PIC-GCC-Library
Bibliothèque standard et de composants

Ing. Fernando Pujaico Rivera [1]
Développeur Principal
Chercheur
Universidad Nacional de Ingenieria
Ing. Pedro Jose Ramirez Gutierrez
Prof. Pierre Launay
Santiago González Rodríguez

traduction en français Pierre Launay per.launay chez free.fr

PIC-GCC-Library est un ensemble de bibliothèques pour le compilateur PIC GCC.

PIC GCC est un compilateur de C pour microcontrôleurs PIC de la famille 16F de Microchip

[1] courriel : fernando.pujaico.rivera en gmail.com

Table des matières

Liste des tableaux

Chapitre 1

Introduction

PIC-GCC es un compilateur de langage C pour microcontrôleurs de la famille PIC16 de MICROCHIP.

FIG. 1.1 – Logo de Pic Gcc Library

1.1 Description des Dossiers

Dossiers à la racine	Description des Dossiers
bin	Dossier avec les outils pour la compilation
devel	Dossier avec le code source des bibliothèques (seulement la version pour développeurs).
doc	Dossier avec la documentation pour l'utilisation des bibliothèques.
examples	Dossier avec des exemples d'utilisation (ne pas modifier)
include	Dossier avec la définition de toutes les fonctions des bibliothèques.
lib	Dossier avec les bibliothèques statiques prédéfinies.
refman	Dossier avec le manuel de référence des bibliothèques.
schematics	Dossier avec les schémas électroniques correspondant aux exemples

TAB. 1.1 – Description des dossiers de Pic-Gcc-Library

1.2 Méthodes de Compilation

pour compiler un programme on dispose de deux options :

1.2.1 Première Forme

```
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 seconde Forme

./compila.sh ejemplo 16f877a .

si le programme que je veux compiler n'est pas dans le dossier où se situe compila.sh

./compila.sh ejemplo 16f877a /répertoire_du_code_source

1.3 Bibliothèques en Pic-Gcc-Library

TAB. 1.2 – Dossier dans le Dossier INCLUDE de Pic-Gcc-Library

Dossiers	Description des Dossiers
disp	Contient les fonctions pour contrôler les périphériques de chaque dispositif : UART, SPI, PWM, IO, etc.
drivers	Contient les fonctions pour contrôler les composants externes comme : Keypad, Lcd, etc.
pic	Contient les définitions basiques pour chaque type de PIC
util	Dossier avec la définition de toutes les fonctions spéciales comme : ”delay”
.	Dans le dossier racine on trouve les bibliothèques standard du C.

1.4 Microcontrôleurs supportés

Microcontrôleur	Disp	Driver	Pic	Util	Libc
12F635	-	-	-	-	-
16F610	-	-	-	-	-
16F616	-	-	-	-	-
16F627	-	-	-	OK	OK
16F627A	-	-	-	OK	OK
16F628	-	-	-	OK	OK
16F628A	-	-	-	OK	OK
16F630	-	-	-	-	-
16F631	-	-	-	-	-
16F636	-	-	-	OK	OK

Microcontrôleur	Disp	Driver	Pic	Util	Libc
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
16F876A	-	-	-	OK	OK
16F877	-	-	-	OK	OK
16F877A	OK	OK	OK	OK	OK
16F88	-	-	-	-	-
16F882	-	-	-	-	-
16F883	-	-	-	-	-
16F884	-	-	-	-	-
16F886	-	-	-	-	-
16F887	-	-	-	-	-
16F913	-	-	-	OK	OK
16F914	-	-	-	OK	OK
16F916	-	-	-	OK	OK
16F917	-	-	-	OK	OK
16F946	-	-	-	-	-
16C432	-	-	-	OK	OK
16C433	-	-	-	OK	OK

Microcontrôleur	Disp	Driver	Pic	Util	Libc
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
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

Microcontrôleur	Disp	Driver	Pic	Util	Libc
16C84	-	-	-	-	-
16C923	-	-	-	OK	OK
16C924	-	-	-	OK	OK
16C925	-	-	-	OK	OK
16C926	-	-	-	OK	OK
16CR62	-	-	-	OK	OK
16CR620A	-	-	-	-	-
16CR63	-	-	-	OK	OK
16CR64	-	-	-	OK	OK
16CR65	-	-	-	OK	OK
16CR72	-	-	-	OK	OK
16CR73	-	-	-	-	-
16CR74	-	-	-	-	-
16CR76	-	-	-	-	-
16CR77	-	-	-	-	-
16CR83	-	-	-	-	-
16CR84	-	-	-	OK	OK

TAB. 1.3 – Microcontrôleurs supportés actuellement

1.5 Entêtes définies

Dossier	Entêtes
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

TAB. 1.4 – Entêtes définies dans Pic-Gcc-Library

1.6 Bibliothèque de Dispositifs Supportés

μ C	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	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F627A	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F628	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F628A	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F630	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F631	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F636	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F639	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F648A	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F676	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F677	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F684	-	-	-	-	-	-	-	-	-	-	-	-	-	-

μ C	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
16F685	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F687	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F688	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F689	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F690	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F716	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F72	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F73	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F737	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F74	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F747	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F76	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F767	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F77	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F777	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F785	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F818	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F819	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F83	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F84	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F84A	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F87	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F870	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F871	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F872	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F873	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F873A	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F874	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16F874A	-	-	-	-	-	-	-	-	-	-	-	-	-	-
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	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C558	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C61	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C62	-	-	-	-	-	-	-	-	-	-	-	-	-	-

μ C	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
16C62A	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C62B	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C620	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C620A	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C621	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C621A	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C622	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C622A	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C63	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C63A	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C64	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C64A	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C641	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C642	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C65	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C65A	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C65B	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C66	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C661	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C662	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C67	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C71	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C710	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C711	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C712	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C715	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C716	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C717	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C72	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C72A	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C73	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C73A	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C73B	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C74	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C74A	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C74B	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C745	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C76	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C765	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C77	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C770	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C771	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C773	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C774	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C781	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C782	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C84	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C923	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C924	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C925	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16C926	-	-	-	-	-	-	-	-	-	-	-	-	-	-

μ C	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
16CR62	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16CR620A	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16CR63	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16CR64	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16CR65	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16CR72	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16CR73	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16CR74	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16CR76	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16CR77	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16CR83	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16CR84	-	-	-	-	-	-	-	-	-	-	-	-	-	-

TAB. 1.5 – Dispositifs supportés actuellement

Chapitre 2

Bibliothèque de Dispositifs

Dans tous les modules montrés ici son utilisation dépend du type de PIC

2.1 Module EEPROM Interne

Montre les fonctions pour pouvoir travailler avec l'EEPROM Interne des PIC.

`include <eeprom.h>`

2.1.1 eeprom_read

Lit l'adresse **addr** dans l'EEprom interne du PIC

BYTE eeprom_read(BYTE addr) ;

2.1.2 eeprom_write

Écrit la donnée **data** à l'adresse **addr** dans l'EEprom interne du PIC

void eeprom_write(BYTE data,BYTE addr) ;

2.1.3 Exemple de Eeprom

pour compiler vous avez besoin des commandes suivantes :

`Exemple de compilation : ./compila.sh ej_eeprom 16f877a`

ej_eeprom.c

```
#include <pic/p16f877a.h>
#define FOSC_HZ 20000000
#include <uart.h>
#include <delayms.h>
#include <inteeprom.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(read_int_eeprom(0));
        uart_putc(read_int_eeprom(1));
        uart_putc(read_int_eeprom(2));
        uart_putc(read_int_eeprom(3));
        uart_putc('\n');
        i=0;
        while((i<4)&&(tmp!=13))
          {
                uart_putc('>');
                tmp = uart_getc();
                uart_putc(tmp);
                uart_putc('\n');
                write_int_eeprom(tmp,i);
                i++;
          }
        uart_puts("FIN\n");
        return 0;
}
```

2.2 Module PWM

Voici les fonctions pour pouvoir travailler avec le module PWM Interne des PIC. Toutes les équations sont en secondes .Tosc est l'inverse de la fréquence du quartz FOSC_HZ.

