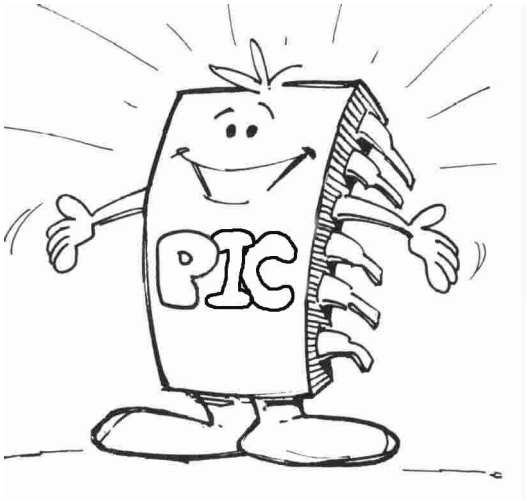
COMPILATEUR

Microchip C18 v16

Cours et exercices



MICROCHIP

www.microchip.com

Christian Dupaty
Lycée Fourcade
13120 Gardanne
Académie d'Aix-Marseille

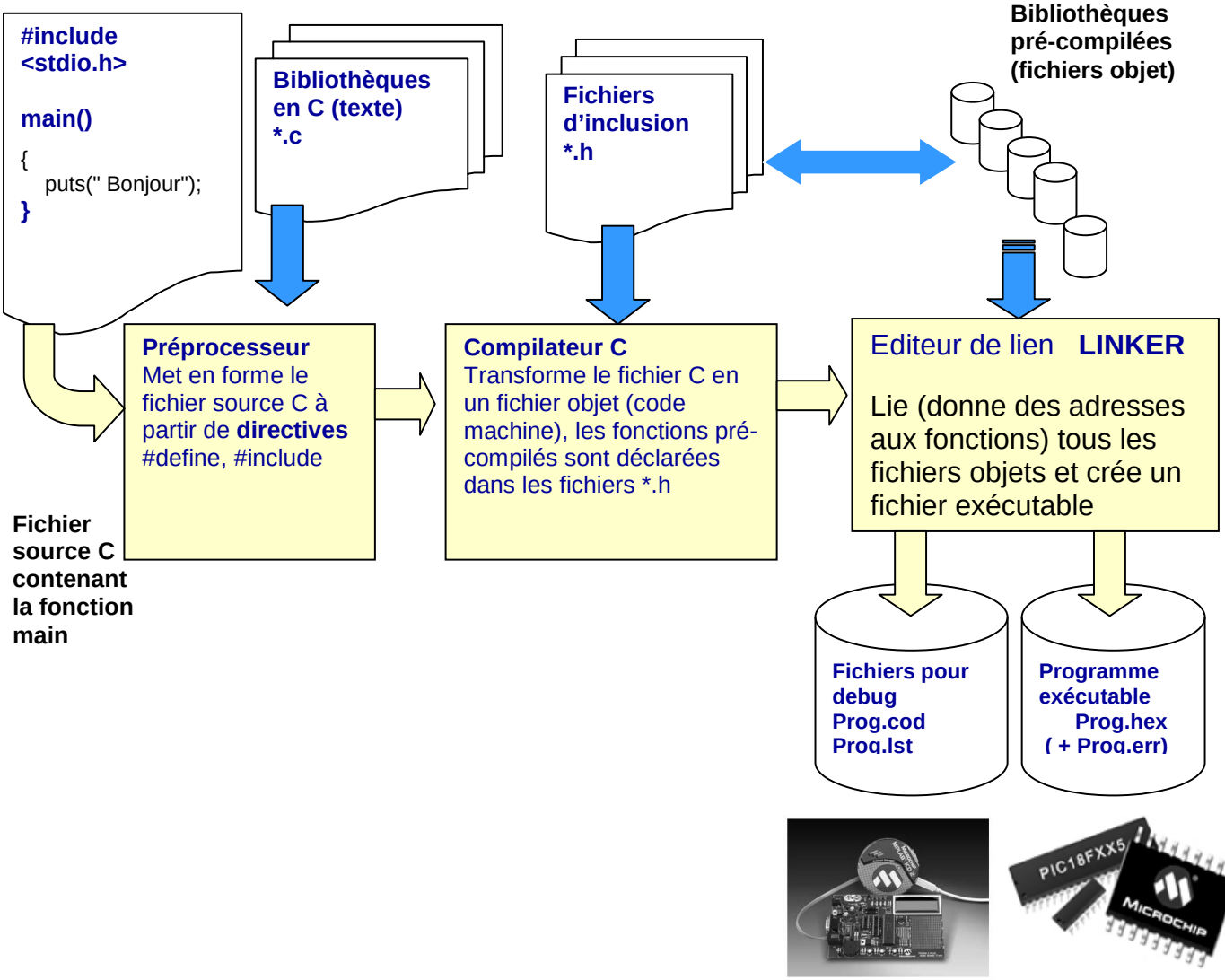
SOMMAIRE

1. Caractéristiques générales de MCC18

1.1. Propriétés

- Compatibilité C ANSI
- Intégrable dans MPLAB pour faciliter la mise au point et la gestion d'un projet
- Génération de modules objet relogeables
- Compatible avec des modules objets générés par MP ASM
- Bibliothèque étendue incluant des modules de gestion des périphériques ; PWM, SPI, …
- Contrôle total par l'utilisateur de l'allocation des données et du code en mémoire

