PIC GCC Library

Biblioteca Estándar y de Componentes

Ing. Fernando Pujaico Rivera
Desarrollador Principal
Investigador
Universidad Nacional de Ingenieria

Ing. Pedro Jose Ramirez Gutierrez

Prof. Pierre launay

Santiago González Rodríguez

PIC GCC Library : Biblioteca Estándar y de Componentes

por Ing. Fernando Pujaico Rivera, Ing. Pedro Jose Ramirez Gutierrez, Prof. Pierre launay, y Santiago González Rodríguez

PIC GCC Library (http://pic-gcc-library.sourceforge.net/) es un conjunto de bibliotecas para el compilador PIC GCC (https://forja.rediris.es/projects/cls-pic-16f877/). PIC GCC es un compilador de C para microcontroladores "PIC" de la familia 16F de Microchip

<fernando.pujaico.rivera en gmail.com>

Tabla de contenidos

Lista de tablas

Capítulo 1. Introducción

PIC GCC (http://forja.rediris.es/frs/?group_id=101) es un compilador de leguaje c para microcontroladores de la familia PIC16 de MICROCHIP.

Figura 1-1. Logo de Pic Gcc Library

1.1. Descripción de Carpetas

Tabla 1-1. Descripción de carpetas en Pic-Gcc-Library

Carpetas en la raiz	Descripción de carpetas
bin	Carpeta con las herramientas para la compilación.
devel	Carpeta con el código fuente de las bibliotecas (sólo en versión para desarrolladores).
doc	Carpeta con la documentación para el uso de las bibliotecas.
examples	Carpeta con ejemplos de uso (no modificar).

Carpetas en la raiz	Descripción de carpetas
include	Carpeta con la definición de todas las funciones de las funciones.
lib	Carpeta con las bibliotecas estáticas predefinidas.
refman	Carpeta con el manual de referencia de las bibliotecas.
schematics	Carpeta con los diagramas esquemáticos correspondientes a los ejemplos.

1.2. Métodos de Compilación

Para compilar un programa se tiene dos opciones:

1.2.1. Primera Forma

```
pic-gcc -Os ejemplo.c -mp=16f877a -S -o ejemplo.asm
-I /DIRX/include/disp -I /DIRX/include/drivers
-I /DIRX/include/pic -I /DIRX/include/util
-I /DIRX/include
```

```
gpasm -c ejemplo.asm -p 16f877a -o ejemplo.o
```

```
gplink ejemplo.o
/DIRX/lib/libgcc.a /DIRX/lib/libc.a
/DIRX/lib/libutil.a /DIRX/lib/libdisp_16f877a.a
-o ejemplo.hex
-s /usr/share/gputils/lkr/16f877a.lkr
```

1.2.2. segunda Forma

```
./compila.sh ejemplo 16f877a .
```

si el programa que quiero compilar no esta en la carpeta en donde se encuentra compila.sh

```
./compila.sh ejemplo 16f877a  /directorio_de_codigo_fuente
```

1.3. Bibliotecas en Pic-Gcc-Library

Tabla 1-2. Carpeta en la carpeta INCLUDE de Pic-Gcc-Library

Carpetas	Descripción de carpetas
disp	Contiene las funciones para controlar los perifericos de cada dispositivo: UART, SPI, PWM, IO, etc. Las funciones dependen del tipo de microcontrolador usado.
drivers	Contiene funciones para controlar componentes externos como: Keypad, Lcd, etc. Estas funciones no estan compiladas e invocan funciones que dependen del tipo de microcontrolador usado
pic	Contiene definiciones básicas para cada tipo de PIC, como por ejemplo: están definidos a modo de estructuras todos los registros del PIC.
util	Carpeta con la definición de todas las funciones especiales como: "delay", o modulos virtuales. Estas funciones no dependen del tipo de PIC.
.	En la carpeta raiz se encuentra la biblioteca estandar de C.

1.4. Microcontroladores Soportados

Tabla 1-3. Microcontroladores Soportados Actualmente

uC	Disp	Driver	Pic	Util	Libc
12f635	-	-	-	-	-
16f610	-	-	-	-	-
16f616	-	-	-	-	-
16f627	-	-	-	OK	OK
16f627a	-	-	-	OK	OK
16f628	-	-	-	OK	OK
16f628a	-	-	-	OK	OK

uC	Disp	Driver	Pic	Util	Libc
16f630	-	-	-	-	-
16f631	-	-	-	-	-
16f636	-	-	-	OK	OK
16f639	-	-	-	OK	OK
16f648a	-	-	-	OK	OK
16f676	-	-	-	-	-
16f677	-	-	-	-	-
16f684	-	-	-	OK	OK
16f685	-	-	-	OK	OK
16f687	-	-	-	OK	OK
16f688	-	-	-	OK	OK
16f689	-	-	-	OK	OK
16f690	-	-	-	OK	OK
16f716	-	-	-	OK	OK
16f72	-	-	-	-	-
16f73	-	-	-	OK	OK
16f737	-	-	-	OK	OK
16f74	-	-	-	OK	OK
16f747	-	-	-	-	-
16f76	-	-	-	OK	OK
16f767	-	-	-	OK	OK
16f77	-	-	-	OK	OK
16f777	-	-	-	OK	OK
16f785	-	-	-	OK	OK
16f818	-	-	-	-	-
16f819	-	-	-	OK	OK
16f83	-	-	-	-	-
16f84	-	-	-	OK	OK
16f84a	-	-	-	OK	OK
16f87	-	-	-	OK	OK
16f870	-	-	-	-	-
16f871	-	-	-	-	-
16f872	-	-	-	-	-
16f873	-	-	-	-	-
16f873a	-	-	-	-	-
16f874	-	-	-	-	-
16f874a	-	-	-	-	-
16f876	-	-	-	OK	OK

uC	Disp	Driver	Pic	Util	Libc
16f876a	-	-	-	OK	OK
16f877	-	-	-	OK	OK
16f877a	OK	OK	OK	OK	OK
16f88	-	-	-	OK	OK
16f882	-	-	-	-	-
16f883	-	-	-	-	-
16f884	-	-	-	-	-
16f886	-	-	-	-	-
16f887	-	-	-	-	-
16f913	-	-	-	OK	OK
16f914	-	-	-	OK	OK
16f916	-	-	-	OK	OK
16f917	-	-	-	OK	OK
16f946	-	-	-	-	-
16c432	-	-	-	OK	OK
16c433	-	-	-	OK	OK
16c554	-	-	-	OK	OK
16c557	-	-	-	OK	OK
16c558	-	-	-	OK	OK
16c61	-	-	-	OK	OK
16c62	-	-	-	OK	OK
16c62a	-	-	-	OK	OK
16c62b	-	-	-	OK	OK
16c620	-	-	-	OK	OK
16c620a	-	-	-	OK	OK
16c621	-	-	-	OK	OK
16c621a	-	-	-	OK	OK
16c622	-	-	-	OK	OK
16c622a	-	-	-	OK	OK
16c63	-	-	-	OK	OK
16c63a	-	-	-	OK	OK
16c64	-	-	-	OK	OK
16c64a	-	-	-	OK	OK
16c641	-	-	-	-	-
16c642	-	-	-	OK	OK
16c65	-	-	-	OK	OK
16c65a	-	-	-	OK	OK
16c65b	-	-	-	OK	OK

uC	Disp	Driver	Pic	Util	Libc
16c66	-	-	-	OK	OK
16c661	-	-	-	-	-
16c662	-	-	-	OK	OK
16c67	-	-	-	OK	OK
16c71	-	-	-	-	-
16c710	-	-	-	-	-
16c711	-	-	-	OK	OK
16c712	-	-	-	OK	OK
16c715	-	-	-	OK	OK
16c716	-	-	-	OK	OK
16c717	-	-	-	OK	OK
16c72	-	-	-	OK	OK
16c72a	-	-	-	OK	OK
16c73	-	-	-	OK	OK
16c73a	-	-	-	OK	OK
16c73b	-	-	-	OK	OK
16c74	-	-	-	OK	OK
16c74a	-	-	-	OK	OK
16c74b	-	-	-	OK	OK
16c745	-	-	-	OK	OK
16c76	-	-	-	OK	OK
16c765	-	-	-	OK	OK
16c77	-	-	-	OK	OK
16c770	-	-	-	OK	OK
16c771	-	-	-	OK	OK
16c773	-	-	-	OK	OK
16c774	-	-	-	OK	OK
16c781	-	-	-	OK	OK
16c782	-	-	-	OK	OK
16c84	-	-	-	-	-
16c923	-	-	-	OK	OK
16c924	-	-	-	OK	OK
16c925	-	-	-	OK	OK
16c926	-	-	-	OK	OK
16cr62	-	-	-	OK	OK
16cr620a	-	-	-	-	-
16cr63	-	-	-	OK	OK
16cr64	-	-	-	OK	OK

uC	Disp	Driver	Pic	Util	Libc
16cr65	-	-	-	OK	OK
16cr72	-	-	-	OK	OK
16cr73	-	-	-	-	-
16cr74	-	-	-	-	-
16cr76	-	-	-	-	-
16cr77	-	-	-	-	-
16cr83	-	-	-	-	-
16cr84	-	-	-	OK	OK

1.5. Cabeceras Definidas

Tabla 1-4. Cabeceras definidas en PicGccLibrary

Carpeta	Cabeceras
Disp	pinio.h, uart.h, comp.h, adc.h, pwm.h, eeprom.h, i2c.h, timer0.h, timer2.h, interrupt.h, system.h
Driver	keypad4x4.h, keypad4x4flex.h
Util	delayms.h, memory.h, uart2.h
Libc	string.h, stdarg.h, stddef.h, limits.h, tipos.h