```
Periode=(Pr2+1)*4*Tosc*PreescalerTMR2
TempsImpulsionHaute=CCPR1L\_CCP1CON54*Tosc*PreEscalerTmr2
Pré-diviseur =PreEscaler

#include <pwm.h>
#define TMR2_PREESCALER_1 0
#define TMR2_PREESCALER_4 1
#define TMR2_PREESCALER_16 2
```

2.2.1 pwm1_set_data

Configure le module PWM1 du PIC, il est nécessaire d'indiquer le pré-diviseur (Preescaler) et le CCPR1L_CCP1CON54

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

2.2.2 pwm2_set_data

Configure le module PWM2 du PIC, il est nécessaire d'indiquer le pré-diviseur (Preescaler) et le CCPR2L_CCP2CON54

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

2.2.3 Exemple de PWM

pour compiler vous avez besoin des commandes suivantes :

Exemple de compilation : ./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 Module UART

Montre les fonctions pour pouvoir travailler avec le module UART Interne des PIC. Avant d'utiliser ce module on doit définir la fréquence utilisée avec la commande.

```
#define FOSC_HZ 20000000
```

On a réalisé quelques définitions comme :

```
#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
```

Ces définitions peuvent être utilisées dans la fonction uart_open.

2.3.1 uart_open

Configure le port Série Asynchrone

void uart_open(BYTE STATUS_SPBRG, BYTE STATUS_SPEED) ;

exemple :

uart_open(SET_9600_8N1) ;

2.3.2 uart_kbhit

Renvoie 1 si il y a un octet (byte) dans le tampon (buffer) d'entrée du port Série Asynchrone

BYTE uart_kbhit(void) ;

2.3.3 uart_getc

Lit un octet (byte) dans le tampon (buffer) d'entrée du port Série Asynchrone, si il n'y a rien attend jusqu'à l'arrivée d'un octet, la fonction est bloquante

char uart_getc(void) ;

2.3.4 uart_putc

Écrit une donnée sur le port Série Asynchrone

void uart_putc(char dato) ;

2.3.5 uart_puts

Écrit une chaîne de données sur le port Série Asynchrone

void uart_puts(char *datos) ;

2.3.6 uart_close

Ferme le port Série Asynchrone

void uart_close(void) ;

2.3.7 Exemple de UART

pour compiler vous avez besoin des commandes suivantes :

Exemple de compilation : ./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 Module IO (E/S) et manipulation des broches

Montre les fonctions pour pouvoir travailler avec les broches du PIC.

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

2.4.1 output_high_slow

Établit une broche de sortie (PIN) à l'état haut, tension proche de VCC, vérifie l'état de **tris**

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

2.4.2 output_high_fast

Établit une broche de sortie (PIN) à l'état haut, tension proche de VCC, ne vérifie pas l'état de **tris**

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

2.4.3 output_high

Établit une broche de sortie (PIN) à l'état haut, tension proche de VCC. La vérification de l'état de tris dépend de l'état de la macro SLOW_IO come vrai TRUE ou faux FALSE

Par défaut, si on n'écrit pas #define SLOW_IO TRUE , SLOW_IO a la valeur TRUE

```
#define SLOW_IO TRUE

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

2.4.4 output_low_slow

Établit une broche de sortie (PIN) a l'état bas, tension proche de la masse, vérifie l'état de **tris**

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

2.4.5 output_low_fast

Établit une broche de sortie (PIN) a l'état bas, tension proche de la masse, ne vérifie pas l'état de **tris**

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

2.4.6 output_low

Établit une broche de sortie (PIN) a l'état bas, tension proche de la masse, La vérification de l'état de tris dépend de l'état de la macro SLOW_IO come vrai TRUE ou faux FALSE

Par défaut, si on n'écrit pas #define SLOW_IO TRUE , SLOW_IO a la valeur TRUE

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

2.4.7 input_slow

Retourne l'état d'une broche d'entrée (pin). peut être 1 ou 0, vérifie l'état de **tris**

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

2.4.8 input_fast

Retourne l'état d'une broche d'entrée (pin) peut être 1 ou 0, ne vérifie pas l'état de **tris**

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

2.4.9 input

Retourne l'état d'une broche d'entrée (pin). peut être 1 ou 0 La vérification de l'état de tris dépend de l'état de la macro SLOW_IO come vrai TRUE ou faux FALSE

Par défaut, si on n'écrit pas #define SLOW_IO TRUE , SLOW_IO a la valeur TRUE

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

2.4.10 set_pin_to_slow

Établit l'état d'une broche de sortie (pin) à 1 ou 0

```
 void set_pin_to_slow(volatile BYTE puerto,
                        BYTE pin,
                        BYTE X);
        exemple :
        set_pin_to_slow(PIN_C1,1);//la broche C1 est maintenant à l'état haut
        set_pin_to_slow(PIN_C1,0);//la broche C1 est maintenant à l'état bas
```

2.4.11 set_pin_to_fast

Établit l'état d'une broche de sortie (pin) à 1 ou 0

```
void set_pin_to_fast(volatile BYTE puerto,
                     BYTE pin,
                     BYTE X);
       exemple :
       set_pin_to_fast(PIN_C1,1);//la broche C1 est maintenant à l'état haut
       set_pin_to_fast(PIN_C1,0);//la broche C1 est maintenant à l'état bas
```

2.4.12 set_pin_to

Établit l'état d'une broche de sortie (pin) à 1 ou 0

```
void set_pin_to(volatile BYTE puerto,
                     BYTE pin,
                     BYTE X);
       exemple :
       set_pin_to(PIN_C1,1);//la broche C1 est maintenant à l'état haut
       set_pin_to(PIN_C1,0);//la broche C1 est maintenant à l'état bas
```

2.4.13 set_pin_tris_to

TRIS data direction register : registre de direction des données.

Établit le TRIS d'une broche à 1 ou 0
– si TRIS est à 1, la broche est en entrée
– si TRIS est à 0, la broche est en sortie

```
         void set_pin_tris_to(volatile BYTE *tris,
                volatile BYTE *puerto,
                BYTE pin,
                BYTE X);
exemple:
       set_pin_tris_to(PIN_C1,1);//el TRIS C1 à 1, C1 en entrée
       set_pin_tris_to(PIN_C1,0);//el TRIS C1 à 0, C1 en sortie
```

2.4.14 set_porta_as_digital

Établit le port A comme DIGITAL

```
  #define set_porta_as_digital() ADCON1=0x06
       exemple:
       set_porta_as_digital();
```

2.4.15 set_pullup_portb

autorise ou inhibe les résistances de tirage vers le haut (pull-up) du port B

– pour autoriser les résistances de tirage vers le haut (pull-up) X= 1 o TRUE
– pour inhiber les résistances de tirage vers le haut (pull-up) X= 0 o FALSE

void set_pullup_portb(BYTE X) ;

2.4.16 Exemple de PinIO

pour compiler vous avez besoin des commandes suivantes :

```
Exemple de compilation : ~./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 Module I2C

Auteur : Pierre Launay

per.launay chez free.fr

Le bus I2C est un bus série synchrone développé par Philips

Description du bus I2C : http ://fr.wikipédia.org/wiki/I2C

Comprendre le bus I2C en espagnol http://robots-argentina.com.ar/Comunicacion_busI2c.htm

Voici les fonctions pour travailler avec le Module I2c en mode Maître Interne dans les PIC

Avant d'utiliser ce module, on doit définir la fréquence utilisée avec la commande :