1.2. Schéma général du processus de compilation



1.3. Rôle du pré-processeur

Le pré-processeur ou pré-compilateur réalise des mises en forme et des aménagements du texte d'un fichier source, juste avant qu'il ne soit traité par le compilateur. Il existe un ensemble d'instructions spécifiques appelées **directives** pour indiquer les opérations à effectuer durant cette étape.
Les deux directives les plus courantes sont #define et #include.
#define correspond à une équivalence ex : #define pi 3.14 ou une définition de macro

1.4. Rôle des fichiers d'inclusion (*.h)

Les fichiers d'inclusion ou d'en tête ***.h** (*header*) contiennent pour l'essentiel cinq types d'informations :

- Des définitions de nouveau type
- Des définitions de structure
- Des définitions de constantes
- Des déclarations de fonctions
- Des définitions de macro_fonctions →

Exemple :
#define add(a,b) a+b

En général ces fichiers contiennent des directives de compilation ou pré_compilation conditionnelles. De ce fait ils ne sont pas toujours aisés à déchiffrer pour une personne qui débute en langage C. néanmoins il est indispensable d'en prendre petit à petit connaissance.

Il s'agit d'un fichier d'inclusion particulièrement important lorsqu'on travaille en C sur un micro-contrôleur : le fichier de définition des registres internes du micro-contrôleur **p18fxxx.h** (les xxx sont à remplacer par le numéro du microcontrôleur : p18f4620.h) .

p18fxxx.h possède les définitions des registres et des bits ce qui permet d'accéder directement aux registres du µcontrôleur par leur nom **(ceux du data sheet)** et également de tester ou positionner individuellement les bits de ces registres de la façon suivante :**nom_registre.nom_bit**

exemples : ***PORTB=0xA4 ; ou a=PORTB ;***
PORTBbits.RB0=0 ; ou PORTBbits.RB0=1 *;*
If (PORTAbits.RA4==1) … ; else …. ;
Remarque : L'expression sera ***vraie*** *si RA4 est non nul, il est donc inutile d'écrire (==1)*
On pourra écrire : ***if (PORTAbits.RA4) … ; else …. ;***
De même pour tester si RA4==0 ***(faux)*** *on écrira :*
if (!PORTAbits.RA4) … ; else …. ;

Pour inclure un fichier contenant du code source (.c ou .h) dans un autre fichier il faut utiliser la directive **#include** de la façon suivante :

#include<Nomfichier>
recherche du fichier dans :
- Les répertoires mentionnés à l'aide de l'option de compilation /Idirectory
- Les répertoires définis à l'aide de la variable d'environnement INCLUDE

#include "Nomfichier"
recherche du fichier dans :
Idem cas précédent **+**
Le répertoire courant
Il est également possible de préciser le chemin complet du fichier : **#include "c:\exo\monfichier.c"**

Un fichier source en C pour PIC18F4620 contiendra toujours la déclaration :

#include <p18f4620.h>

1.5. Fichier P18fxxx.h

Il s'agit d'un fichier d'inclusion particulièrement important lorsqu'on travaille en C sur un micro-contrôleur : le fichier de définition des registres internes du micro-contrôleur (**P18fxxx.h**) qui sont déclarés dans le fichier de déclaration des registres du processeur (**p18fxxx.asm**), fichier assembleur qui après compilation donne un fichier (**p18fxxx.o**) lui même contenu dans la bibliothèque pré-compilée (**p18fxxx.lib**) .

Par exemple dans le le fichier P18fxxx.h **port A** est définit de la façon suivante :

```
extern volatile near unsigned char PORTA;
extern volatile near union  {
    struct {
        unsigned RA0:1;
        unsigned RA1:1;
        unsigned RA2:1;
        unsigned RA3:1;
        unsigned RA4:1;
        unsigned RA5:1;
        unsigned RA6:1;
    } ;
    struct {
        unsigned AN0:1;
        unsigned AN1:1;
        unsigned AN2:1;
        unsigned AN3:1;
        unsigned :1;
        unsigned AN4:1;
        unsigned OSC2:1;
    } ;
    struct {
        unsigned :2;
        unsigned VREFM:1;
        unsigned VREFP:1;
        unsigned T0CKI:1;
        unsigned SS:1;
        unsigned CLK0:1;
    } ;
    struct {
        unsigned :5;
        unsigned LVDIN:1;
    } ;
} PORTAbits ;
```

→ Le port A est un octet (unsigned char) défini dans un fichier externe (extern) dont la valeur peut être écrasée entre 2 appels (volatile).

→ La deuxième déclaration précise que PORTAbits est une union de structures **anonymes** de bits adressables.
Du fait que chaque bit d'un registre de fonction peut avoir plusieurs affectations, il y peut y avoir plusieurs définitions de structures à l'intérieur de l'union pour un même registre.