1.6. Biblioteca de Dispositivos Soportados

Tabla 1-5. Dispositivos Soportados Actualmente

uC	PINIO.H	UART.H	COMP.H	ADC.H	PWM.H	EEPROM.H	I2C.H	SPI.H	FLASH.H	TIMER0.H	TIMER1.H	TIMER2.H	INTERRUPT.H	SYSTEM.H
12f635	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f610	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f616	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f627	-	-	-	-	-	-	-	-	-	-	-	-	-	-

uC	PINIO.H	UART.H	COMP.H	ADC.H	PWM.H	EEPROM.H	I2C.H	SPI.H	FLASH.H	TIMER0.H	TIMER1.H	TIMER2.H	INTERRUPT.H	SYSTEM.H
16f627a		-	-	-	-	-	-	-	-	-	-	-	-	-
16f628	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f628a		-	-	-	-	-	-	-	-	-	-	-	-	-
16f630	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f631	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f636	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f639	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f648a		-	-	-	-	-	-	-	-	-	-	-	-	-
16f676	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f677	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f684	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f685	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f687	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f688	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f689	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f690	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f716	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f72	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f73	-	-	-	-	-	-	-	-	-	-	-	-	-	-

uC	PINIO.H	UART.H	COMP.H	ADC.H	PWM.H	EEPROM.H	I2C.H	SPI.H	FLASH.H	TIMER0.H	TIMER1.H	TIMER2.H	INTERRUPT.H	SYSTEM.H
16f737	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f74	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f747	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f76	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f767	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f77	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f777	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f785	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f818	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f819	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f83	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f84	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f84a	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f87	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f870	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f871	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f872	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f873	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f873a		-	-	-	-	-	-	-	-	-	-	-	-	-
16f874	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f874a		-	-	-	-	-	-	-	-	-	-	-	-	-

uC	PINIO.H	UART.H	COMP.H	ADC.H	PWM.H	EEPROM.H	I2C.H	SPI.H	FLASH.H	TIMER0.H	TIMER1.H	TIMER2.H	INTERRUPT.H	SYSTEM.H
16f876	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f876a		-	-	-	-	-	-	-	-	-	-	-	-	-
16f877	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f877a	OK	OK	OK	OK	OK	OK	OK	-	-	OK	-	OK	OK	OK
16f88	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f882	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f883	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f884	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f886	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f887	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f913	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f914	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f916	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f917	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16f946	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c432	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c433	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c554	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c557	-	-	-	-	-	-	-	-	-	-	-	-	-	-

uC	PINIO.H	UART.H	COMP.H	ADC.H	PWM.H	EEPROM.H	I2C.H	SPI.H	FLASH.H	TIMER0.H	TIMER1.H	TIMER2.H	INTERRUPT.H	SYSTEM.H
16c558	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c61	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c62	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c62a	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c62b	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c620	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c620a	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c621	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c621a	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c622	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c622a	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c63	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c63a	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c64	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c64a	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c641	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c642	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c65	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c65a	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c65b	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c66	-	-	-	-	-	-	-	-	-	-	-	-	-	-

uC	PINIO.H	UART.H	COMP.H	ADC.H	PWM.H	EEPROM.H	I2C.H	SPI.H	FLASH.H	TIMER0.H	TIMER1.H	TIMER2.H	INTERRUPT.H	SYSTEM.H
16c661	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c662	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c67	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c71	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c710	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c711	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c712	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c715	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c716	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c717	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c72	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c72a	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c73	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c73a	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c73b	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c74	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c74a	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c74b	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c745	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c76	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c765	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c77	-	-	-	-	-	-	-	-	-	-	-	-	-	-

uC	PINIO.H	UART.H	COMP.H	ADC.H	PWM.H	EEPROM.H	I2C.H	SPI.H	FLASH.H	TIMER0.H	TIMER1.H	TIMER2.H	INTERRUPT.H	SYSTEM.H
16c770	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c771	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c773	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c774	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c781	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c782	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c84	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c923	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c924	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c925	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16c926	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16cr62	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16cr620a		-	-	-	-	-	-	-	-	-	-	-	-	-
16cr63	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16cr64	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16cr65	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16cr72	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16cr73	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16cr74	-	-	-	-	-	-	-	-	-	-	-	-	-	-

uC	PINIO.H	UART.H	COMP.H	ADC.H	PWM.H	EEPROM.H	I2C.H	SPI.H	FLASH.H	TIMER0.H	TIMER1.H	TIMER2.H	INTERRUPT.H	SYSTEM.H
16cr76	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16cr77	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16cr83	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16cr84	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Capítulo 2. Biblioteca de Dispositivos

En todos los modulos aqui mostrados su utilizacion depende del tipo de PIC

2.1. Módulo EEPROM Interna

Muestra las funciones para poder trabajar con la EEPROM Interna de los PIC.

```
#include <eeprom.h>
```

2.1.1. eeprom_read

Lee la dirección **addr** en la EEprom interna del PIC

```
BYTE eeprom_read(BYTE addr);
```

2.1.2. eeprom_write

Escribe el **dato** en la dirección **addr** en la EEprom interna del PIC

```
void eeprom_write(BYTE dato,BYTE addr);
```

2.1.3. Ejemplo de Eeprom

para compilar necesitas los siguientes comandos

```
Ejemplo de compilacion : ./compila.sh ej_eeprom 16f877a
```

ej_eeprom.c

```
#include <pic/p16f877a.h>

#define FOSC_HZ 20000000
```

```
#include <uart.h>
#include <delayms.h>
#include <eeprom.h>

int main (void)
{
unsigned char tmp,i;

delayms(500);

uart_open(SET_9600_8N1);
uart_putc('\n');

uart_puts("Ultimos 4 Datos:");

uart_putc(eeprom_read(0));
uart_putc(eeprom_read(1));
uart_putc(eeprom_read(2));
uart_putc(eeprom_read(3));
uart_putc('\n');

i=0;
while((i<4)&&(tmp!=13))
{
uart_putc('>');
tmp = uart_getc();
uart_putc(tmp);
uart_putc('\n');
eeprom_write(tmp,i);
i++;
}
uart_puts("FIN\n");
return 0;
}
```

2.2. Módulo PWM

Muestra las funciones para poder trabajar con el módulo PWM Interno de los PIC.Todas las ecuaciones estan en segundos.Tosc es la inversa de la frecuencia del cristal FOSC_HZ.

```
Periodo=(Pr2+1)*4*Tosc*PreescalerTMR2
TiempoPulsoAlto=CCPR1L_CCP1CON54*Tosc*PreEscalerTmr2
```

```
#include <pwm.h>
```

```
#define TMR2_PREESCALER_1  0
#define TMR2_PREESCALER_4  1
#define TMR2_PREESCALER_16 2
```

2.2.1. pwm1_set_data

Configura el módulo PWM1 del PIC, es necesario indicar el Preescaler y el CCPR1L_CCP1CON54

```
void pwm1_set_data( BYTE Pr2,BYTE PreescalerTMR2,int16 CCPR1L_CCP1CON54);
```

2.2.2. pwm2_set_data

Configura el módulo PWM2 del PIC, es necesario indicar el Preescaler y el CCPR2L_CCP2CON54

```
void pwm2_set_data( BYTE Pr2,BYTE PreescalerTMR2,int16 CCPR2L_CCP2CON54);
```

2.2.3. Ejemplo de PWM

para compilar necesitas los siguientes comandos

```
Ejemplo de compilacion : ./compila.sh ej_pwm 16f877a
```

ej_pwm.c

```
#include <pic/p16f877a.h>

#define FOSC_HZ 20000000

#include <delayms.h>
#include <pwm.h>

int main (void)
{
delayms(100);

pwm1_set_data(224,TMR2_PREESCALER_1,301);
pwm2_set_data(224,TMR2_PREESCALER_1,301);

while(TRUE);
return 0;
```

```
}
```

2.3. Módulo UART

Muestra las funciones para poder trabajar con el módulo UART Interno de los PIC.Antes de usar este módulos se debe definir la frecuencia usar con el comando.

```
#define FOSC_HZ 20000000
```

Se han realizado algunas definiciones como:

```
#include <uart.h>

SET_1200_8N1
SET_2400_8N1
SET_4800_8N1
SET_9600_8N1
SET_19200_8N1
SET_38400_8N1
SET_57600_8N1
SET_115200_8N1
```

Estas definiciones pueden ser usandas en la función uart_open.

2.3.1. uart_open

Configura el puerto Serie Asincrono

```
void uart_open( BYTE STATUS_SPBRG, BYTE STATUS_SPEED);

ejemplo:
uart_open(SET_9600_8N1);
```

2.3.2. uart_kbhit

Devuelve **1** si hay un byte en el buffer de entrada del puerto Serie Asincrono

```
BYTE uart_kbhit(void);
```

2.3.3. uart_getc

Lee un byte en el buffer de entrada del puerto Serie Asíncrono, si no hay ninguno espera a que exista uno, la función es bloqueante

```
char uart_getc(void);
```

2.3.4. uart_putc

Escribe un **dato** en el puerto Serie Asíncrono

```
void uart_putc(char dato);
```

2.3.5. uart_puts

Escribe un cadena de **datos** en el puerto Serie Asíncrono

```
void uart_puts(char *datos);
```

2.3.6. uart_close

Cierra el puerto Serie Asíncrono

```
void uart_close(void);
```

2.3.7. Ejemplo de UART

para compilar necesitas los siguientes comandos

```
Ejemplo de compilacion : ./compila.sh ej_uart 16f877a
```

ej_uart.c

```
#include <pic/p16f877a.h>

#define FOSC_HZ 20000000
```

```
#include <uart.h>
#include <delayms.h>

int main (void)
{
char c=0;
char INTRO[6]="HOLA\n";
char FIN[6]="FIN\n";

delayms(100);

uart_open(SET_9600_8N1);

uart_set_baudrate(19200);

uart_puts(INTRO);
while(c!=13)
{
if(uart_kbhit()==1)
{
c=uart_getc();
uart_putc(c);
}
}
uart_puts(FIN);
uart_close();

return 0;
}
```

2.4. Módulo IO y manipulación de pines

Muestra las funciones para poder trabajar con los pines del PIC.

```
#include <pinio.h>
```

2.4.1. output_high_slow

Establece un PIN a un estado alto de voltaje. Verifica el estado del **tris**.

```
void output_high_slow( BYTE puerto,
BYTE pin);
```

```
ejemplo:
output_high_slow(PIN_B0);
```

2.4.2. output_high_fast

Establece un PIN a un estado alto de voltaje. No verifica el estado del **tris**.

```
void output_high_fast( BYTE puerto,
BYTE pin);
ejemplo:
output_high_fast(PIN_B0);
```

2.4.3. output_high

Establece un PIN a un estado alto de voltaje. La verificación del estado del **tris** depende de habilitar el macro SLOW_IO como TRUE o FALSE.