```
#define FOSC_HZ 4000000
```

```
#include <i2c.h>
```

Quelques définitions prédéfinies :

```
#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
```

Ces définitions peuvent être utilisées dans la fonction i2c_init

2.5.1 i2c_init

Configure le port série synchrone

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

Attend que le bus soit libre.

void i2c_wait_for_idle(void) ;

2.5.3 i2c_start

Début de trame, séquence de départ

void i2c_start(void) ;

2.5.4 i2c_restart

Recommencer une trame, fin de trame et début de trame permet de mettre 2 trames pour lire. Voir i2c_read()

Exemple pour lire les secondes dans une trame : Début de trame (start), écrire l'adresse du circuit I2c, écrire l'adresse du registre des secondes. Relancer une trame (restart), écrire l'adresse ducircuit I2C, lire le registre des secondes, fin de trame(stop).

void i2c_restart(void) ;

2.5.5 i2c_stop

fin de trame, séquence d'arrêt

void i2c_stop(void) ;

2.5.6 i2c_delay

Temps d'attente entre fin de trame et début de trame $>= 4,7\mu s$ (fréquence d'horloge de 100KHz)

void i2c_delay(void) ;

2.5.7 i2c_ack_read

réception de l'acquittement

unsigned char i2c_ack_read (v oid) ;

"0" bonne réception

"1" mauvaise réception

2.5.8 i2c_ack_write

émission de l'acquittement

void i2c_ack_write(unsigned char ack) ;

"0" la trame peut continuer "1" la trame s'arrête

2.5.9 i2c_write

Le maître écrit dans l'esclave le caractère c, le caractère c peut être une adresse, un registre interne, une donnée
void i2c_write(unsigned char c) ;

2.5.10 i2c_write_with_ack

Le maître écrit dans l'esclave le caractère c, et reçoit l'acquittement (ack).
Cette routine rassemble i2c_write(c) et reception_ack()
unsigned char i2c_write_with_ack(unsigned char c) ;

2.5.11 i2c_read

Le maître lit le caractère envoyé par l'esclave.
unsigned char i2c_read(void) ;

2.5.12 i2c_read_with_ack

Le maître lit le caractère envoyé par l'esclave et envoie l'acquittement (ack).
unsigned char i2c_read_with_ack(unsigned char ack) ;

2.5.13 Exemple du Module I2C

Pour compilar tu as besoin des commandes suivantes
Exemple de compilation : ./compila.sh ej_i2c 16f877a
ej_i2c.c

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

2.6 Module ADC

Auteur : Santiago Gonzalez
Courrier : santigoro en gmail.com
Comprendre la conversion Analogique-Numérique : ici
(http://es.wikipedia.org/wiki/Conversion_analogica-digital)
Voici les fonctions pour travailler avec le module ADC 10 bits interne des PIC16F87X.

2.6.1 adc_init

Initialise le module ADC avec la configuration établie dans les paramètres d'entrée ; fréquence d'oscillateur et configuration d'entrée. Le module ADC restera configuré et prêt à l'usage, mais il sera activé jusqu'à ce qu'on ouvre un canal pour éviter une consommation non nécessaire de courant.

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

set_fosc Establit la frequence de adc.

Les options possibles sont :

FRÉQUENCE DE L'OSCILLATEUR ADC :

FOSC_DIV_2 Fréquence de l'oscillateur du PIC / 2
FOSC_DIV_4
FOSC_DIV_8
FOSC_DIV_16
FOSC_DIV_32
FOSC_DIV_64
FOSC_RC Oscillateur interne du module ADC
set_channel Définit quels broches (pins) on utilisera comme entrées analogiques ou comme Vref externe. Quand on définit seulement une broche (pin) comme tension de référence, ce sera Vref, référence positive et elle sera en RA3, quand on définit 2 entrées comme Vref la positive sera RA3 et la négative RA2.

Les options possibles sont :

CONFIGURATION DES ENTRÉES ET Vref(PCFG en ADCON1) A8_R0 // 8 entrées analogiques y 0 comme Vref (Vref+ = Vdd , Vref- = Ground)
A7_R1 // 7 entrées analogiques y Vref+ en AN3
A5_R0
A4_R1
A3_R0
A2_R1
A0_R0
A6_R2 // 6 entradas analogiques, Vref+ en AN3 et Vref- en AN2
A6_R0
A5_R1
A4_R2
A3_R2
A2_R2
A1_R0
A1_R2
Consulter le Datasheet du pic utilisé pour voir les canaux disponibles, on ne doit pas initialiser les canaux non implantés dans le modèle du pic à utiliser, par exemple, le pic16f876a n'a que 5 canaux ADC, ne jamais utiliser A6_R0 dans ce pic.

2.6.2 adc_set_channel

Établit la configuration des broches d'entrée ADC et Vref

void adc_set_channel(unsigned char set_c29 :hannel) ;

set_channel Définit quels broches (pins) on utilisera comme entrées analogiques ou comme Vref externe.

Quand on définit seulement une broche (pin) comme tension de référence, ce sera Vref, référence positive et sera en RA3, quand on définit 2 entrées comme Vref la positive sera RA3 et la négative RA2.

Les options possibles sont les mêmes que dans adc_init : A2_R1 , etc.

2.6.3 adc_open

Ouvre le canal sélectionné et active le module ADC

void adc_open(unsigned char channel) ;

channel établit le canal ADC a lire

Les options possibles sont :
CHANNEL_0
CHANNEL_1
CHANNEL_2
CHANNEL_3
CHANNEL_4
CHANNEL_5
CHANNEL_6
CHANNEL_7

2.6.4 adc_read

Lit un canal préalablement ouvert

unsigned int adc_read(void) ;

2.6.5 adc_close

Ferme le module ADC.

void adc_close(void) ;

La configuration reste comme on l'a initialisé la dernière fois, on peut réactiver le module ADC au moyen de adc_open(channel), sans exécuter adc_init()

2.6.6 Exemple de Module ADC

pour compiler tu as besoin des commandes suivantes Exemple de compilation : ./compila.sh ej29 :_adc_87Xa.c 16f877a

ej_adc_87Xa.c

```
/*
Exemple d'utilisation des fonctions du module ADC.
validé pour la famille 16f87Xa,
On utilise le canal 0,
connecter les tensions à convertir à RA0,
connecter une led ou une autre sortie à RB7.
RB7 se mettra à l'état haut (Vdd) quand la tension en RA0 sera supérieur à 1/2 Vdd.
*/
#include <pic/p16f877a.h>
#include <adc.h>
int main(void)
{
TRISAbits.TRISA0 = 1; // A0 comme entrée
TRISBbits.TRISB7 = 0; // B7 comme sortie
adc_init( FOSC_DIV_32, A1_R0); // Initialise le module ADC
adc_open(CHANNEL_0); // Ouvre le canal 029: ADC
while (1)
{
if (adc_read() > 512 )
        PORTBbits.RB7 = 1; // Allumer la led
else
```

