono: 765-864-8360
Fax: 765-864-8387

los Angeles

Misión Viejo, CA
Teléfono: 949-462-9523
Fax: 949-462-9608

San Jose

Vista a la montaña, CA
Teléfono: 650-215-1444
Fax: 650-961-0286

toronto

mississauga, ontario,
Canadá
Teléfono: 905-673-0699
Fax: 905-673-6509

ASIA/PACÍFICO

Australia - Sidney
Teléfono: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Pekín
Teléfono: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdú
Teléfono: 86-28-8676-6200
Fax: 86-28-8676-6599

China - Fuzhou
Teléfono: 86-591-8750-3506
Fax: 86-591-8750-3521

China - RAE de Hong Kong
Teléfono: 852-2401-1200
Fax: 852-2401-3431

China - Shanghái
Teléfono: 86-21-5407-5533
Fax: 86-21-5407-5066

China-Shenyang
Teléfono: 86-24-2334-2829
Fax: 86-24-2334-2393

China-Shenzhen
Teléfono: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Shunde
Teléfono: 86-757-2839-5507
Fax: 86-757-2839-5571

China-Qingdao
Teléfono: 86-532-502-7355
Fax: 86-532-502-7205

ASIA/PACÍFICO

India - Bangalore
Teléfono: 91-80-2229-0061
Fax: 91-80-2229-0062

India - Nueva Delhi
Teléfono: 91-11-5160-8631
Fax: 91-11-5160-8632

Japón - Kanagawa
Tel: 81-45-471-6166
Fax: 81-45-471-6122

Corea - Seúl
Teléfono: 82-2-554-7200
Fax: 82-2-558-5932 o
82-2-558-5934

Malasia - Penang
Teléfono: 011-604-646-8870
Fax: 011-604-646-5086

Filipinas - Manila
Teléfono: 011-632-634-9065
Fax: 011-632-634-9069

Singapur
Teléfono: 65-6334-8870
Fax: 65-6334-8850

Taiwán - Kaohsiuung
Teléfono: 886-7-536-4818
Fax: 886-7-536-4803

Taiwán - Taipéi
Teléfono: 886-2-2500-6610
Fax: 886-2-2508-0102

Taiwán - Hsinchu
Teléfono: 886-3-572-9526
Fax: 886-3-572-6459

EUROPA

Austria-Weis
Teléfono: 43-7242-2244-399
Fax: 43-7242-2244-393

Dinamarca - Ballerup
Teléfono: 45-4450-2828
Fax: 45-4485-2829

Francia - Massy
Teléfono: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Alemania - Ismaning
Teléfono: 49-89-627-144-0
Fax: 49-89-627-144-44

Italia - Milán
Teléfono: 39-0331-742611
Fax: 39-0331-466781

Paises Bajos - Drunen
Teléfono: 31-416-690399
Fax: 31-416-690340

Inglaterra - Berkshire
Teléfono: 44-118-921-5869
Fax: 44-118-921-5820