Dans le cas présent les bits du port A sont définis comme :

1ère structure :
port d'E/S parallèle (7 bits ; RA0 à RA6)
2ème structure :
port d'entrées analogiques (5 entrées AN0 à AN4) + entrée OSC2.
3ème structure :
Des entrées de tension de référence du CAN, entrée horloge externe du timer0 (T0CKI), entrée de sélection du port série synchrone (SS), sortie du timer0 (CLK0).
4ème structure :
entrée low voltage detect (LVDIN)

Le contenu du registre ADCON1 déterminera l'affectation d'un bit (cf DS39564B page 182).

L'accés à un bit du portA se fait de la façon suivante :

Nom_union.nom_bit

Exemple :
PORTAbits.RA0 = 1 ; // mise à l'état haut de RA0

2. TP N°1: Prise en main du compilateur MCC18

(Travail individuel, Durée : 1h30)

Objectifs :

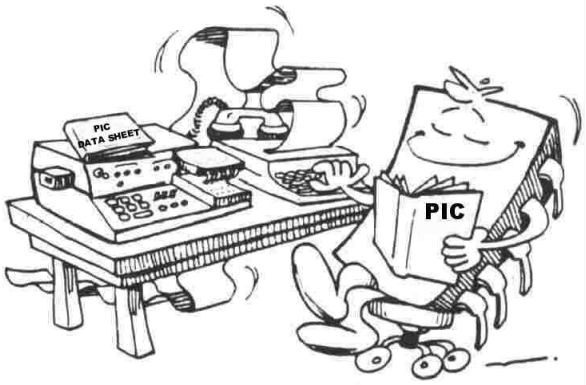
- Utiliser le compilateur MCC18 dans l'environnement MPLAB
- Etre capable de gérer les ports parallèles en C
- Etre capable de créer une fonction avec paramètres

Prérequis :

- Caractéristiques générales du compilateur MCC18 - Connaissance élémentaire du langage C
- Notions d'algorithmique
- Architecture du µcontrôleur PIC 18F4620

Données :

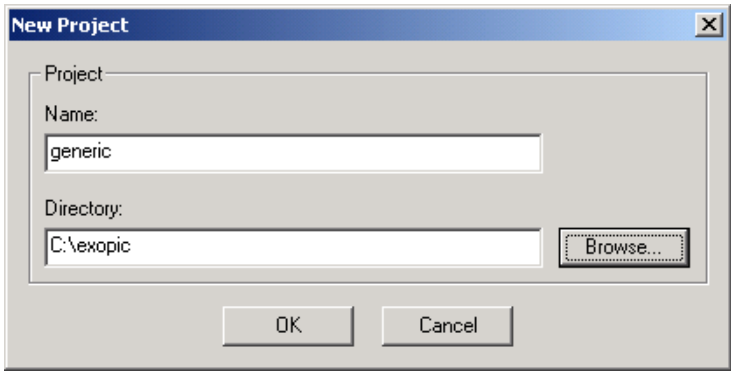
- Documentation minimale PIC 18F4620
- Guide d'utilisation de la carte PICDEM2 PLUS + TD associé



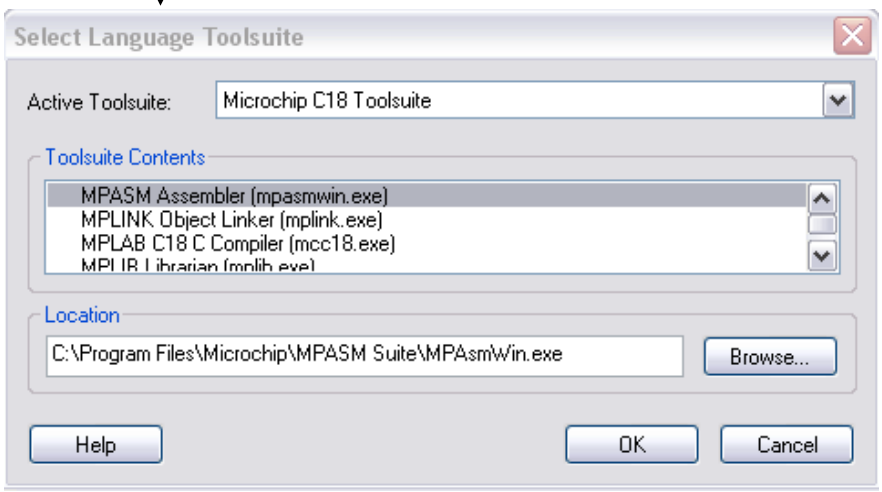
2.1. Prise en main du compilateur

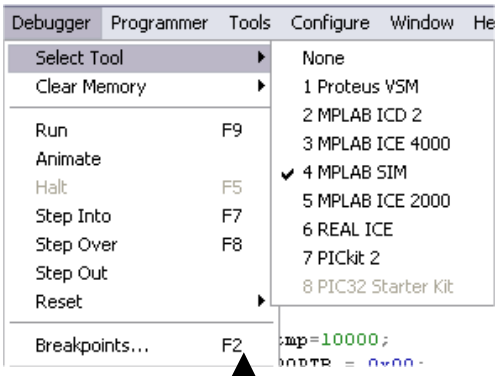
Création d'un projet générique en C, ce projet pourra servir pour tester **tous** les programmes exemples et effectuer les exercices.