```
        PORTBbits.RB7 = 0; // Éteindre la led
      }
}
```

2.7 Module Comparateurs

Auteur : Santiago Gonzalez

Courrier : santigoro en gmail.com

Comprendre la conversion Analogique-Numérique : ici

(http://es.wikipedia.org/wiki/Conversion_analogica-digital)

Voici les fonctions pour travailler avec les deux comparateurs internes des PIC16F87X.

2.7.1 comp_init

Initialise le module comparateurs

void comp_init(unsigned char set_config)) ;

set_config Definit les entrées analogiques et Vref, Les options possibles sont :
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 multiplexad
COMP_OFF

2.7.2 comp_set_multiplex

Établit que les entrées sont connectées dans le mode multiplexé

comp_set_multiplex(unsigned char set_switch) ;

set_switch détermine que les canaux sont multiplexés.

Les options possibles sont :
RA0_RA1 RA2_RA3

2.7.3 comp_set_vref

cette fonction change la valeur de Vref interne, si préalablement on a initialisé l'usage de Vref interne avec : comp_vref_mode().

void comp_set_vref(unsigned char set_vref) ;

set_vref détermine la valeur de Vref interne, en poucentage de Vpp, par exemple : comp_Vref_mode(RA0_RA1, VREF_65) active les entrées RA0 et RA1 avec Vref interne égal à 65% de Vpp. Les valeurs sont valides entre VREF_0 y VREF_75 , bien que dans la pratique on n'obtiendra pas de valeurs supérieures à 70% de Vdd et ce module n'est capable de générer que 30 valeurs distinctes de Vref, ainsi le pourcentage sélectionné arrondit au valeur la plus proche. La précision est plus mauvaise au dessous de 25% de Vdd. Pour plus de details consulter le datasheet du pic utilisé.

Pour vérifier le fonctionnement et l'exactitude de Vref interne on peut activer le bit 6 CVROE de CVRCON, Vref restant connectée à RA2 : CVRCONbits.CVROE = 1 ; //maintenant Vref est présent en RA2

Cette sortie peut s'utiliser comme source variable de tension (D-A), bien que de haute impédance, pour obtenir une plus grande capacuté de courant on peut utiliser un amplificateur opérationnel comme suiveur de tension

2.7.4 comp_inv

Inverse ou non les sorties des comparateurs

void comp_inv(unsigned char set_inv) ;

set_inv definit si les sorties des comparateurs s'inversent ou non

Les options possibles sont :
NO_INV
C1_INV
C2_INV
C1_C2_INV

2.7.5 comp1_read

Cette fonction lit la sortie du comparateur 1, les valeurs possibles sont 0 ou 1.

void comp1_read() ;

2.7.6 comp2_read

Cette fonction lit la sortie du comparateur 2, les valeurs possibles sont 0 ou 1.

void comp2_read() ;

2.7.7 comp_reset

Cette fonction remet à zéro (reset) le module Comparateurs, le module reste actif mais la lecture sera de 0.

void comp_reset(void) ;

2.7.8 comp_off

Cette fonction éteint le module Comparateurs, évitant la consommation inutile de courant.

void comp_off(void) ;

2.7.9 Exemple du Module Comparateurs

pour compiler tu as besoin des commandes suivantes

Exemple de compilation : ./compila.sh ej_comp_87Xa.c 16f877a

ej_comp_87Xa.c

```
/*
     Exemple d'utilisation des fonctions du module comparateurs.
  *
     valide pour la famille 16f87Xa,
  *
     En cet exemple on utilise le comparateur 1 con Vref interne al 50\% de Vdd,
  *
     connecter la tension à comparer à RA0,
  *
     connecter une led ou une autre sortie à RB7.
  *
     RB7 se mettra à l'état haut (Vdd) quand la tension en RA0 sera inférieure à 50\% de Vdd
```

```
  *
*/
\#include <pic/p16f877a.h>
\#include <comp.h>
int main(void)
{
     TRISAbits.TRISA0 = 1;           // A0 comme entée
     TRISAbits.TRISA3 = 1;           // A3 comme entrée
     TRISBbits.TRISB7 = 0;           // B7 como sortie
     comp\_init( TWO\_COMP\_MULTI4\_VREF ); // Initialise les comparateurs en modo multiplexé con V
     comp\_set\_vref( VREF\_50 );         // Establece Vref en 50\% Vdd
     while (1)
     {

         if ( comp1\_read() == 1 )    // lee comparador 1 = comprueba bit C1OUT de regist
             PORTBbits.RB7 = 1;      // Enciende led
         else
             PORTBbits.RB7 = 0;      // Apaga led
     }
}
```

2.8 module interruption

Ce module génère les fonctions nécessaires pour travailler avec les interruptions #include ¡interrupt.h¿

2.8.1 _ISR_MAIN

_ISR_MAIN Définit la fonction interruption principal, cette fonction s'exécutera quand arrivera n'importe quelle interruption. On peut choisir n'importe quel nom pour la fonction interruption principal., on l'utilise ainsi :

```
#include <interrupt.h>

_ISR_MAIN void nombre_de_la_funcion(void);
//Cette fonction s'exécutera quand arrivera n'importe quelle interruption.
//Dans son code, on devra vérifier les drapeaux (FLAG), pour savoir
//quelle interruption est arrivé.
void nombre_de_la_funcion(void)
 {
  //code générique
  //quand tu termines tu dois remettre à zéro le drapeau (FLAG)
  //de l'interruption que tu as utilisé
 }
int main(void)
 {
  //Dans la fonction principal on devra autoriser l'interruption GLOBAL
 }
```

2.8.2 ISR_MAIN

ISR_MAIN Definit la fonction/code interruption principal. Cette fonction/code s'exécutera quand arrivera n'importe quelle interruption.

On peut choisir n'importe quel nom pour la fonction interruption principal., on l'utilise ainsi :

```
#include <interrupt.h>
//Cette fonction s'exécutera quand arrivera n'importe quelle interruption.
```