Por defecto, si no se escribe **#define SLOW_IO TRUE** , SLOW_IO tiene el valor de TRUE.

```
#define SLOW_IO TRUE

void output_high( BYTE puerto,
BYTE pin);
ejemplo:
output_high(PIN_B0);
```

2.4.4. output_low_slow

Establece un PIN a un estado bajo de voltaje. Verifica el estado del **tris**.

```
void output_low_slow( BYTE puerto,
BYTE pin);
ejemplo:
output_low_slow(PIN_B0);
```

2.4.5. output_low_fast

Establece un PIN a un estado bajo de voltaje. No verifica el estado del **tris**.

```
void output_low_fast( BYTE puerto,
BYTE pin);
ejemplo:
output_low_fast(PIN_B0);
```

2.4.6. output_low

Establece un PIN a un estado bajo de voltaje. La verificación del estado del **tris** depende de habilitar el macro SLOW_IO como TRUE o FALSE.

Por defecto, si no se escribe **#define SLOW_IO TRUE** , SLOW_IO tiene el valor de TRUE.

```
#define SLOW_IO TRUE

void output_low( BYTE puerto,
BYTE pin);
ejemplo:
output_low(PIN_B0);
```

2.4.7. input_slow

Devuelve el estado de un pin. puede ser **1** o **0**. Verifica el estado del **tris**.

```
BYTE input_slow( BYTE puerto,
BYTE pin);
ejemplo:
input_slow(PIN_B1);
```

2.4.8. input_fast

Devuelve el estado de un pin. puede ser **1** o **0**. No verifica el estado del **tris**.

```
BYTE input_fast( BYTE puerto,
BYTE pin);
ejemplo:
input_fast(PIN_B1);
```

2.4.9. input

Devuelve el estado de un pin. puede ser **1** o **0**. La verificación del estado del **tris** depende de habilitar el macro SLOW_IO como TRUE o FALSE.

Por defecto, si no se escribe **#define SLOW_IO TRUE** , SLOW_IO tiene el valor de TRUE.

```
#define SLOW_IO TRUE

BYTE input( BYTE puerto,
BYTE pin);
ejemplo:
input(PIN_B1);
```

2.4.10. set_pin_to_slow

Establece el estado de un pin a **1** o **0**. Verifica el estado del **tris**.

```
void set_pin_to_slow( BYTE puerto,
BYTE pin,
BYTE X);

ejemplo:
set_pin_to_slow(PIN_C1,1);//el pin C1 ahora tiene un estado alto
set_pin_to_slow(PIN_C1,0);//el pin C1 ahora tiene un estado bajo
```

2.4.11. set_pin_to_fast

Establece el estado de un pin a **1** o **0**. No verifica el estado del **tris**.

```
void set_pin_to_fast( BYTE puerto,
BYTE pin,
BYTE X);

ejemplo:
set_pin_to_fast(PIN_C1,1);//el pin C1 ahora tiene un estado alto
set_pin_to_fast(PIN_C1,0);//el pin C1 ahora tiene un estado bajo
```

2.4.12. set_pin_to

Establece el estado de un pin a **1** o **0**.

La verificación del estado del **tris** depende de habilitar el macro SLOW_IO como TRUE o FALSE.

Por defecto, si no se escribe **#define SLOW_IO TRUE** , SLOW_IO tiene el valor de TRUE.

```
#define SLOW_IO TRUE

void set_pin_to( BYTE puerto,
BYTE pin,
BYTE X);

ejemplo:
set_pin_to(PIN_C1,1);//el pin C1 ahora tiene un estado alto
set_pin_to(PIN_C1,0);//el pin C1 ahora tiene un estado bajo
```

2.4.13. set_pin_tris_to

Establece el TRIS de un pin a **1** o **0**.

```
void set_pin_tris_to( BYTE puerto,
BYTE pin,
BYTE X);

ejemplo:
set_pin_tris_to(PIN_C1,1);//el TRIS C1 ahora tiene un estado alto
set_pin_tris_to(PIN_C1,0);//el TRIS C1 ahora tiene un estado bajo
```

2.4.14. set_porta_as_digital

Establece el puerto A como DIGITAL.

```
#define set_porta_as_digital() ADCON1=0x06

ejemplo:
set_porta_as_digital();
```

2.4.15. set_pullup_portb

Habilita o deshabilita las resistencias de pull-up del puerto B.

Para habilitar las resistencias de pull-up X= 1 o TRUE.

Para deshabilitar las resistencias de pull-up X= 0 o FALSE.

```
void set_pullup_portb(BYTE X);
```

2.4.16. Ejemplo de PinIO

Para compilar necesitas los siguientes comandos.

```
Ejemplo de compilacion : ./compila.sh ej_pinio 16f877a
```

ej_pinio.c

```
#include <pic/p16f877a.h>

#define FOSC_HZ 20000000

#include <pinio.h>
#include <delayms.h>

int main (void)
{
delayms(100);

while(TRUE)
{
delayms(50);
output_high(PIN_C2);
delayms(50);
output_low(PIN_C2);

if(input(PIN_A4)==1) output_high(PIN_B7);
else                 output_low (PIN_B7);
}
return 0;
}
```

2.5. Módulo I2C

Autor : Pierre Launay

per.launay en free.fr

El bus I2C es un bus serie sincrono desarollado por Philips

Descripción del bus I2C aqui (http://es.wikipedia.org/wiki/I2C)

Entender el bus I2C aqui (http://robots-argentina.com.ar/Comunicacion_busI2C.htm)

Aqui se muestran las funciones para poder trabajar con el módulo I2C Maestro Interno de los PIC.

Antes de usar este módulos se debe definir la frecuencia usada con el comando.

```
#define FOSC_HZ 4000000
#include <i2c.h>
```

Se han realizado algunas definiciones como:

```
#define I2C_400K i2c_find_frec(400000) //frecuencia de 400Khz
#define I2C_100K i2c_find_frec(100000) //frecuencia de 100Khz
#define I2C_40K i2c_find_frec(40000) //frecuencia de 40Khz
#define I2C_10K i2c_find_frec(10000) //frecuencia de 10Khz
#define I2C_4K i2c_find_frec(4000) //frecuencia de 4Khz
#define I2C_1K i2c_find_frec(1000) //frecuencia de 1Khz
```

Estas definiciones pueden ser usandas en la función init_i2c

2.5.1. i2c_init

Configura el puerto Serie sincrono

```
void i2c_init(unsigned char SPEED_I2C);
```

SPEED_I2C es el registro SSPADD, reloj_I2C=FOSC_HZ/(4*(SSPADD+1))

2.5.2. i2c_wait_for_idle

Espera a que el bus esté libre

```
void i2c_wait_for_idle(void);
```

2.5.3. i2c_start

principio de trama, secuencia de inicio

```
void i2c_start(void);
```

2.5.4. i2c_restart

Devolver una trama, fin de trama y principio de trama permite poner dos tramas para leer . Ver i2c_read()

Ejemplo para leer las segundas en una trama :

Principio de trama (start), escribir la dirección del circuito I2C, escribir el registro de las segundas,

Devolver una trama (restart), escribir la dirección del circuito I2C, leer el registro de las segundas.

fin de trama (stop)

```
void i2c_restart(void);
```

2.5.5. i2c_stop

fin de trama, secuencia de parada

```
void i2c_stop(void);
```

2.5.6. i2c_delay

Tiempo de espera entre fin de trama y principio de trama >=4.7 us (Frecuencia Reloj SCL = 100KHz)

```
void i2c_stop(void);
```

2.5.7. i2c_ack_read

recepción del pulso de reconocimiento

```
unsigned char i2c_ack_read (void);
```

"0" buena recepción

"1" mala recepción

2.5.8. i2c_ack_write

emisión. del pulso de reconocimiento

```
void i2c_ack_write(unsigned char ack);
```

"0" la trama puede continuar

"1" la trama se para

2.5.9. i2c_write

el maestro escribe en el esclavo el caracter c, el caracter c puede ser una dirección, una registro interno, un dato

```
void i2c_write(unsigned char c);
```

2.5.10. i2c_write_with_ack

El maestro escribe el caracter c en el esclavo y remite el reconocimiento (ack).

Esta rutina junta i2c_write(c) y reception_ack()

```
unsigned char i2c_write_with_ack(unsigned char c) ;
```

2.5.11. i2c_read

El maestro lee el caracter enviado por el esclavo

```
unsigned char i2c_read(void);
```

2.5.12. i2c_read_with_ack

el maestro lee el caracter enviado por el esclavo y envia el reconocimiento (ack).

```
unsigned char i2c_read_with_ack(unsigned char ack);
```

2.5.13. Ejemplo de Módulo I2C

para compilar necesitas los siguientes comandos

```
Ejemplo de compilacion : ./compila.sh ej_i2c 16f877a
```

ej_i2c.c

```
int main(void)
{
return 0;
}
```

2.6. Módulo ADC

Autor: Santiago Gonzalez

Correo: santigoro en gmail.com

Entender la conversión Analógico-Digital: aqui (http://es.wikipedia.org/wiki/Conversion_analogica-digital)

Aquí se muestran las funciones para trabajar con el modulo ADC 10 bits Interno de los PIC16F87X.

2.6.1. adc_init

Inicia el módulo ADC con la confiración establecida en los parametros de entrada: frecuencia de oscilador y configuración de entradas. El módulo ADC quedará configurado y listo para usar, pero será activado hasta que se habra un canal para evitar consumo innecesario de corriente.

```
void adc_init(unsigned char set_fosc, unsigned char set_channel);
```

set_fosc Establece la frecuencia de adc.