Project :New
Name : generic (par exemple) →
Directory : c:\exopic\ (par exemple)



Project « Select language tools suite » , **choisir Microchip C18 Tools suite**
Vérifier les chemins des programmes et bibliothèques dans « Select language tools location »





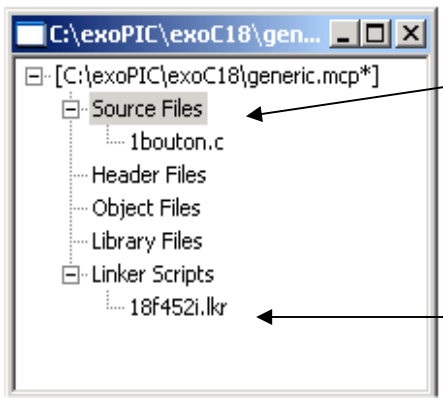
Sélectionner le type de debugger (en général ICD2 ou ICD3)

Chemins des programmes :
Assembleur : C c:\mcc18\mpasm\mpasmwin.exe
Linker : C:\mcc18\lkr
Compilateur : C:\mcc18\bin\mcc18.exe
Générateur de bibliothèques : C:\mcc18\mpLib.exe

Chemins des bibliothèques
Include search path : *.h c:\mcc18\h
Libray search path : *.lib c:\mcc18\lib
Linker-Script search path : *.lkr c:\mcc18\lkr

Les librairies du C18 sont compilées pour le mode d'adressage étendu.
Afin d'éviter certains « warning » lors de la compilation :
Dans project-buld options- project
Onglet MPLAB C18 catégorie « memory model »
Valider « large code model »

IMPORTANT : Project - Build options - project - directory , cliquer defaults



Sources Files contient les fichiers sources en C à compiler. Pour essayer les exemples qui suivent. Placer ici le fichier à compiler.

Un fichier d'édition de lien (.lkr) est un fichier de commande pour contrôler les opérations d'édition de lien par MPLINK pour un processeur cible donné ; ici un 18f452 (le i indique une configuration pour ICD2)
A partir de MPLAB 8.40, le lkr est définit par défaut

1.1. Directive #pragma config

La version 2.40 de MCC18 permet de configurer le microcontrôleur cible sans passer par les menu de MPLAB grâce à la directive #pragma config
(voir MPLAB C18 C Compiler Configuration Bit Setting Addendum (DS51518))

Exemple :

```
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
#pragma config DEBUG = ON
```

Configuration Bits

Address	Value	Category	Setting
300001	FA	Oscillator	HS
		Osc. Switch Enable	Disabled
300002	FF	Power Up Timer	Disabled
		Brown Out Detect	Enabled
		Brown Out Voltage	2.5V
300003	FE	Watchdog Timer	Disabled-Contro
		Watchdog Postscaler	1:128
300005	FF	CCP2 Mux	RC1
300006	7B	Stack Overflow Reset	Enabled
		Low Voltage Program	Disabled
300008	FF	Code Protect 00200-01FFF	Disabled
		Code Protect 02000-03FFF	Disabled
		Code Protect 04000-05FFF	Disabled

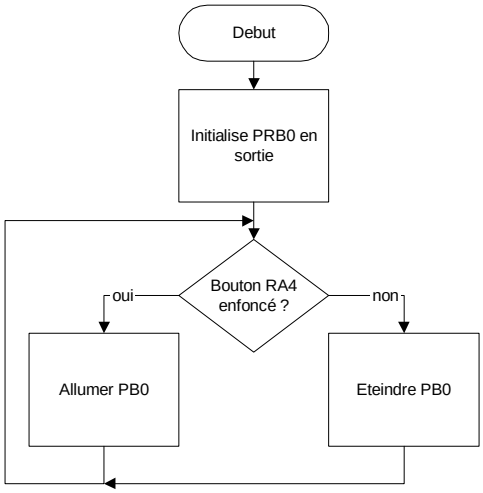
1.2. Gestion des ports parallèles

Créer un nouveau fichier avec le programme « bouton.c » ci dessous

```
/* Bouton et LED LED sur PICDEM2+*/
/* La LED sur PB0 s'éteint si S2 (PA4) est enfoncé*/

#include <p18fxxx.h>
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
#pragma config DEBUG = ON

void main(void)
{     TRISA=0xFF;       // PORTA en entrée
      TRISB = 0;        /* PB en sortie */
       while(1)         // une boucle infinie
      {
            if (PORTA & 0x10) PORTB=1;
            else PORTB=0;
      }
}
```

« header » du processeur cible (contient en particulier les définitions de TRISB, PORTA et PORTB

PORTA&0x10 est « vrai » si PORTA4 est à 0, bouton relâché



Remarques :