```
//Dans son code, on devra vérifier les drapeaux (FLAG), pour savoir
//quelle interruption est arrivé.
ISR_MAIN(nombre_de_la_funcion)
{
//code générique
//quand tu termines tu dois remettre à zéro le drapeau (FLAG)
//de l'interruption que tu as utilisé
}
int main(void)
{
//Dans la fonction principal on devra autoriser l'interruption GLOBAL
}
```

2.8.3 enable_int_global

Autorise globalement toutes les interruptions

void enable_int_global(void) ;

exemple : enable_int_global() ;

2.8.4 disable_int_global

Inhibe globalmente toutes les interruptions

void disable_int_global(void) ;

exemple : disable_int_global() ;

2.8.5 interruption Externe

Les fonctions pour travailler avec l'interruption externe (RB0) sont :

2.8.5.1 enable_int_ext

Autorise l'interruption externe

#define H_TO_L 0

#define L_TO_H 1

void enable_int_ext(BYTE flanco) ;

exemple : enable_int_global(L_TO_H) ;

H_TO_L indique une interruption activée sur front descendant

L_TO_H indique une interruption activée sur front montant

2.8.5.2 disable_int_ext

Inhibe l'interruption externe

void disable_int_ext(void) ;

En inhibant l'interruption on ne modifie pas l'autorisation de l'interruption globale

2.8.5.3 int_ext_flag

Retourne l'état actuel du drapeau (flag) indicateur de l'interruption externe

BYTE int_ext_flag(void) ;

L'usage de cette fonction ne modifie pas le drapeau (flag)

2.8.5.4 int_ext_free_flag

Remise à zéro du drapeau (flag) indicateur d'interruption externe

void int_ext_free_flag(void) ;

L'usage de cette fonction modifie le drapeau (flag), le mettant à 0

2.8.6 interruption TIMER0

Les fonctions pour travailler avec l'interruption par débordement du TIMER0 sont :

2.8.6.1 enable_int_timer0

autorise l'interruption par débordement du TIMER0

void enable_int_timer0(void) ;

Pour que cette interruption soit autorisé il faut aussi activer l'interruption globale

2.8.6.2 disable_int_timer0

inhibe l'interruption par desbordement du TIMER0

void disable_int_timer0(void) ;

En inhibant l'interruption on ne modifie pas l'autorisation de l'interruption globale

2.8.6.3 int_timer0_flag

Retourne l 'état actuel du drapeau (flag) indicateur de l'interruption par desbordement du TIMER0

BYTE int_timer0_flag(void) ;

L'usage de cette fonction ne modifie pas le drapeau (flag)

2.8.6.4 int_timer0_free_flag

Remise à zéro du drapeau (flag) indicateur de l'interruption par débordement du TIMER0

void int_timer0_free_flag(void) ;

L'usage de cette fonction modifie le drapeau (flag), le mettant à 0

2.8.7 interruption RB4 à RB7

Les fonctions pour travailler avec l'interruption par changement d'état des broches (pins) RB4 a RB7 sont :

2.8.7.1 enable_int_rb4to7

autorise l'interruption par changement d'état des broches (pins) de RB4 à RB7

void enable_int_rb4to7(void) ;

Pour que cette interruption soit autorisée il faut aussi activer l'interruption globale

2.8.7.2 disable_int_rb4to7

inhibe l'interruption par changement d'état des broches (pins) de Rb4 à RB7

void disable_int_rb4to7(void) ;

En inhibant l'interruption on ne modifie pas l'autorisation de l'interruption globale

2.8.7.3 int_rb4to7_flag

Retourne l 'état actuel du drapeau (flag) indicateur de l'interruption par changement d'état des broches (pins) de RB4 à RB7

BYTE int_rb4to7_flag(void) ;

L'usage de cette fonction ne modifie pas le drapeau (flag)

2.8.7.4 int_rb4to7_free_flag

Remise à zéro du drapeau (flag) indicateur de l'interruption par changement d'état des broches (pins) de RB4 à RB7

void int_rb4to7_free_flag(void) ;

L'usage de cette fonction modifie le drapeau (flag), le mettant à 0

2.8.8 interruption Rx UART

Les fonctions pour travailler avec l'interruption par réception de caractère dans l 'UART sont :

2.8.8.1 enable_int_uart

autorise l'interruption par réception de caractère dans l'UART

void enable_int_uart(void) ;

Pour que cette interruption soit autorisée il faut aussi activer l'interruption globale

2.8.8.2 disable_int_uart

inhibe l'interruption par réception de caractère dans l'UART

void disable_int_uart(void) ;

En inhibant l'interruption on ne modifie pas l'autorisation de l'interruption globale

2.8.8.3 int_uart_flag

Retourne l 'état actuel du drapeau (flag) indicateur de l'interruption par réception de caractère dans l 'UART

BYTE int_uart_flag(void) ;

L'usage de cette fonction ne modifie pas le drapeau (flag), en plus on ne peut pas.

2.8.8.4 int_uart_free_flag

Cette fonction n'existe pas. Le drapeau (flag) se remettra à zéro lors de la lecture avec la fonction uart_getc du caractère entrant, cette fonction doit être placé à l'intérieur de la fonction interruption sinon le drapeau (flag) ne se remettra jamais à 0 et on ne sortira jamais de la fonction interruption, car la fonction interruption sera invoqué jusqu'à la fin des temps.

2.8.9 interruption ADC

Les fonctions pour travailler avec l'interruption qui donne la fin de la conversion analogique numérique sont :

2.8.9.1 enable_int_adc

autorise l'interruption de fin de conversion analogique numérique

void enable_int_adc(void) ;

Pour que cette interruption soit autorisée il faut aussi activer l'interruption globale

2.8.9.2 disable_int_adc

inhibe l'interruption de fin de conversion analogique numérique

void disable_int_adc(void) ;

En inhibant l'interruption on ne modifie pas l'autorisation de l'interruption globale

2.8.9.3 int_adc_flag

Retourne l 'état actuel du drapeau (flag) indicateur de l'interruption de fin de conversion analogique numérique

BYTE int_adc_flag(void) ;

L'usage de cette fonction ne modifie pas le drapeau (flag).

2.8.10 interruption Comparateur

Les fonctions pour travailler avec l'interruption de fin de comparaison sont :

2.8.10.1 enable_int_comp

autorise l'interruption de fin de comparaison

void enable_int_comp(void) ;

Pour que cette interruption soit autorisée il faut aussi activer l'interruption globale

2.8.10.2 disable_int_comp

inhibe l'interruption de fin de comparaison

void disable_int_comp(void) ;

En inhibant l'interruption on ne modifie pas l'autorisation de l'interruption globale

2.8.10.3 int_comp_flag

Retourne l 'état actuel du drapeau (flag) indicateur de l'interruption de fin de comparaison

BYTE int_comp_flag(void) ;

L'usage de cette fonction ne modifie pas le drapeau (flag).

2.8.11 Exemple du module Interruption Externe

Pour compiler tu as besoin des commandes suivantes

Exemple de compilation : ./compila.sh ej_intext 16f877a

ej_intext.c

```
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 module TIMER0

Auteur : Santiago Gonzalez

Courrier : santigoro en gmail.com

Voici les fonctions pour travailler avec le TIMER0 des PIC16F87X.

2.9.1 timer0_init

Initialise TIMER0 avec l'horloge interne (mode timer) ou externe (mode compteur).

void timer0_init(unsigned char intern_extern) ;

intern_extern Établit si on utilise une horloge interne ou une horloge/stímuli externe (modo compteur).
TIMER_INTERN :Utilise l'horloge interne.
COUNTER_EXTERN :Utilise l'horloge/stimuli externe (mode compteur)

timer0_init(TIMER_INTERN) ;

Met à 0 le compteur du timer0 y efface le drapeau (flag) des interruptions. Le compteur s'incrémente à chaque cycle d'instructions (Freq.Osc. / 4) si on n'utilise pas le pré-diviseur (prescaler.)

2.9.2 timer0_set_edge

Indique si l'incrément du compteur se produit sur le front montant ou descendant (mode compteur).

void timer0_set_edge(unsigned char set_risfal) ;

Met à 0 le compteur du timer0 y efface le drapeau (flag) des interruptions Le compteur s'incrémente sur le front montant ou descendant présent sur la broche (pin) RA4 si on n'utilise pas le pré-diviseur (prescaler.)

set_risfal Établit si l'incrément du compteur se produit sur le front montant ou descendant

Les options possibles sont :

RIS_EDGE FAL_EDGE

Où RIS_EDGE indique l'incrément du compteur sur front montant (rising) et FAL_EDGE sur front descendant (falling).

timer0_set_edge(FAL_EDGE) ;

2.9.3 timer0_set_prescaler

Assigne le pré-diviseur (prescaler) du timer0 , ceci annule l'usage du prédiviseur (prescaler) du chien de garde (watchdog) et établit la valeur du pré-diviseur (prescaler.)

void timer0_set_prescaler(unsigned char set_presc) ;

set_presc Établit la valeur du pré-diviseur (prescaler.)

Les options possibles sont :

// Fréquence du cycle d'instructions
PRESC_DIV_2 // (FOSC/4) del PIC divisé par 2
PRESC_DIV_4 // (FOSC/4) del PIC divisé par 4
PRESC_DIV_8
PRESC_DIV_16
PRESC_DIV_32
PRESC_DIV_64
PRESC_DIV_128
PRESC_DIV_256
PRESC_OFF // assigné au chien de garde (watchdog)

N'importe quelle opération d'écriture dans le registre TMR0 met automatiquement à 0 le compteur du pré-diviseur (prescaler), bien qu'il garde sa configuration.

2.9.4 timer0_write

Écrit la valeur du compteur TMR0.
void timer0_write(unsigned char set_count) ;
set_count Écrit la valeur du compteur TMR0.