Las posibles opcines son:

```
FRECUENCIA DE OSCILADOR ADC:


FOSC_DIV_2 // Frecuencia de oscilador del PIC / 2
FOSC_DIV_4
FOSC_DIV_8
FOSC_DIV_16
FOSC_DIV_32
FOSC_DIV_64
FOSC_RC // Oscilador interno del módulo ADC
```

set_channel Define que pines se usarán como entradas analógicas o como Vref externa. Cuando solo se define un pin como voltaje de referencia, este será Vref+, referencia positiva y estará en RA3, cuando se definen dos entradas como Vref la positiva será RA3 y la negativa RA2.

Las posibles opcines son:

```
CONFIGURACION DE ENTRADAS Y Vref(PCFG en ADCON1)

A8_R0 // 8 entradas Analógicas y 0 como Vref (Vref+ = Vdd , Vref- = Ground)
A7_R1 // 7 entradas analógicas y Vref+ en AN3
A5_R0
A4_R1
A3_R0
A2_R1
A0_R0
A6_R2 // 6 entradas Analógicas, Vref+ en AN3 y Vref- en AN2
A6_R0
A5_R1
A4_R2
A3_R2
A2_R2
A1_R0
A1_R2
```

Consultar el Datasheet del pic utilizado para ver canales diponibles, no se deben inicializar canales no implementados en el modelo de pic a usar, por ejemplo, el pic16f876a solo tiene 5 canales ADC, nunca usar A6_R0 en este pic.

2.6.2. adc_set_channel

Establece configuración de pines de entrada ADC y Vref

```
void adc_set_channel(unsigned char set_channel);
```

set_channel Define que pines se usarán como entradas analógicas o como Vref externa. Cuando solo se define un pin como voltaje de referencia, este será Vref+, referencia positiva y estará en RA3, cuando se definen dos entradas como Vref la positiva será RA3 y la negativa RA2.

Las posibles opcines son las mismas que en adc_init: A2_R1 , etc.

2.6.3. adc_open

Abre canal seleccionado y activa módulo ADC

```
void adc_open(unsigned char channel);
```

channel Establece el canal ADC a leer

Las posibles opciones son:

```
CHANNEL_0
CHANNEL_1
CHANNEL_2
CHANNEL_3
CHANNEL_4
CHANNEL_5
CHANNEL_6
CHANNEL_7
```

2.6.4. adc_read

Lee un canal previamente abierto

```
unsigned int adc_read(void);
```

2.6.5. adc_close

Cierra módulo ADC.

```
void adc_close(void);
```

La configuración queda tal como se inicializó la última vez, se puede volver a activar el módulo ADC mediante adc_open(channel), sin ejecutar adc_init()

2.6.6. Ejemplo de Módulo ADC

para compilar necesitas los siguientes comandos

```
Ejemplo de compilacion : ./compila.sh ej_adc_87Xa.c 16f877a
```

ej_adc_87Xa.c

```
/*
Ejemplo de utilización de las funciones del módulo ADC.
válido para la familia 16f87Xa,
Se utiliza el canal 0,
conectar los voltajes a convertir a RA0,
conectar un led u otra salida a RB7.
RB7 se pondrá en estado alto (Vdd) cuando la tensión en RA0 sea mayor que 1/2 Vdd.
*/
#include <pic/p16f877a.h>
#include <adc.h>

int main(void)
{
TRISAbits.TRISA0 = 1; // A0 como entrada
TRISBbits.TRISB7 = 0; // B7 como salida

adc_init( FOSC_DIV_32, A1_R0); // Inicializa módulo ADC
adc_open(CHANNEL_0); // Abre canal 0 ADC

while (1)
{
if (adc_read() > 512 )
PORTBbits.RB7 = 1; // Enciende led
else
```

```
PORTBbits.RB7 = 0; // Apaga led
}
}
```

2.7. Módulo Comparadores.

Autor: Santiago Gonzalez

Correo: santigoro en gmail.com

Aquí se muestran las funciones para trabajar con los dos comparadores internos de los PIC16F87X.

2.7.1. comp_init

Inicia el módulo comparadores

```
void comp_init(unsigned char set_config));
```

set_config Define entradas analógicas y Vref,

Las posibles opciones son:

```
 COMP_RESET
 COMP1_OUT              // Comparador 1 con salida por RA4
 TWO_COMP               // Dos comparadores independientes
 TWO_COMP_OUT           // Dos comparadores independientes, salidas por RA4 y RA5
 TWO_COMP_COMREF        // Dos comparadores con entrada común: RA3
 TWO_COMP_COMREF_OUT    // Dos comparadores, entrada común RA3, salidas RA4 RA5
     TWO_COMP_MULTI4_VREF   // Dos comparadores con Vref interna, 4 entradas multiplexadas
 COMP_OFF
```

2.7.2. comp_set_multiplex

Establece que entradas son conectadas en el modo multiplexado

```
comp_set_multiplex(unsigned char set_switch);
```

set_switch determina que canales son multiplexados.

Las posibles opciones son:

```
RA0_RA1
RA2_RA3
```

2.7.3. comp_set_vref

Esta función cambia el valor de Vref inerna, si previamente ha sido iniciado el uso de Vref interna con: comp_vref_mode().

```
void comp_set_vref(unsigned char set_vref);
```

set_vref determina el valor de Vref interna, en porcentaje de Vpp, por ejemplo: comp_Vref_mode(RA0_RA1, VREF_65) activa las entradas RA0 y RA1 con Vref interna común al 65% de Vpp. Son vÃ¡lidos valores entre VREF_0 y VREF_75 , aunque en la prÃ¡ctica no se obtendrÃ¡n valores superiores a un 70% de Vdd y este mÃ³dulo solo es capaz de generar 30 valores distintos de Vref, asÃ que el porcentaje seleccionado se redondea al valor mÃ¡s prÃ³ximo. La precisiÃ³n es menor por debajo del 25% de Vdd. Para mÃ¡s detalles consultar el datasheet del pic utilizado.

Para comprobar el funcionamiento y exactitud de Vref interna se puede activar el bit 6 CVROE de CVRCON, quedando conectada Vref a RA2:

```
  CVRCONbits.CVROE = 1;  //ahora Vref estÃ¡ presente en RA2
```

Esta salida se puede usar como fuente variable de voltaje (D-A), aunque de alta impedancia, para conseguir mayor capacidad de corriente se puede usar un amplificador operacional como segidor de voltaje.

2.7.4. comp_inv

Invierte o no las salidas de los comparadores

```
void comp_inv(unsigned char set_inv);
```

set_inv define si las salidas de comparadores se invierten o no

Las posibles opciones son:

```
NO_INV
```

```
C1_INV
C2_INV
C1_C2_INV
```

2.7.5. comp1_read

Esta función lee la salida del comparador 1, los posibles valores son 0 o 1.

```
void comp1_read();
```

2.7.6. comp2_read

Esta función lee la salida del comparador 2, los posibles valores son 0 o 1.

```
void comp2_read();
```

2.7.7. comp_reset

Esta función resetea el módulo Comparadores,

el módulo queda activo pero la lecturá será 0.

```
void comp_reset(void);
```

2.7.8. comp_off

Esta función apaga el módulo Comparadores, evitando el consumo innecesario de corriente.

```
void comp_off(void);
```

2.7.9. Ejemplo de Módulo Comparadores

para compilar necesitas los siguientes comandos

```
Ejemplo de compilacion : ./compila.sh ej_comp_87Xa.c 16f877a
```

ej_comp_87Xa.c

```
/*
 *  Ejemplo de utilización de las funciones del módulo comparadores.
 *  válido para la familia 16f87Xa,

 *  En este ejemplo se utiliza el comparador 1 con Vref interna al 50% de Vdd,

 *  conectar voltaje a comparar a RA0,

 *  conectar un led u otra salida a RB7.

 *  RB7 se pondrá en estado alto (Vdd) cuando la tensión en RA0 sea menor que el 50% de Vdd.
*/

#include <pic/p16f877a.h>
#include <comp.h>

int main(void)

{

    TRISAbits.TRISA0 = 1;           // A0 como entrada

    TRISAbits.TRISA3 = 1;           // A3 como entrada

    TRISBbits.TRISB7 = 0;           // B7 como salida



    comp_init( TWO_COMP_MULTI4_VREF ); // Inicia comparadores en modo multiplexado con Vref interna


    comp_set_vref( VREF_50 );         // Establece Vref en 50% Vdd


    while (1)

    {

        if ( comp1_read() == 1 )    // lee comparador 1  = comprueba bit C1OUT de registro CMCON

            PORTBbits.RB7 = 1;      // Enciende led
        else
            PORTBbits.RB7 = 0;      // Apaga led

    }
```

```
}
```

2.8. Módulo Interrupción

Este módulo genera las funciones necesarias para trabajar con interrupciones.

```
#include <interrupt.h>
```

2.8.1. _ISR_MAIN

_ISR_MAIN Define la función interrupción principal, esta función se ejecutará cuando acurra cualquier interrupción.

Puede escoger cualquier nombre para la función interrupción principal.se utiliza:

```
#include <interrupt.h>

_ISR_MAIN void nombre_de_la_funcion(void);

//Esta función se ejecutará cuando ocurra cualquier interrupción.
//Dentro de su código se deberá verivicar los FLAG, para saber
//que interrupción ocurrió.
void nombre_de_la_funcion(void)
{
//código genérico
//cuando termines debes limpiar el FLAG de la interrupción que usaste
}

int main(void)
{
//En la función principal se deberá habilitar la interrupción GLOBAL
}
```

2.8.2. ISR_MAIN

ISR_MAIN Define la función/código interrupción principal, esta función/código se ejecutará cuando acurra cualquier interrupción.