- seule la LED sur PB0 devant être modifiée, on aurait pu écrire : PORTB=PORTB|0b00000001; pour mettre PB0 à 1 et PORTB=PORTB&0b11111110; pour mettre PB0 à 0.
- Très souvent les masques sont utilisés en C pour les tests ou les positionnements de bit, cependant MCC18 permet de contrôler simplement n'importe quel bit à l'aide de ses déclarations de structure :

ex : **PORTAbits.RA0=1 ou a= PORTAbits.RA0**

Exemples

- **pour tester si PA4=1**

PORTA	x	x	x	x	x	x	x	x
&	0	0	0	1	0	0	0	0
=	0	0	0	x	0	0	0	0

Ex1 : Modifier ce programme afin d'incrémenter PRB à chaque pression sur RA4.
(pour tester RA4 : while(PORTAbits.RA4) ; …
On utilisera les définitions de bits. PORTxbits de p18fxxx.h

Le résultat est nul si PA4=0. *Le C associe dans les tests la notion de faux au 0 et la notion de vrai à un nombre différent de 0.*

- **Positionner PA4 à 0**

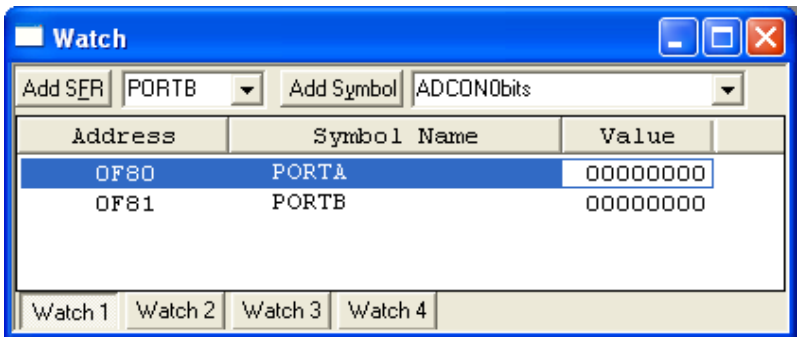
PORTA	x	x	x	x	x	x	x	x
&	1	1	1	0	1	1	1	1
=	x	x	x	0	x	x	x	x

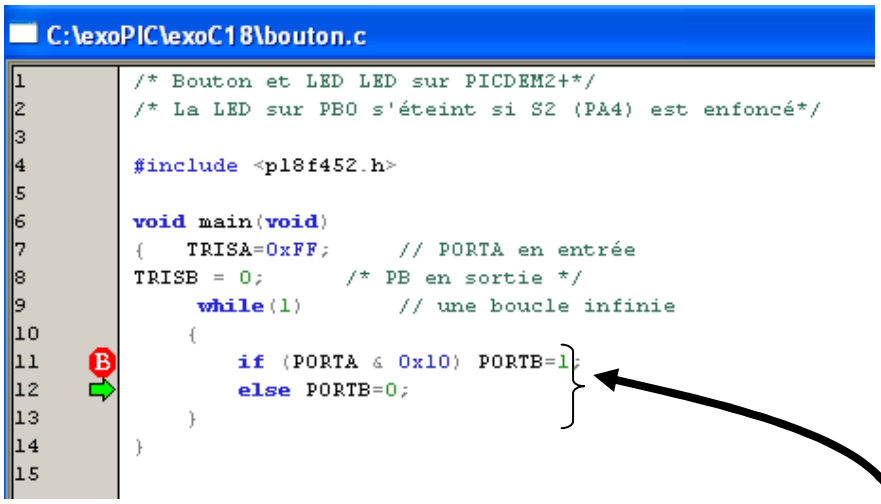
Positionner PA4 à 1

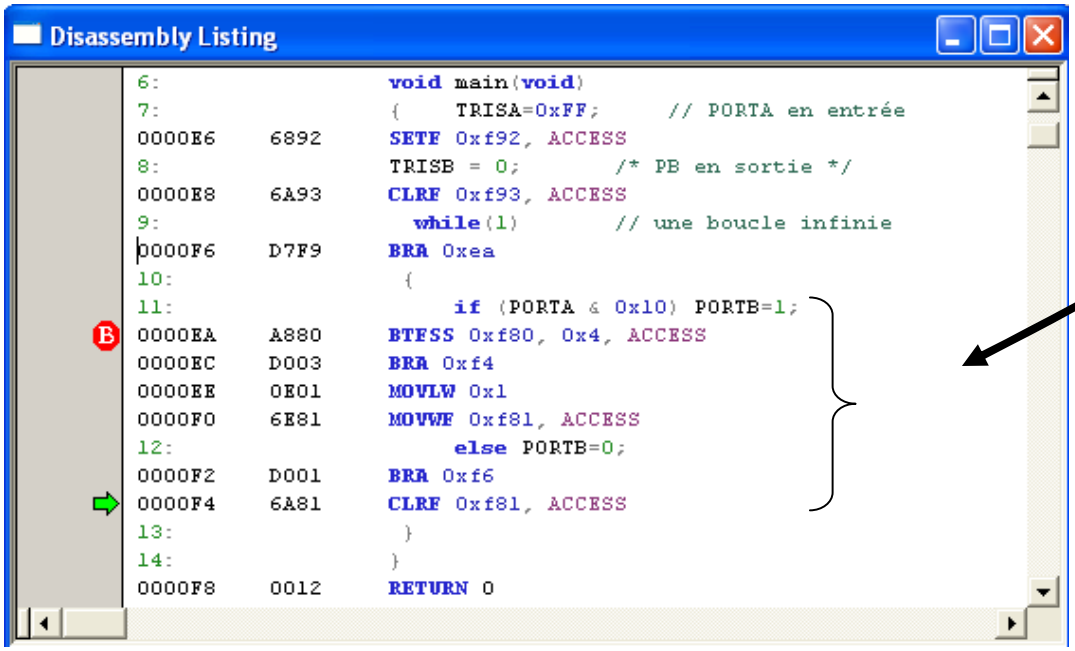
PORTA	x	x	x	X	x	x	x	x
OU	0	0	0	1	0	0	0	0
=	x	x	x	1	x	x	x	x