2.9.5 timer0_read

Lit la valeur du compteur TMR0.
#define timer0_read() TMR0

2.9.6 Exemple module TIMER0

Pour compiler tu as besoin des commandes suivantes

Exemple de compilation : ./compila.sh ej_timer0.c 16f877a

ej_timer0.c

```
/* Exemple d'utilisation des fonctions du module TIMER0,
 valide pour la famille 16f87Xa,
Dans cet exemple on utilise le timer0 comme compteur,
connecter un bouton poussoir entre RA4 y 0V
et une résistance de tirage vers le haut (pullup)
de 5 Kohms entre RA4 y Vdd (+5v),
connecter une led ou une autre sortie à RB7.
RB7 se mettra à l'état haut (Vdd) quand on appuie 4 fois. */
#include <pic/p16f877a.h>
#include <timer0.h>

int main(void)
 {
   TRISAbits.TRISA4 = 1;
   TRISBbits.TRISB7 = 0;
   PORTBbits.RB7 = 0;
   timer0_init(COUNTER_EXTERN);
   // Initialise timer0 mode compteur (horloge ou stímuli externe en RA4)
   timer0_set_edge(FAL_EDGE); // indique l'incrément du compteur sur front descendant
   timer0_set_prescaler(PRESC_DIV_2);
   // établit  le pré-diviseur (prescaler) en fréquence d'instructions / 2
   while (timer0_read() < 2); // Attends ici tant que le compteur du timer < 2
   PORTBbits.RB7 = 1;
 }
```

2.10 module timer2

Voici les fonctions pour travailler avec le TIMER2 des PIC16F87X.

2.10.1 timer2_init

Initialise le timer2.

void timer2_init(void) ;

Met à 0 le compteur du timer2 et efface le drapeau (flag) des interruptions. Le compteur s'incrémente à chaque cycle d'instructions (Freq.Osc. / 4) si on n'utilise pas le pré-diviseur (prescaler).

2.10.2 timer2_set_period

Établit la période du timer2.

void timer2_set_period(unsigned char set_period) ;

set_period Établit la période du timer2, cela doit être une valeur entre 0 et 255.

2.10.3 timer2_set_prescaler

Établit la valeur du prédiviseur (prescaler) du timer2.

void timer2_set_prescaler(unsigned char set_presc) ;

set_presc Établit la valeur du prédiviseur (prescaler)

Les options possibles sont : PRESC_DIV_1 PRESC_DIV_4 // Fréquence du cycle d'instructions (FOSC/4) del PIC / 4 PRESC_DIV_16

2.10.4 timer2_set_postscaler

Établit la valeur du post-diviseur (postscaler) del timer2.

void timer2_set_postscaler(unsigned char set_postsc) ;

set_postsc Établit la valeur du post-diviseur (postscaler)

Les options possibles sont :
POSTSC_DIV_1
POSTSC_DIV_2 // TMR2IF à l'état haut chaque 2 débordements 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 sortie du post-diviseur (postscaler) met à 1 le drapeau (flag) des interruptions du timer2 (TMR2IF).

2.10.5 timer2_write

Écrit la valeur du registreTMR2.
void timer2_write(unsigned char set_count) ;
set_count Écrit la valeur du registreTMR2..

2.10.6 timer2_read

Lit a valeur du registreTMR2.
#define timer2_read() TMR2

2.10.7 Exemple d'utilisation du timer2

Pour compiler tu as besoin des commandes suivantes

Exemple de compilation : ./compila.sh ej_timer2.c 16f877a

ej_timer2.c

```
ej_timer2.c
/*
Exemple d'utilisation des fonctions du module TIMER2.
valide pour la famille 16f87Xa,
Dans  cet exemple on utilise le timer2 avec un compteur par logiciel,
pour faire une led clignotante:
connecter une led ou autre sortie à RB7.
RB7 se mettra à l'état haut (Vdd) durant approximativement 1 seconde avec une horloge de 4 MHz, et à
*/
#include <pic/p16f877a.h>
#include <timer2.h>
int main(void)
 {
   unsigned char compteur;
   TRISBbits.TRISB7 = 0;
   PORTBbits.RB7 = 0;
   timer2_init();                          // Initialise le timer2
   timer2_set_prescaler(PRESC_DIV_16);     // Établit le pré-diviseur en fréquence d'instructions
   timer2_set_period(255);                 // Établit la période à 255
   compteur = 0;
   bucle:
   while (timer2_read() < 255);            // Attend ici tant que le compteur du timer <255
   compteur = compteur + 1;                // Incrémente le compteur
   if (compteur == 255)                    // Si compteur = 255 inverse l'état de RB7
     {
       if (PORTBbits.RB7 == 0)
           PORTBbits.RB7 = 1;
       else
           PORTBbits.RB7 = 0;
       compteur = 0;                     // Reinitialise le compteur
     }
   goto bucle;
 }
```

2.11 module system

On trouve ici quelques fonctions de configuration

#include ¡system.h¿

2.11.1 sleep

Met le microcontrôleur en sommeil.

void sleep(void) ;

2.11.2 ASM

fonction qui inclut du code ASSEMBLER

void ASM(char comando[]) ;

2.11.3 Exemple System

Pour compiler tu as besoin des commandes suivantes

Exemple de compilation : ./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;
   //Port B comme sortie
   ASM("BANKSEL TRISB");
   ASM("MOVLW 0x00");
   ASM("MOVWF TRISB");
   //0xF0 --> PORTB
   ASM("BANKSEL F_REG");// la donnée se trouve dans la banque 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 trouve en la banque de F_REG
   ASM("MOVWF %0":"=v" (res));
   //Quand tu finis d'écrire en assembleur
   // tu dois toujours le quitter dans la banque de F_REG,
   //dans le cas contraire le programme ne fonctionnera pas..
   //Je mets le micro en sommeil
   sleep();
   return 0;
 }
```

Chapitre 3

Bibliothèque de Drivers

Dans cette bibliothèque on trouve tous les modules pour pouvoir contrôler les composants externes, c'est la partie avec le plus grand potentiel pour l'apport de programmeurs externes.

3.1 KEYPAD 4x4

Quand on désire connecter un clavier (keypad) de 4x4 avec le port D ou B, il suffit d'ajouter la bibliothèque suivante

pour utiliser le port D

```
#define PORTD_FOR_KEYPAD4X4
#include <keypad4x4.h>
```

pour utiliser le port B

```
#define PORTB_FOR_KEYPAD4X4
#include <keypad4x4.h>
```

3.1.1 Diagramme des Broches (Pines)

Les claviers KeyPad sont un ensemble de boutons poussoirs disposés en forme de matrice de 4 lignes et 4 colonnes, ces dispositifs son utilisés pour introduire une information au microcontrôleur.

```
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)

Cette fonction n'est pas bloquante et renvoie une variable de type char avec la valeur de la touche appuyée, si on ne trouve pas de touche appuyée ou si on trouve plus d'une touche on renvoie 0

3.1.3 kbd_getchar

char kbd_getchar(void)

Cette fonction est similaire à kbd_get mais bloquante elle retourne aussi une variable de type char avec la valeur de la touche appuyée

3.1.4 Exemple de KeyPad4x4

pour compiler vous avez besoin des commandes suivantes :

Exemple de compilation : ./compila.sh ej_keypad4x4 16f877a

ej_keypad4x4.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

Quand on désire connecter un clavier (keypad) de 4x4 avec n'importe quel broche (PIN), il suffit d'ajouter la bibliothèque suivante

```
#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 Diagramme des Broches (Pines)

Les claviers KeyPad sont un ensemble de boutons poussoirs disposés en forme de matrice de 4 lignes et 4 colonnes, ces dispositifs son utilisés pour introduire une information au microcontrôleur.