Puede escoger cualquier nombre para la función/código interrupción principal. se utiliza:

```
#include <interrupt.h>

//Esta función/código se ejecutará cuando acurra cualquier interrupción.
//Dentro de su código se deberá verivicar los FLAG, para saber
//que interrupción ocurrió.
ISR_MAIN(nombre_de_la_funcion)
{
//código genérico
//cuando termines debes limpiar el FLAG de la interrupción que usaste
}

int main(void)
{
//En la función principal se deberá habilitar la interrupción GLOBAL
}
```

2.8.3. enable_int_global

Habilita globalmente todas las interrupciones.

```
void enable_int_global(void);

ejemplo:
enable_int_global();
```

2.8.4. disable_int_global

Deshabilita globalmente todas las interrupciones.

```
void disable_int_global(void);

ejemplo:
disable_int_global();
```

2.8.5. Interrupción Externa

Las funciones para trabajar con la interrupción externa (RB0) son:

2.8.5.1. enable_int_ext

Habilita la interrupción externa

```
#define  H_TO_L    0
#define  L_TO_H    1

void enable_int_ext(BYTE flanco);

ejemplo:
enable_int_global(L_TO_H);
```

H_TO_L indica una interrupción activada por el flanco de bajada

L_TO_H indica una interrupción activada por el flanco de subida

2.8.5.2. disable_int_ext

Deshabilita la interrupción externa

```
void disable_int_ext(void);
```

Al deshabilitar la interrupción no modifica el habilitador de interrupción global

2.8.5.3. int_ext_flag

Devuelve el estado actual del flag indicador de interrupción externa

```
BYTE int_ext_flag(void);
```

El uso de esta función no modifica el flag

2.8.5.4. int_ext_free_flag

Limpia el flag indicador de interrupción externa

```
void int_ext_free_flag(void);
```

El uso de esta función modifica el flag, poniendolo a 0

2.8.6. Interrupción TIMER0

Las funciones para trabajar con la interrupción por desbordamiento del TIMER0 son:

2.8.6.1. enable_int_timer0

Habilita la interrupcion por desbordamiento del TIMER0

```
void enable_int_timer0(void);
```

Para que esta interrupción se habilite tambien deberá activarse la interrupción global

2.8.6.2. disable_int_timer0

Deshabilita la interrupción por desbordamiento del TIMER0

```
void disable_int_timer0(void);
```

Al deshabilitar la interrupción no modifica el habilitador de interrupción global

2.8.6.3. int_timer0_flag

Devuelve el estado actual del flag indicador de interrupción por desbordamiento de TIMER0

```
BYTE int_timer0_flag(void);
```

El uso de esta función no modifica el flag

2.8.6.4. int_timer0_free_flag

Limpia el flag indicador de interrupción por desbordamiento del TIMER0

```
void int_timer0_free_flag(void);
```

El uso de esta función modifica el flag, poniendolo a 0

2.8.7. Interrupción RB4 to RB7

Las funciones para trabajar con la interrupción por cambio de estado de los pines RB4 a RB7 son:

2.8.7.1. enable_int_rb4to7

Habilita la interrupcion por cambio de estado de los pines de Rb4 a RB7

```
void enable_int_rb4to7(void);
```

Para que esta interrupción se habilite tambien deberá activarse la interrupción global

2.8.7.2. disable_int_rb4to7

Deshabilita la interrupción por cambio de estado de los pines de RB4 a RB7

```
void disable_int_rb4to7(void);
```

Al deshabilitar la interrupción no modifica el habilitador de interrupción global

2.8.7.3. int_rb4to7_flag

Devuelve el estado actual del flag indicador de interrupción por cambio de estado de los pines de RB4 a RB7

```
BYTE int_rb4to7_flag(void);
```

El uso de esta función no modifica el flag

2.8.7.4. int_rb4to7_free_flag

Limpia el flag indicador de interrupción por cambio de estado de los pines de RB4 a RB7

```
void int_rb4to7_free_flag(void);
```

El uso de esta función modifica el flag, poniendolo a 0

2.8.8. Interrupción Rx UART

Las funciones para trabajar con la interrupción por recepcion de caracter en el UART son:

2.8.8.1. enable_int_uart

Habilita la interrupción por recepcion de caracter en el UART

```
void enable_int_uart(void);
```

Para que esta interrupción se habilite tambien deberá activarse la interrupción global

2.8.8.2. disable_int_uart

Deshabilita la interrupción por recepcion de caracter en el UART

```
void disable_int_uart(void);
```

Al deshabilitar la interrupción no modifica el habilitador de interrupción global

2.8.8.3. int_uart_flag

Devuelve el estado actual del flag indicador de interrupción por recepcion de caracter en el UART

```
BYTE int_uart_flag(void);
```

El uso de esta función no modifica el flag, es mas no puede.

2.8.8.4. int_uart_free_flag

esta función no existe

El flag no se limpiará hasta leer con la función uart_getc el caracter entrante, esta función tiene que colocarse dentro de la función interrupción pues de lo contrario el flag nunca se limpiará y nunca saldras de la función interrupción, pues esta se invocará hasta el fin de los tiempos.

2.8.9. Interrupción ADC

Las funciones para trabajar con la interrupción por finalización de la conversión analógica a digital son:

2.8.9.1. enable_int_adc

Habilita la interrupción por finalización de la conversión analogica a digital

```
void enable_int_adc(void);
```

Para que esta interrupción se habilite tambien deberá activarse la interrupción global

2.8.9.2. disable_int_adc

Deshabilita la interrupción por finalizacion de la conversión analógica a digital

```
void disable_int_adc(void);
```

Al deshabilitar la interrupción no modifica el habilitador de interrupción global

2.8.9.3. int_adc_flag

Devuelve el estado actual del flag indicador de interrupción por rfinalización de la conversión analógica a digital

```
BYTE int_adc_flag(void);
```

El uso de esta función no modifica el flag.

2.8.10. Interrupción Comparador

Las funciones para trabajar con la interrupción por finalización de comparación son:

2.8.10.1. enable_int_comp

Habilita la interrupción por finalización de la comparación

```
void enable_int_comp(void);
```

Para que esta interrupción se habilite tambien deberá activarse la interrupción global

2.8.10.2. disable_int_comp

Deshabilita la interrupción por finalización de la comparación

```
void disable_int_comp(void);
```

Al deshabilitar la interrupción no modifica el habilitador de interrupción global

2.8.10.3. int_comp_flag

Devuelve el estado actual del flag indicador de interrupción por r finalización de la comparación

```
BYTE int_comp_flag(void);
```

El uso de esta función no modifica el flag.

2.8.11. Ejemplo de Módulo Interrupcion Externa

Para compilar necesitas los siguientes comandos

```
Ejemplo de compilacion : ./compila.sh ej_intext 16f877a
```

ej_intext.c

```
#include <pic/p16f877a.h>

#define FOSC_HZ 20000000

#include <pinio.h>
#include <delayms.h>
#include <interrupt.h>

BYTE bandera=0;

void funcion_con_mucho_codigo(void)
{
//codigo
bandera=0;
}

ISR_MAIN(funcion_interrupcion_global)
```

```
{
if(int_ext_flag()==TRUE)
{
bandera=1;
int_ext_free_flag();
}
}

int main (void)
{
delayms(100);

enable_int_ext(L_TO_H);
enable_int_global();

set_pullup_portb(TRUE);

while(TRUE)
{
if(bandera==1) funcion_con_mucho_codigo();
}

disable_int_ext();
return 0;
}
```

2.9. Módulo TIMER0

Autor: Santiago Gonzalez

Correo: santigoro en gmail.com

Aqui se muestran las funciones para trabajar con el TIMER0 de los PIC16F87X.

2.9.1. timer0_init

Inicializa TIMER0 con reloj interno (modo timer) o externo (modo counter).

```
void timer0_init(unsigned char intern_extern);
```

intern_extern Establece si se usa reloj interno o reloj/estímulo externo (modo counter).

```
TIMER_INTERN :Usa reloj interno.
COUNTER_EXTERN :Usa reloj/estímulo externo (modo counter)
```

```
timer0_init(TIMER_INTERN);
```

Pone a 0 la cuenta del timer0 y borra flag de interrupciones. El contador se incrementa en cada ciclo de intrucciones (Freq.Osc. / 4) si no se usa el prescaler.

2.9.2. timer0_set_edge

Establece si el incremento de contador se produce en el flanco de subida o en el flanco de bajada (modo counter).

```
void timer0_set_edge(unsigned char set_risfal);
```

Pone a 0 la cuenta del timer0 y borra flag de interrupciones El contador se incrementa en cada subida o bajada en el pin RA4 si no se usa prescaler.

set_risfal Establece si el incremento de contador se produce en el flanco de subida o en el flanco de bajada.

Las posibles opciones son:

```
 RIS_EDGE
 FAL_EDGE
```

Donde RIS_EDGE establece incremento de contador en flanco de subida (rising) y FAL_EDGE en flanco de bajada (falling).

```
timer0_set_edge(FAL_EDGE);
```

2.9.3. timer0_set_prescaler

Asigna prescaler a timer0 (esto anula el uso de prescaler en watchdog) y establece el valor del prescaler.

```
void timer0_set_prescaler(unsigned char set_presc);
```

set_presc Establece el valor del prescaler.

Las posibles opciones son:

```
// Frecuencia de ciclo de instrucciones
PRESC_DIV_2 // (FOSC/4) del PIC dividido entre 2
PRESC_DIV_4 // (FOSC/4) del PIC dividido entre 4
```

```
PRESC_DIV_8
PRESC_DIV_16
PRESC_DIV_32
PRESC_DIV_64
PRESC_DIV_128
PRESC_DIV_256
PRESC_OFF // asignado a wathdog
```

Cualquier operación de escritura en el registro TMR0 automaticamente pone a 0 el contador del prescaler, aunque mantiene su configuración.

2.9.4. timer0_write

Establece (escribe a TMR0) el valor del contador TMR0.

```
void timer0_write(unsigned char set_count);
```

set_count Establece el valor del contador TMR0.