1.3. Mise au point d'un programme écrit en C dans MPLAB

Les fonctions de débug sont les mêmes qu'en assembleur : step into, step over, points d'arrêts etc…
Il est possible de tracer un programme en C et simultanément dans le fichier assembleur généré par MCC18.
Le compilateur C gérant les adresses, le programmeur ne connaît pas les adresses physiques des données.
Le fichier asm généré par le C et la fenêtre watch permet de visualiser les données,







.

Etre capable de créer une fonction avec paramètres d'entrée/sortie

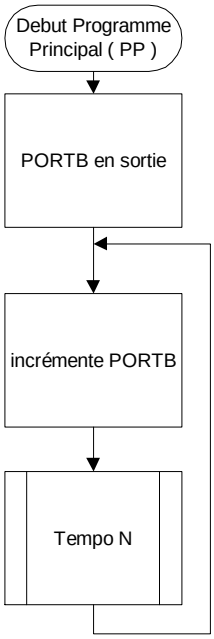
1.4. Création d'une fonction

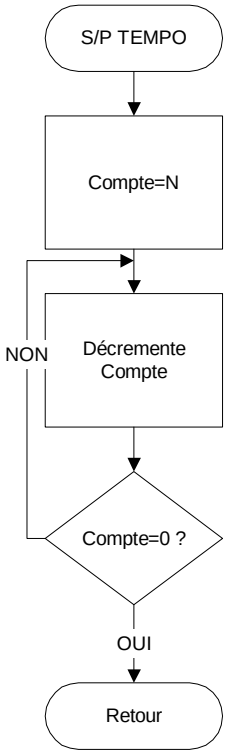
Recopier le programme led.c

La déclaration d'un prototype est nécessaire car la fonction tempo est définie après son appel

```
#include <p18fxxx.h>

#define duree 10000

void tempo(unsigned int count);

void main(void)
{
      PORTB = 0x00;
      TRISB = 0x00;
      while(1) {
            PORTB++;
            tempo(duree);
            }
}

void tempo(unsigned int compte)
{
      while(compte--);
}
```