```
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)

Cette fonction n'est pas bloquante et renvoie une variable de type char avec la valeur de la touche appuyée, si on ne trouve pas de touche appuyée ou si on trouve plus d'une touche on renvoie 0

3.2.3 kbd_getchar

char kbd_getchar(void)

Cette fonction est similaire à kbd_get mais bloquante elle retourne aussi une variable de type char avec la valeur de la touche appuyée

3.2.4 Exemple de KeyPad4x4 Flex

pour compiler vous avez besoin des commandes suivantes :

```
Exemple de compilation : ./compila.sh ej_keypad4x4flex 16f877a
```

ej_keypad4x4flex.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;
}
```

Chapitre 4

Bibliothèque de Pics

Ici on trouvera les définitions de tous les registres du PIC, les registres peuvent être traités de deux façons

comme variable :

```
PORTA=0xff;
X=PORTA;
```

comme structure :

```
PORTAbits.RA0=1;
b=PORTAbits.RA0;
```

4.1 Pic16f877a

pour des définitions exclusives pour le pic 16f877a, on doit inclure.

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

4.2 Exemple de Pic

pour compiler vous avez besoin des commandes suivantes :

```
Exemple de compilation : ./compila.sh ej_pic 16f877a
```

ej_pic.c

```
#include <pic/p16f877a.h>
int main(void)
{
        TRISBbits.TRISB0=0;

        while(1)
        {
                PORTBbits.RB0=1;
                PORTBbits.RB0=0;
        }
        return 0;
}
```

Chapitre 5

Bibliothèque d'Utilitaires

C'est un ensemble de bibliothèques d'utilitaires d'usage générique, dont l'utilisation est indépendante du type de PIC

5.1 Delayms

Routine qui génère un retard en ms

Auparavant il faut définir la valeur de la fréquence du quartz

```
#define FOSC_HZ 20000000
#include <delayms.h>
```

5.1.1 delayms

La fonction génère un retard en millisecondes :

void delayms(unsigned int retraso) ;

5.1.2 Exemple de DelayMs

pour compiler vous avez besoin des commandes suivantes :

```
 Exemple de compilation :./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 Module Mémoire RAM

Cette bibliothèque sert pour interroger la mémoire RAM libre dans le PIC. Les fonctions nous aideront à connaître la mémoire libre dans chaque banque.

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

5.2.1 memory_bank0

La fonction retourne la quantité d'octets (bytes) (RAM) libres dans la BANQUE 0.

Par défaut cette fonction est autorisée, du fait de la macro I_HAVE_BANK0.

BYTE memory_bank0(void) ;

5.2.2 memory_bank1

La fonction retourne la quantité d'octets (bytes) (RAM) libres dans la BANQUE 1.

La fonction doit être autorisée avec la macro I_HAVE_BANK1.

BYTE memory_bank1(void) ;

5.2.3 memory_bank2

La fonction retourne la quantité d'octets (bytes) (RAM) libres dans la BANQUE 2.

La fonction doit être autorisée avec la macro I_HAVE_BANK2.

BYTE memory_bank2(void) ;

5.2.4 memory_bank3

La fonction retourne la quantité d'octets (bytes) (RAM) libres dans la BANQUE 3. La fonction doit être autorisée avec la macro I_HAVE_BANK3.

BYTE memory_bank3(void) ;

5.2.5 memory_bank_all

La fonction retourne la quantité d'octets (bytes) (RAM) libres dans toutes les BANQUES.

La fonction a besoin que l'on autorise avec la macro I_HAVE_BANK0,I_HAVE_BANK1, etc. Les banques qui sont utilisées.

BYTE memory_bank_all(void) ;

5.2.6 Exemple de Mémoire

pour compiler les commandes suivantes sont nécessaires :

Exemple de compilation : ./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 Module UART2

Cette bibliothèque nous aidera à convertir les nombres binaires au format texte dans une représentation DÉCIMAL y HEXADÉCIMAL.

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

5.3.1 puth

transforme le chiffre hexadécimal (un demi octet) en 1 caractère ASCII et l'envoie.

Par exemple si tu as l'hexadécimal 0x4A et tu prends le nombre "A" (soit 10 en décimal - 1010 en binaire), "puth" le transforme en caractère 'A' et l'envoie.

Transforme le nombre "0" en caractère '0'.
Transforme le nombre "1" en caractère '1'.
Transforme le nombre "2" en caractère '2'.
...
Transforme le nombre "A" en caractère 'A'.
Transforme le nombre "B" en caractère 'B'.
...
Transforme le nombre "F" en caractère 'F'.
Le caractère généré est envoyé utilisant la function miputc.

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

5.3.2 puthex

transforme un nombre hexadécimal (un octet) en 2 caractères ASCII y l'envoie.

Par exemple, si tu as l'hexadécimal 0x4A, puthex prend le nombre "4" le transforme en caractère '4' et l'envoie avec miputc, prend le nombre "A" le transforme en caractère 'A' et l'envoie avec miputc.

$0x4A \Rightarrow miputc('4');$
miputc('A') ;

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

5.3.3 putint

transforme un nombre entier de 16 bits en utilisant sa représentation hexadécimale de 4 caractères ASCII et ensuite les envoie un par un.

Par exemple, si tu as le nombre 1030, putint le transforme en HEXADECIMAL "0x0406", putint prend '0','4','0','6' et les envoie avec miputc.

$1030 \Rightarrow 0x0406 \Rightarrow miputc('0');$
miputc('4') ;
miputc('0') ;
miputc('6') ;
void putint(void (*miputc)(char),int num) ;

5.3.4 geth

Lit un caractère ASCII et le transforme en un chiffre hexadécimal. Par exemple si tu lis le caractère "A" , "geth" le transforme en le nombre 10 en décimal codé en hexadécimal.

Transforme le caractère '0' en nombre 0.
Transforme le caractère '1' en nombre 1.
Transforme le caractère '2' en nombre 2.
...
Transforme le caractère 'a' ou 'A' en nombre 10 (0x0a).
Transforme le caractère 'b' ou 'B' en nombre 11 (0x0b).
...
Transforme le caractère 'f' ou 'F' en nombre 15 (0x0f).
n'importe quel autre nombre est converti en 0xff
Pour obtenir le caractère on utilise la fonction migetc.
char geth(char (*migetc)(void)) ;

5.3.5 gethex

Lit 2 caractères ASCII y les transforma en nombre hexadecimal.

Par exemple, si tu as deux caractères ASCII comme "4" et "A", gethex prend le caractère "4" et le transforme en nombre 0x40 et prend le caractère "A" et le transforme en nombre 0x0A puis les regroupe en 0x4A.

miputc('4') ;
$miputc('A'); \Rightarrow 0x4A$
char gethex(char (*migetc)(void)) ;

5.3.6 getint

Lee 4 caractères ASCII que représentent un nombre en hexadecimal et les transforme en un nombre entier de 16 bit.

Par exemple, si tu as les caractères '0','4','0' y '6', getint retourne le nombre 1030.
miputc('0') ;
miputc('4') ;
miputc('0') ;
miputc('6') ; =¿ 0x0406 =¿ 1030
int getint(char (*migetc)(void)) ;

5.3.7 getd

Lit un caractère ASCII et le transforme en un seul chiffre décimal.

Par exemple si tu lis le caractère "2" , "getd" le transforme en nombre 2 en décimal codé en binaire BCD.

Transforme le caractère '0' en nombre 0.
Transforme le caractère '1' en nombre 1.
Transforme le caractère '2' en nombre 2.
...
Transforme le caractère '9' en nombre 9.
n'importe quel autre nombre est converti en 0xff
Pour obtenir le caractère on utilise la fonction migetc.