2.9.5. timer0_read

Lee el valor del contador TMR0.

```
#define timer0_read() TMR0
```

2.9.6. Ejemplo Módulo TIMER0

para compilar necesitas los siguientes comandos

```
Ejemplo de compilacion : ./compila.sh ej_timer0.c 16f877a
```

ej_timer0.c

```
/*
Ejemplo de utilización de las funciones del módulo TIMER0.
válido para la familia 16f87Xa,
En este ejemplo se utiliza el timer0 como contador,
conectar un pulsador entre RA4 y 0V
y una resistencia pull-up de unos 5 Kohm entre RA4 y Vdd (+5v),
conectar un led u otra salida a RB7.
```

```
RB7 se pondrá en estado alto (Vdd) cuando se pulse 4 veces.
*/

#include <pic/p16f877a.h>
#include <timer0.h>

int main(void)
{
TRISAbits.TRISA4 = 1;
TRISBbits.TRISB7 = 0;
PORTBbits.RB7 = 0;

timer0_init(COUNTER_EXTERN); // Inicia timer0 modo contador (reloj o
// estímulo externo en RA4)

timer0_set_edge(FAL_EDGE); // Establece incremento de contador en flanco de bajada

timer0_set_prescaler(PRESC_DIV_2); // Establece prescaler en frecuencia
// de instrucciones / 2

while (timer0_read() < 2); // Espera aqui mientras la cuenta del timer < 2

PORTBbits.RB7 = 1;
}
```

2.10. Módulo timer2

Aqui se muestran las funciones para trabajar con el timer2 de los PIC16F87X.

2.10.1. timer2_init

Inicializa timer2.

```
void timer2_init(void);
```

Pone a 0 la cuenta del timer2 y borra flag de interrupciones. El contador se incrementa en cada ciclo de intrucciones (Freq.Osc. / 4) si no se usa el prescaler.

2.10.2. timer2_set_period

Establece periodo de timer2.

```
void timer2_set_period(unsigned char set_period);
```

set_period Establece periodo de timer2, debe ser un valor entre 0 y 255.

2.10.3. timer2_set_prescaler

Establece el valor del prescaler del timer2.

```
void timer2_set_prescaler(unsigned char set_presc);
```

set_presc Establece el valor del prescaler.

Las posibles opciones son:

```
PRESC_DIV_1
PRESC_DIV_4 // Frecuencia de ciclo de instrucciones (FOSC/4) del PIC / 4
PRESC_DIV_16
```

2.10.4. timer2_set_postscaler

Establece el valor del postscaler del timer2.

```
void timer2_set_postscaler(unsigned char set_postsc);
```

set_postsc Establece el valor del postscaler.

Las posibles opciones son:

```
    POSTSC_DIV_1
    POSTSC_DIV_2    // TMR2IF en alto cada 2 desbordamientos de TMR2.
    POSTSC_DIV_3
    POSTSC_DIV_4
    POSTSC_DIV_5
    POSTSC_DIV_6
    POSTSC_DIV_7
    POSTSC_DIV_8
    POSTSC_DIV_9
    POSTSC_DIV_10
    POSTSC_DIV_11
    POSTSC_DIV_12
    POSTSC_DIV_13
    POSTSC_DIV_14
```

```
    POSTSC_DIV_15
    POSTSC_DIV_16
```

La salida del postscaler pone en alto el flag de interrupciones del timer2 (TMR2IF).

2.10.5. timer2_write

Establece (escribe a TMR2) el valor del registro TMR2.

```
void timer2_write(unsigned char set_count);
```

set_count Establece el valor del registro TMR2.

2.10.6. timer2_read

Lee el valor del registro TMR2.

```
#define timer2_read() TMR2
```

2.10.7. Ejemplo de uso del timer2

para compilar necesitas los siguientes comandos

```
Ejemplo de compilacion : ./compila.sh ej_timer2.c 16f877a
```

ej_timer2.c

```
/*
Ejemplo de utilización de las funciones del módulo TIMER2.
válido para la familia 16f87Xa,
En este ejemplo se utiliza el timer2 junto con un copntador por software,
para hacer un led parpadeante:
conectar un led u otra salida a RB7.
RB7 se pondrá en estado alto (Vdd) durante aprox. 1 segundo con reloj de 4 MHz,
y en estado bajo durante otro segundo.
*/

#include <pic/p16f877a.h>

#include <timer2.h>
```

```
int main(void)
{
    unsigned char contador;

    TRISBbits.TRISB7 = 0;
    PORTBbits.RB7 = 0;

    timer2_init();                          // Inicia timer2

    timer2_set_prescaler(PRESC_DIV_16);     // Establece prescaler en frecuencia de instrucciones / 16

    timer2_set_period(255);                 // Establece periodo en 255

    contador = 0;

bucle:

    while (timer2_read() < 255);            // Espera aqui mientras la cuenta del timer < 250

    contador = contador + 1;                // Incrementa contador

    if (contador == 255)                    // Si contador = 255 invierte estado de RB7
    {
        if (PORTBbits.RB7 == 0)
            PORTBbits.RB7 = 1;
        else
            PORTBbits.RB7 = 0;

        contador = 0;                       // Reinicia contador
    }
    goto bucle;
}
```

2.11. Módulo system

Aquí se ecnuentran algunas funciones de configuración

```
#include <system.h>
```

2.11.1. sleep

Pone a dormir el microcontrolador

```
void sleep(void);
```

2.11.2. ASM

Función que incrusta código ASSEMBLER

```
void ASM(char comando[]);
```

2.11.3. Ejemplo System

Para compilar necesitas los siguientes comandos

```
Ejemplo de compilacion : ./compila.sh ej_system 16f877a
```

ej_system.c

```
#include <pic/p16f877a.h>
#include <system.h>
#include <pinio.h>

int main(void)
{
unsigned char dato=0xF0;
unsigned char res;

//Puerto B como salida
ASM("BANKSEL TRISB");
ASM("MOVLW  0x00");
ASM("MOVWF  TRISB");

//0xF0  --> PORTB
ASM("BANKSEL F_REG");//dato se encuentra en el banco de F_REG
ASM("MOVF  %0,W"::"r" (dato));
ASM("BANKSEL PORTB");
ASM("MOVWF  PORTB");

//PORTB --> res
ASM("BANKSEL PORTB");
```

```
ASM("MOVF  PORTB,W");
ASM("BANKSEL F_REG");//res se encuentra en el banco de F_REG
ASM("MOVWF %0":"=v" (res));

//cuando termines de escribir en asembler
//siempre debes dejarlo en el banco de F_REG,
//de lo contrario el programa falla.

//Pongo a dormir al micro
sleep();

return 0;
}
```

Capítulo 3. Biblioteca de Driver's

En esta bibliteca se encuentrán todos los modulos para poder controlar componentes externos, es la parte con mayor potencial apra el aporte de programadores externos.

3.1. KEYPAD 4x4

Cuando se desee conectar un keypad de 4x4 con el puerto D o B, sólo se tiene que agregar la siguiente biblioteca.

```
//para usar el puerto D
#define PORTD_FOR_KEYPAD4X4
#include  <keypad4x4.h>

//para usar el puerto B
#define PORTB_FOR_KEYPAD4X4
#include  <keypad4x4.h>
```

3.1.1. Diagrama de Pines

Los KeyPad son un conjunto de pulsadores dispuestos en forma de una matriz de 4 filas y 4 columnas, estos dispositivos son utilizados para introducir informacion al microcontrolador.

```
ROW0 --> PIN_X0
ROW1 --> PIN_X1
ROW2 --> PIN_X2
ROW3 --> PIN_X3
COL0 --> PIN_X4
COL1 --> PIN_X5
COL2 --> PIN_X6
COL3 --> PIN_X7

 --R-- ROW0 {'1' ,'2' ,'3' ,'A'}
 --R-- ROW1 {'4' ,'5' ,'6' ,'B'}
 --R-- ROW2 {'7' ,'8' ,'9' ,'C'}
 --R-- ROW3 {'*' ,'0' ,'#' ,'D'}
            COL0 COL1 COL2 COL3
              |    |    |    |
              R    R    R    R
              |____|____|____|___+vcc
```

3.1.2. kbd_get

```
char kbd_get(void)
```

Esta funcion no es bloqueante y devuelve una variable de tipo char con el valor de la tecla pulsada, si no encuentra alguna tecla pulsada o si encuentra mas de una tecla, devuelve 0

3.1.3. kbd_getchar

```
char kbd_getchar(void)
```

Esta funcion es similar a kbd_get pero bloqueante tambien devuelve una variable de tipo char con el valor de la tecla pulsada

3.1.4. Ejemplo de KeyPad4x4

para compilar necesitas los siguientes comandos

```
Ejemplo de compilacion : ./compila.sh ej_keypad4x4 16f877a
```

ej_pinio.c

```
#include <pic/p16f877a.h>

#define FOSC_HZ 20000000

#define PORTB_FOR_KEYPAD4X4

#include <uart.h>
#include <keypad4x4.h>
#include <delayms.h>

int main (void)
{
char tmp;
char INTRO[]="\nPRESIONA UNA TECLA\n";

TRISB=0xFF;
TRISC=0xFF;

delayms(100);

uart_open(SET_9600_8N1);
```

```
set_pullup_portb(TRUE);
uart_puts(INTRO);

while(TRUE)
{
delayms(200);
tmp = kbd_getchar();
uart_putc(tmp);
}
return 0;
}
```

3.2. KEYPAD 4x4 FLEX

Cuando se desee conectar un keypad de 4x4 con cualquier PIN, sólo se tiene que agregar la siguiente biblioteca.

```
#define ROW0 PIN_C0
#define ROW1 PIN_C1
#define ROW2 PIN_C2
#define ROW3 PIN_C3
#define COL0 PIN_C4
#define COL1 PIN_C5
#define COL2 PIN_C6
#define COL3 PIN_C7

#include  <keypad4x4flex.h>
```

3.2.1. Diagrama de Pines

Los KeyPad son un conjunto de pulsadores dispuestos en forma de una matriz de 4 filas y 4 columnas, estos dispositivos son utilizados para introducir información al microcontrolador.

```
ROW0 --> PIN_X0
ROW1 --> PIN_X1
ROW2 --> PIN_X2
ROW3 --> PIN_X3
COL0 --> PIN_X4
COL1 --> PIN_X5
COL2 --> PIN_X6
COL3 --> PIN_X7

 --R-- ROW0 {'1' ,'2' ,'3' ,'A'}
 --R-- ROW1 {'4' ,'5' ,'6' ,'B'}
 --R-- ROW2 {'7' ,'8' ,'9' ,'C'}
```

```
 --R-- ROW3 {'*' ,'0' ,'#' ,'D'}
           COL0 COL1 COL2 COL3
             |    |    |    |
             R    R    R    R
             |____|____|____|___+vcc
```