Boucle infinie incrémentant PRB



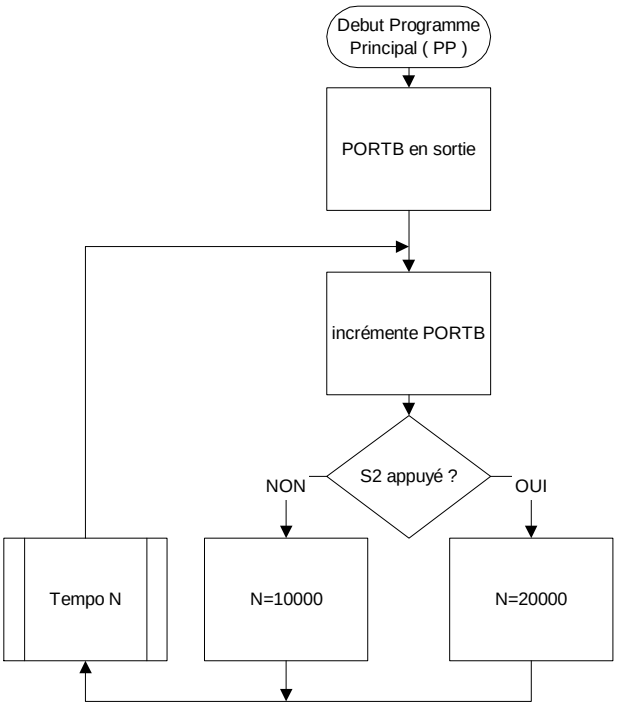


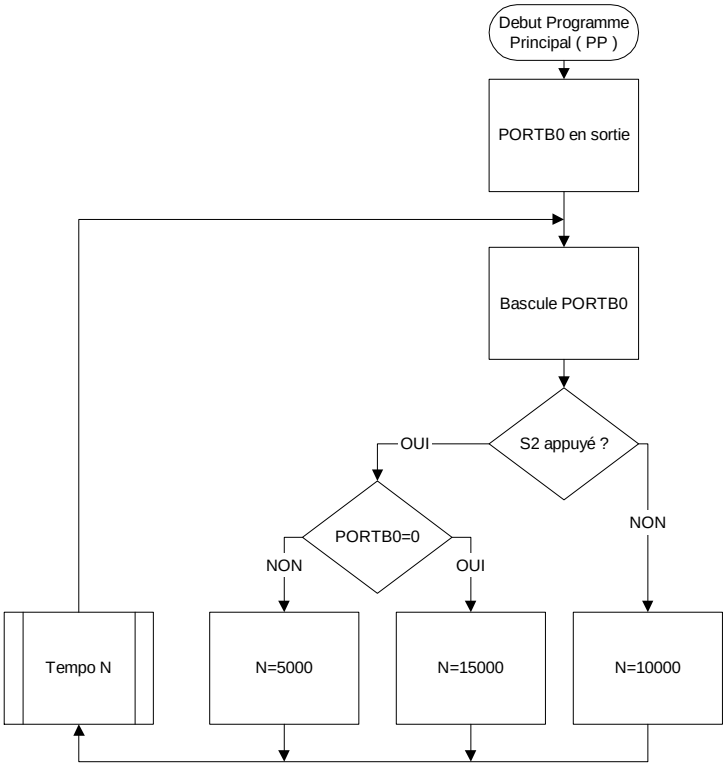
La fonction tempo reçoit un paramètre (int) qui est recopié dans la variable « compte », locale à la fonction. (duree n'est pas modifié)

***Remarque** : Si une fonction est écrite avant son appel le prototype devient inutile.*

***Ex2 :** modifier le programme led.c de manière à modifier la tempo (passer de 10000 à 20000) si S2 est appuyé.*

***Ex3** : Réaliser un programme faisant clignoter RB0 avec une période proche de 1s et un rapport cyclique ¼ si S2 est appuyé et ½ sinon.*





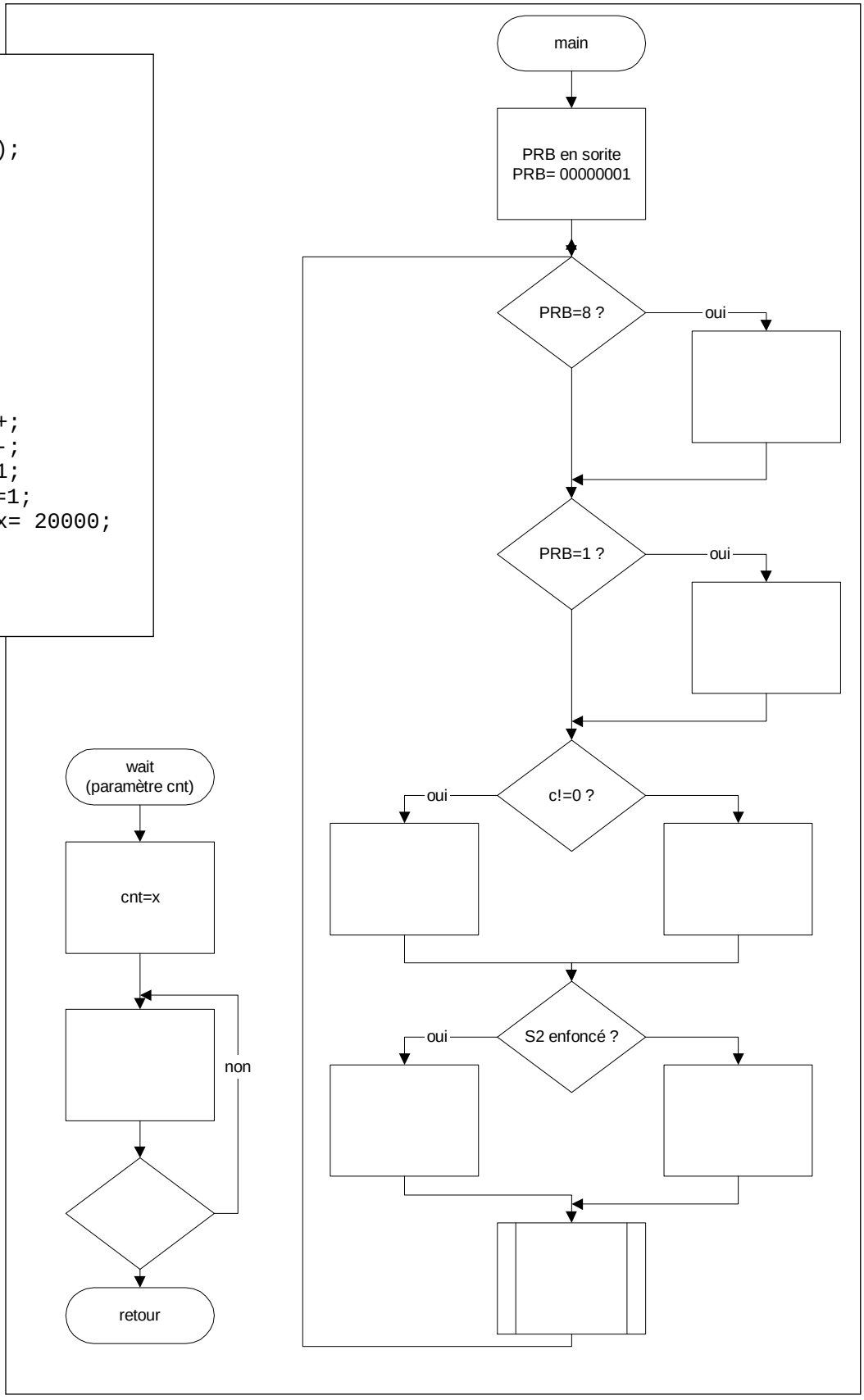
Être capable d'analyser un programme avec une syntaxe complexe en C