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

5.3.8 Ejemplo de UART2

pour compiler les commandes suivantes sont nécessaires :

Exemple de compilation : ./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);
   //laisser un peu de temps pour que le dernier caractère soit envoyé
   uart_close();
   return 0;
 }
```

Chapitre 6

Bibliothèque standard du C

C'est un ensemble de bibliothèques standard de C, le code de ces fonctions est indépendant du type de PIC

6.1 String

Définit la bibliothèque standard de C : Chaîne de caractères String.h.

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

6.1.1 Memchr

Localise la première détection du caractère c (converti en caractère non signé unsigned char) dans les n premiers caractères (chacun interprété comme un caractère non signé unsigned char) de l'objet pointé par s.

La fonction retourne un pointeur sur le caractère localisé, ou un pointeur nulle si le caractère n'apparait pas dans l'objet.

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

6.1.2 Memcmp

Compare les n premiers caractères de l'objet pointé par s1 (interprété comme un caractère non signé unsigned char) avec les n premiers caractères de l'objet pointé par s2 (interprété comme unsigned char).

La fonction retourne un nombre entier supérieur, égal, ou inférieur à zéro, approprié selon que l'objet pointé par s1 est supérieur, égal, ou inférieur à l'objet pointé par s2.

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

6.1.3 Memcpy

Copie les n premiers caractères de l'objet pointé par s2 dans l'objet pointé par s1.

La fonction retourne la valeur de s1. Si en copiant un objet vers l'autre ils se superposent, alors le comportement n'est pas défini.

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

6.1.4 Memmove

Copie les n premiers caractères de l'objet pointé par s2 dans l'objet pointé par s1.

La fonction retourne la valeur de s1. Si en copiant un objet dans l'autre ils se superposent, alors le comportement n'est pas défini.

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

6.1.5 Memset

Copie la valeur de c (convertie en unsigned char) dans chacun des n premiers caractères de l'objet pointé par s.

La fonction retourne la valeur de s.

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

6.1.6 Strcat

Ajoute une copie de la chaîne pointé par s2 (incluant le caractère nul) à la fin de la chaîne pointée par s1. Le caractère initial de s2 récrit le caractère nul à la fin de s1.

La fonction retourne la valeur de s1. Si la copie fait en sorte que les objets se superposent, alors le comportement n'est pas défini

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

6.1.7 Strchr

Localise la première apparition de c (converti en unsigned char) dans la chaîne pointée par s (incluant le caractère nul).

La fonction retourne un pointeur à partir du caractère trouvé. Si il n'a pas trouvé le caractère, c, alors retourne un pointeur nul.

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

6.1.8 Strcmp

Compare la chaîne pointée par s1 avec la chaîne pointée par s2.

La fonction retourne un nombre entier supérieur, égal, ou inférieur à zéro, approprié selon que l'objet pointé par s1 est supérieur, égal, ou inférieur à l'objet pointé par s2.

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

6.1.9 Strcpy

Copia la chaîne pointée par s2 (incluant le caractère nul) à la chaîne pointée par s1.

La fonction retourne la valeur de s1. Si en copiant une chaîne sur l'autre elles se superposent, alors le comportement n'est pas défini.

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

6.1.10 Strcspn

Compte le nombre de caractères d'une sous-chaîne initiale pointée par s1 qui ne contient aucun des caractères de la chaîne pointée par s2.

La fonction retourne le nombre de caractères lus de la sous-chaîne jusqu'à ce qu'il trouve quelques un des caractères de s2. Le caractère nul n'est pas compté.

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

6.1.11 Strerror

Convertit le nombre d'erreur en errnum dans un message d'erreur (une chaîne de caractères).

La fonction retourne la chaîne de caractères contenant le message associé avec le nombre d'erreur. Cette conversion y le contenu du message dépendent de l'implémentation. La chaîne ne sera pas modifiée par le programme, mais si elle peut être récrite avec un autre appel à la même fonction.

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

6.1.12 Strlen

Calcule le nombre de caractères de la chaîne pointée par s.

La fonction retourne le nombre de caractères jusqu'au caractère nul, qui n'est pas inclus.

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

6.1.13 Strncat

Ajoute pas plus de n caractères (un caractère nul et les caractères suivants ne sont pas ajoutés) de la chaîne pointée par s2 à la fin de la chaîne pointée par s1. Le caractère initial de s2 récrit le caractère nul à la fin de s1. Le caractère nul est toujours ajouté au résultat.

La fonction retourne le nombre de caractères jusqu'au caractère nul, qui n'est pas inclus.

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

6.1.14 Strncmp

Compare pas plus de n caractères (les caractères postérieurs au caractère nul ne sont pas pris en compte) de la chaîne pointée par s1 avec la chaîne pointée par s2.

La fonction retourne un nombre entier supérieur, égal, ou inférieur à zéro, approprié selon que l'objet pointé par s1 est supérieur, égal, ou inférieur à l'objet pointé par s2.

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

6.1.15 Strncpy

Copie pas plus de n caractères (caractères postérieurs au caractère nul ne sont pas copiés) de la chaîne pointée par s2 à la chaîne pointée par s1

La fonction retourne la valeur de s1. Si en copiant une chaîne dans l'autre ils se superposent, alors le comportement n'est pas défini. Si le tableau pointé par s2 est une chaîne qui est plus courte que n caractères, alors des caractères nuls son ajoutés à la copie dans le tableau pointé par s1.

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

6.1.16 Strpbrk

Localise la première apparition de la chaîne pointée par s1 de n'importe quel caractère de la chaîne pointée par s2 .

La fonction retourne un pointeur sur le caractère, o un pointeur nul si aucun caractère de s2 n'apparaît dans s1.

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

6.1.17 Strrchr

Localise la dernière apparition de c (converti en unsigned char) dans la chaîne pointée par s (incluant le caractère nul).

La fonction retourne un pointeur à partir du caractère trouvé. Si on n'a pas trouvé le caractère, c, alors on retourne un pointeur nul.

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

6.1.18 Strspn

renvoie la position du premier caractère d' une chaîne que ne coïncide avec aucun des caractères de l'autre chaîne donnée

La fonction renvoie la position du premier caractère d'une chaîne que ne coïncide avec aucun des caractères de l'autre chaîne donnée.

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

6.1.19 Strstr

Recherche une chaîne à l'intérieur de l'autre.

La fonction retourne un pointeur sur la chaîne trouvée, ou un pointeur nul si la chaîne n'est pas trouvée. Si s2 pointe sur une chaîne de longueur zéro, la fonction retourne s1.

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

Chapitre 7

Exemples

Un ensemble d'exemples de tous les drivers et extensions

7.1 Compilation

On peut utiliser les méthodes de compilation vues dans Méthodes de Compilation

./compila.sh ejemplo1 16f877a

7.2 ejemplo1.c

Un exemple simple (ejemplo1.c[1])

```
#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;
}
```

[1] Note de traduction : les noms de fichiers restent en espagnol, pour pouvoir les retrouver dans les dossiers.

Chapitre 8

Références

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

Annexe A

Questions fréquentes

1. Je peux participer au projet Pic Gcc ?
   Oui, entre en contact avec : Pedro
   pjanragu en telefonica.net
2. Je peux participer au projet Pic Gcc Library ?
   Oui, entre en contact avec : Fernando
   fernando.pujaico.rivera en gmail.com
3. Pic Gcc peut compiler d'autres familles de PIC ?
   hmmmm, si tu nous aides, Oui.
4. Quels programmateurs de PIC je peux utiliser ? Tu peux construire ou acheter n'importe quel programmateur, par exemple tu peux construire un programmateur modèle PROPIC II et utiliser n'importe quel logiciel de programmation
   En Linux : PikLab, PikDev, etc.
   En Windows : PicKit2, WinPic800, IcProg
5. Quel Environnement Intégré de Développement je peux utiliser ?
   Tu peux utiliser GtkPicGccIDE - Zandor(http://pic-gcc-library.sourceforge.net/data/?page_id=3)

Notes du traducteur :

Évidemment il faut mieux écrire en espagnol à Pedro ou Fernando, sinon en anglais.

Pour les questions en français per.launay chez free.fr