3.2.2. kbd_get

```
char kbd_get(void)
```

Esta funcion no es bloqueante y devuelve una variable de tipo char con el valor de la tecla pulsada, si no encuentra alguna tecla pulsada o si encuentra mas de una tecla, devuelve 0

3.2.3. kbd_getchar

```
char kbd_getchar(void)
```

Esta funcion es similar a kbd_get pero bloqueante tambien devuelve una variable de tipo char con el valor de la tecla pulsada

3.2.4. Ejemplo de KeyPad4x4 Flex

para compilar necesitas los siguientes comandos

```
Ejemplo de compilacion : ./compila.sh ej_keypad4x4flex 16f877a
```

ej_pinio.c

```
#include <pic/p16f877a.h>

#define FOSC_HZ 20000000

#define ROW0  PIN_B0
#define ROW1  PIN_B1
#define ROW2  PIN_B2
#define ROW3  PIN_B3
#define COL0  PIN_B4
#define COL1  PIN_B5
#define COL2  PIN_B6
#define COL3  PIN_B7
```

```
#include <uart.h>
#include <keypad4x4flex.h>
#include <delayms.h>

int main (void)
{
char tmp;
char INTRO[]="PRESIONA UNA TECLA\n";

TRISB=0xFF;
TRISC=0xFF;

delayms(100);

uart_open(SET_9600_8N1);
set_pullup_portb(TRUE);
uart_puts(INTRO);

while(TRUE)
{
delayms(200);
tmp = kbd_getchar();
uart_putc(tmp);
}
return 0;
}
```

Capítulo 4. Biblioteca de Pic's

Aqui encontraran las definiciones de todos los registros del PIC, los registros pueden tratarse de dos formas

```
como variable:
PORTA=0xff;
X=PORTA;

como estructura:
PORTAbits.RA0=1;
b=PORTAbits.RA0;
```

4.1. Pic16f877a

Para definiciones esclusivas para el pic 16f877a, se debe incluir.

```
#include <pic/p16f877a.h>
```

4.2. Ejemplo de Pic

para compilar necesitas los siguientes comandos

```
Ejemplo de compilacion : ./compila.sh ej_pic 16f877a
```

ej_pic.c

```
#include <pic/p16f877a.h>

int main(void)
{
TRISCbits.TRISC2=0; //Pin 2 del PUERTO C, como entrada
TRISB=0xFF; //Todo el PUERTO B, como entrada
TRISA=0x00; //Todo el PUERTO A, como salida
```

```
while(1)
{
//Con un cristal de 20Mhz
//esto genera un clock de 714Khz
//con un pulso alto de 200ns
PORTCbits.RC2=1;//Pin 2 del PUERTO C, alto
PORTCbits.RC2=0;//Pin 2 del PUERTO C, bajo
}
return 0;
}
```

Capítulo 5. Biblioteca de Utilitarios

Es un conjunto de bibliotecas de utilitarios de uso genérico, cuyo uso es independiente del tipo PIC ;

5.1. Módulo Delayms

Autor : Pedro José Ramírez Gutiérrez

pjanragu en telefonica.net

Rutina que genera un retardo en **ms**

Antes necesita definir el valor de la frecuencia del cristal

```
#define FOSC_HZ 20000000

#include <delayms.h>
```

5.1.1. delayms

La función genera un retardo en milisegundos.

```
void delayms(unsigned int retraso);
```

5.1.2. Ejemplo de DelayMs

para compilar necesitas los siguientes comandos.

```
Ejemplo de compilación : ./compila.sh ej_delay 16f877a
```

ej_delay.c

```
#include <pic/p16f877a.h>

#define FOSC_HZ 20000000
```

```
#include <uart.h>
#include <pinio.h>   /*define los PIN_XY*/
#include <delayms.h>

int main (void)
{
char c=0;
char INTRO[8]="TECLEA\n";
int T=100;

delayms(100);

uart_open(SET_115200_8N1);

uart_puts(INTRO);
while(c!=13)
{
if(uart_kbhit()==1)
{
c=uart_getc()-'0';
uart_putc(c+'0');
T=100*c;
}
delayms(T);
output_high(PIN_C2);
delayms(T);
output_low(PIN_C2);
}
uart_close();

return 0;
}
```

5.2. Módulo Memory RAM

Esta biblioteca sirve para averiguar la memoria RAM libre en el PIC. Las funciones nos ayudarán a conocer la memoria libre en cada banco.

```
#include <memory.h>
```

5.2.1. memory_bank0

La función devuelve la cantidad de bytes (RAM) libres en el BANCO 0.

Por defecto esta función se habilita, debido a la macro I_HAVE_BANK0.

```
BYTE memory_bank0(void);
```

5.2.2. memory_bank1

La función devuelve la cantidad de bytes (RAM) libres en el BANCO 1.

La funcion debe ser habilitada con la macro I_HAVE_BANK1.

```
BYTE memory_bank1(void);
```

5.2.3. memory_bank2

La función devuelve la cantidad de bytes (RAM) libres en el BANCO 2.

La funcion debe ser habilitada con la macro I_HAVE_BANK2.

```
BYTE memory_bank2(void);
```

5.2.4. memory_bank3

La función devuelve la cantidad de bytes (RAM) libres en el BANCO 3.

La funcion debe ser habilitada con la macro I_HAVE_BANK3.

```
BYTE memory_bank3(void);
```

5.2.5. memory_bank_all

La función devuelve la cantidad de bytes (RAM) libres en todos los BANCOS.

La función necesita que se habilite con la macro I_HAVE_BANK0,I_HAVE_BANK1, etc. Los bancos que se usen.

```
BYTE memory_bank_all(void);
```

5.2.6. Ejemplo de Memory

para compilar necesitas los siguientes comandos

```
Ejemplo de compilación : ./compila.sh ej_memory 16f877a
```

ej_memory.c

```
#include <pic/p16f877a.h>

#define FOSC_HZ 20000000

#include <uart.h>
#include <delayms.h>

#define I_HAVE_BANK0
#define I_HAVE_BANK1

#include <memory.h>

int main (void)
{
BYTE x;

delayms(250);
uart_open(SET_9600_8N1);

x=memory_bank0(); //bytes libres en  el Banco 0
uart_putc(x);

x=memory_bank1(); //bytes libres en  el Banco 1
uart_putc(x);

delayms(100);//retardo para dar tiempo a que se envie el ultimo caracter
uart_close();

return 0;
}
```

5.3. Módulo UART2

Esta biblioteca nos ayudará a convertir números a binarios a formatos de de texto en representación DECIMAL y HEXADECIMAL.

```
#include <uart2.h>
```

5.3.1. puth

transforma la cifra hexadecimal (medio octeto) en 1 carácter ASCII y lo envia.

Por ejemplo si tienes el hexadecimal 0x4A y tomas el número "A" (osea 10 en decimal - 1010 en binario), "puth" lo transforma en el caracter 'A' y lo envia.

```
Transforma el numero "0" en el caracter '0'.
Transforma el numero "1" en el caracter '1'.
Transforma el numero "2" en el caracter '2'.
...
Transforma el numero "A" en el caracter 'A'.
Transforma el numero "B" en el caracter 'B'.
...
Transforma el numero "F" en el caracter 'F'.
```

El caracter generado es enviado usando la función miputc.

```
void puth(void (*miputc)(char),char a);
```

5.3.2. puthex

transforma un número hexadecimal (un octeto) en 2 caracteres ASCII y lo envia.

Por ejemplo, si tienes el hexadecimal 0x4A, puthex toma el número "4" lo transforma en el caracter '4' y lo envia con miputc, toma el numero "A" lo transforma en el caracter 'A' y lo envia con miputc.

```
0x4A => miputc('4');
miputc('A');
```

```
void puthex(void (*miputc)(char),char nb);
```

5.3.3. putint

transforma un número entero de 16 bit en su representación hexadecimal de 4 caracteres ASCII y luego los envia uno por uno.

Por ejemplo, si tienes el número 1030, putint lo transforma en su representación en HEXADECIMAL "0x0406", putint toma ’0’,’4’,’0’,’6’ y los envia con miputc.

```
1030 => 0x0406 => miputc('0');
miputc('4');
miputc('0');
miputc('6');
```

```
void putint(void (*miputc)(char),int num);
```

5.3.4. geth

Lee un carácter ASCII y lo transforma en una cifra hexadecimal.

Por ejemplo si lees el caracter "A" , "geth" lo transforma en el numero 10 en decimal codificado en binario BCD.

```
Transforma el caracter '0' en el número 0.
Transforma el caracter '1' en el número 1.
Transforma el caracter '2' en el número 2.
...
Transforma el caracter 'a' ó 'A' en el número 10 (0x0a).
Transforma el caracter 'b' ó 'B' en el número 11 (0x0b).
...
Transforma el caracter 'f' ó 'F' en el número 15 (0x0f).

//cualquier otro numero se convierte 0xff
```

Para obtener el caracter se utiliza la función migetc.

```
char geth(char (*migetc)(void));
```

5.3.5. gethex

Lee 2 caracteres ASCII y transforma en nÃºmero hexadecimal.

Por ejemplo, si tienes dos caracteres ASCII como "4" y "A", gethex toma el caracter "4" y lo transforma en el número 0x40 y toma el caracter "A" y lo transforma en el número 0x0A luego los une en 0x4A.

```
miputc('4');
miputc('A'); => 0x4A
```

```
char gethex(char (*migetc)(void));
```

5.3.6. getint

Lee 4 caracteres ASCII que representan un número en hexadecimal y los transforma un número entero de 16 bit.

Por ejemplo, si tienes los caracteres '0','4','0' y '6', getint devuelve el número 1030.

```
miputc('0');
miputc('4');
miputc('0');
miputc('6'); => 0x0406 => 1030
```

```
int getint(char (*migetc)(void));
```

5.3.7. getd

Lee un carácter ASCII y lo transforma en una sola cifra decimal.

Por ejemplo si lees el caracter "2" , "getd" lo transforma en el numero 2 en decimal codificado en binario BCD.

```
Transforma el caracter ’0’ en el número 0.
Transforma el caracter ’1’ en el número 1.
Transforma el caracter ’2’ en el número 2.
...
Transforma el caracter ’9’ en el número 9.

//cualquier otro numero se convierte 0xff
```

Para obtener el caracter se utiliza la función migetc.

```
char geth(char (*migetc)(void));
```

5.3.8. Ejemplo de UART2

para compilar necesitas los siguientes comandos

```
Ejemplo de compilación : ./compila.sh ej_uart2 16f877a
```

ej_uart2.c

```
#include <pic/p16f877a.h>

#define FOSC_HZ 20000000

#include <uart.h>
#include <uart2.h>

int main (void)
{
char a=0x4a;
int  x=1030;

delayms(100);
uart_open(SET_9600_8N1);
uart_puts("UART2\n");

puthex(uart_putc,a);
uart_putc(’\n’);
putint(uart_putc,x);
uart_putc(’\n’);

uart_puts("FIN\n");
```

```
delayms(100);//dar un poco de tiempo para que se envie el ultimo caracter
uart_close();

return 0;
}
```

Capítulo 6. Biblioteca Estandar de C

Es un conjunto de las bibliotecas estandar de C, el código de estas funciones es independiente del tipo de PIC

6.1. String

Define la biblioteca estandar de C : String.h.

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

6.1.1. Memchr

Localiza la primera aparición del caracter c (convertido a unsigned char) en los primeros n caracteres (cada uno interpretado como un unsigned char) del objeto apuntado por s.

La función retorna un puntero al carácter localizado, o un puntero nulo si el caracter no apareció en el objeto.

```
void *memchr (const void* s, int c, size_t n);
```

6.1.2. Memcmp

Compara los primeros n caracteres del objeto apuntado por s1 (interpretado como unsigned char) con los primeros n caracteres del objeto apuntado por s2 (interpretado como unsigned char).

La función retorna un número entero mayor, igual, o menor que cero, apropiadamente segun el objeto apuntado por s1 es mayor, igual, o menor que el objeto apuntado por s2.

```
int memcmp (const void* s1, const void* s2, size_t n);
```

6.1.3. Memcpy

Copia los primeros n caracteres del objeto apuntado por s2 al objeto apuntado por s1.

La función retorna el valor de s1. Si al copiar un objeto al otro se superponen, entonces el comportamiento no está definido.

```
void* memcpy (void* s1, const void* s2, size_t n);
```

6.1.4. Memmove

Copia los primeros n caracteres del objeto apuntado por s2 al objeto apuntado por s1.

La función retorna el valor de s1. Si al copiar un objeto al otro se superponen, entonces el comportamiento no esta definido.

```
void*  memmove (void  *s1, const void *s2, size_t n);
```

6.1.5. Memset

Copia el valor de c (convertido a unsigned char) en cada uno de los primeros n caracteres en el objeto apuntado por s.

La función retorna el valor de s.

```
void*  memset (void* s, int c, size_t n);
```

6.1.6. Strcat

AÃ±ade una copia de la cadena apuntada por s2 (incluyendo el caracter nulo) al final de la cadena apuntada por s1. El carÃ¡cter inicial de s2 sobrescribe el caracter nulo al final de s1.

La función retorna el valor de s1. Si la copia hace que los objetos se superpongan, entonces el comportamiento no está definido

```
char* strcat (char *s1, const char *s2);
```

6.1.7. Strchr

Localiza la primera aparicion de c (convertido a unsigned char) en la cadena apuntada por s (incluyendo el caracter nulo).

La función retorna un puntero a partir del caracter encontrado. Si no se ha encontrado el caracter, c, entonces retorna un puntero null.

```
char*  strchr (const char* s, int c) ;
```

6.1.8. Strcmp

Compara la cadena apuntada por s1 con la cadena apuntada por s2.

La función retorna un número entero mayor, igual, o menor que cero, apropiadamente segun la cadena apuntada por s1 es mayor, igual, o menor que la cadena apuntada por s2.

```
int strcmp (const char* s1, const char* s2);
```

6.1.9. Strcpy

Copia la cadena apuntada por s2 (incluyendo el carácter nulo) a la cadena apuntada por s1.

La función retorna el valor de s1. Si al copiar una cadena a la otra se superponen, entonces el comportamiento no esta definido.

```
char*  strcpy (char *s1, const char *s2);
```

6.1.10. Strcspn

Cuenta el numero de caracteres de una subcadena inicial apuntada por s1 que no contenga ninguno de los caracteres en la cadena apuntada por s2.

La función retorna el número de caracteres leidos de la subcadena hasta que halla alguno de los caracteres de s2. El carÃ¡cter nulo no se cuenta.

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

6.1.11. Strerror

Convierte el numero de error en errnum a un mensaje de error (una cadena de caracteres).

La función retorna la cadena de caracteres conteniendo el mensaje asociado con el numero de error. Esta conversión y el contenido del mensaje dependen de la implementación. La cadena no sera modificada por el programa, pero si puede ser sobrescrito con otra llamada a la misma funcion.

```
char*  strerror (int errnum);
```

6.1.12. Strlen

Calcula el número de caracteres de la cadena apuntada por s.

La función retorna el número de caracteres hasta el caracter nulo, que no se incluye.

```
size_t  strlen (const char *s);
```

6.1.13. Strncat

AÃ±ade no más de n caracteres (un caracter nulo y los demas caracteres siguientes no son aÃ±adidos) de la cadena apuntada por s2 al final de la cadena apuntada por s1. El caracter inicial de s2 sobrescribe el caracter nulo al final de s1. El caracter nulo siempre es aÃ±adido al resultado.

La función retorna el numero de caracteres hasta el caracter nulo, que no se incluye.

```
char *strncat(char *s1, const char *s2, size_t n);
```

6.1.14. Strncmp

Compara no más de n caracteres (caracteres posteriores al caracter nulo no se tienen en cuenta) de la cadena apuntada por s1 con la cadena apuntada por s2.

La función retorna un numero entero mayor, igual, o menor que cero, apropiadamente segun la cadena apuntada por s1 es mayor, igual, o menor que la cadena apuntada por s2.

```
int strncmp(const char *s1, const char *s2, size_t n);
```

6.1.15. Strncpy

Copia no mas de n caracteres (caracteres posteriores al caracter nulo no son copiados) de la cadena apuntada por s2 a la cadena apuntada por s1

La función retorna el valor de s1. Si al copiar una cadena a la otra se superponen, entonces el comportamiento no esta definido. Si el array/arreglo apuntado por s2 es una cadena que es mas corta que n caracteres, entonces caracteres nulos son aÃ±adidos a la copia en el array apuntado por s1.

```
char *strncpy(char *s1, const char *s2, size_t n);
```

6.1.16. Strpbrk

Localiza la primera aparición de la cadena apuntada por s1 de cualquier caracter de la cadena apuntada por s2.

La función retorna un puntero al caracter, o un puntero nulo si ningun caracter de s2 aparecio en s1.

```
char *strpbrk(const char *s1, const char *s2);
```

6.1.17. Strrchr

Localiza la última aparición de c (convertido a unsigned char) en la cadena apuntada por s (incluyendo el carácter nulo).

La funcián retorna un puntero a partir del caracter encontrado. Si no se ha encontrado el caracter, c, entonces retorna un puntero nulo.

```
char *strrchr(const char *s, int c);
```

6.1.18. Strspn

devuelve la posición del primer carácter de una cadena que no coincide con ninguno de los caracteres de otra cadena dada

La función devuelve la posición del primer carácter de una cadena que no coincide con ninguno de los caracteres de otra cadena dada.

```
size_t strspn(const char *s1, const char *s2);
```

6.1.19. Strstr

Busca una cadena dentro de otra.

La función retorna un puntero a la cadena encontrada, o un puntero nulo si no se encontró la cadena. Si s2 apunta a una cadena de longitud cero, la función retorna s1.

```
char *strstr(const char *s1, const char *s2);
```

Capítulo 7. Ejemplos

Un conjunto de ejemplos de todos los drivers y extensiones

7.1. Compilación

Se puede usar los métodos de compilación vistos en Metodos de Compilacion

```
./compila.sh ejemplo1 16f877a
```

7.2. ejemplo1.c

Un ejemplo simple.

```
#define PuertoA (*(volatile unsigned char *)(0x005))
#define TRISdeA (*(volatile unsigned char *)(0x085))

int main (void)
{
unsigned char c=100;

TRISdeA=0;

PuertoA=c;

return 0;
}
```

Capítulo 8. Referencias

- zsoluciones (http://zsoluciones.com)
- http://pic-gcc-library.sourceforge.net (http://pic-gcc-library.sourceforge.net)
- pjmicrocontroladores (http://pjmicrocontroladores.wordpress.com/)
- forja.rediris.es projects cls-pic-16f877 (https://forja.rediris.es/projects/cls-pic-16f877/)
- http://pic-linux.foroactivo.net (http://pic-linux.foroactivo.net/)
- http://per.launay.free.fr/ (http://per.launay.free.fr/dokuwiki-2008-05-05/doku.php)

Apéndice A. Preguntas Frecuentes

1. **Puedo participar en el proyecto Pic Gcc ?**

   Si, entra en contacto con: Pedro

   pjanragu en telefonica.net

2. **Puedo participar en el proyecto Pic Gcc Library ?**

   Si, entra en contacto con: Fernando

   fernando.pujaico.rivera en gmail.com

3. **Pic Gcc puede compilar otras familias de PIC ?**

   hmmmm, si nos ayudas, Si.

4. **Que grabadores de PIC puedo utilizar ?**

   Puedes construir o comprar cualquier grabador, por ejemplo puedes construir un grabador modelo PROPIC II y usar algun software de programación

   En Linux: PikLab, PikDev, etc.

   En Windows: PicKit2, WinPic800, IcProg

5. **Que Entorno Integrado de Desarrollo puedo usar ?**

   Puedes usar GtkPicGccIDE - Zandor (http://pic-gcc-library.sourceforge.net/data/?page_id=3)