1.5. Analyse d'un programme écrit en C : décalages

Utilisation des opérateurs de décalage gauche et droite, ces derniers permettent également des multiplications et divisions par deux très rapides. (Filtre numérique par exemple)

```
#include <p18fxxx.h>
void wait(int cnt)
{
      for (;cnt>0; cnt--);
}

void main(void)
{
int x;
char c=0;
       TRISB = 0;
       PORTB=0b00000001;
      while(1)
      {
        if (PORTB==8) c++;
        if (PORTB==1) c--;
        if (!c) PORTB>>=1;
            else PORTB<<=1;
        if (PORTA&0x10) x= 20000;
            else x=5000;
        wait(x);
      }
}
```

A essayer puis compléter !



2. Bibliothèques MCC18

Une bibliothèque regroupe un ensemble de fonctions. Les fonctions utilisées peuvent être liées directement dans une application par l'éditeur de liens MPLINK à condition d'être déclarée dans un fichier header (.h)

2.1. Editeur de liens MPLINK

Lie entre eux les différents fichiers et résout les problèmes d'affectation en mémoire du programme et des données.

2.1.1. Rôle et contenu des fichiers d'édition de lien

Un fichier d'édition de lien est un fichier de commande pour contrôler les opérations d'édition de lien par MPLINK . Il permet :

- D'indiquer des chemins d'accès à des répertoires supplémentaires
- D'inclure des bibliothèques pré-compilées ou des fichiers objet
- De définir l'organisation mémoire du processeur cible
- D'allouer des sections sur le processeur cible
- D'initialiser la pile (taille et emplacement)

Exemple : fichier 18F4620i.lkr

```
// Sample linker command file for 18F4620i used with MPLAB ICD 2
// $Id: 18f4620i.lkr,v 1.2 2002/07/29 19:09:08 sealep Exp $
LIBPATH  .
FILES c018i.o
FILES clib.lib
FILES p18f4620.lib

CODEPAGE   NAME=vectors  START=0x0        END=0x29       PROTECTED
CODEPAGE   NAME=page     START=0x2A       END=0x7DBF
CODEPAGE   NAME=debug    START=0x7DC0     END=0x7FFF     PROTECTED
CODEPAGE   NAM
CODEPAGE   NAM
CODEPAGE   NAM
CODEPAGE   NAM

ACCESSBANK  NA
DATABANK    NA
DATABANK    NA
DATABANK    NAME=gpr2       START=0x200    END=0x2FF
DATABANK    NAME=gpr3       START=0x300    END=0x3FF
DATABANK    NAME=gpr4       START=0x400    END=0x4FF
DATABANK    NAME=gpr5       START=0x500    END=0x5F3
DATABANK    NAME=dbgspr     START=0x5F4    END=0x5FF    PROTECTED
ACCESSBANK  NAME=accesssfr  START=0xF80    END=0xFFF    PROTECTED
SECTION       NAME=CONFIG     ROM=config
STACK SIZE=0x100 RAM=gpr4
```

Chemins d'accés de bibliothèques ou fichiers objet.

Fichiers objets et bibliothèques précompilées à lier.

Définition de la mémoire programme

Définition de la mémoire Données

Définition de la pile logicielle

Le fichier lkr indique les chemins et librairies à balayer pour trouver le code objet des fonctions déclarées dans les fichiers header (*.h). Il y a trois librairies par défaut pour chaque lkr de chaque processeur.

- **C018i.o contient le CRT (C Run Time) d'initialisation des variables et d'appel « main »**
- **clib.lib contient les fonctions du standard CANSI**
- **p18fxxx.lib contient les équivalences et fonctions propres au microcontrôleur cible.**

2.1.2. Code de démarrage (CRT – C Run Time)

3 versions sont fournies avec le compilateur MCC18

- Co18.o Initialise la pile logicielle et se branche au début du programme utilisateur (fonction **main**) □ minimum de code.
- Co18i.o Idem + initialisation des données avant l'appel du programme utilisateur
- Co18iz.o Idem co18i.o + initialisation à zéro des variables statiques non initialisées par le programme (compatibilité C ANSI).

Le code source de ces programmes se trouve dans **mcc18\src\startup .**Pour reconstruire le code de démarrage et copier les fichiers objet dans le répertoire **\lib** lancer **build.bat .**
Le CRT boucle sur la fonction **main,** il est donc utile de toujours placer une boucle sans fin dans **main**

2.2. Bibliothèques spécifiques d'un processeur

Elles contiennent des fonctions dépendantes du processeur de la famille PIC 18 utilisé.
Ces fonctions sont de véritables composants logiciels fournis par MICROCHIP pour exploiter les ressources matérielles des micro-contrôleurs de la famille PIC18.

Elles sont contenues dans les bibliothèques **" p*processeur*.lib "** □ **P18Fxxx.lib**
.
Les fonctions de ces bibliothèques sont décrites dans le document **MPLAB® C18C COMPILER LIBRARIES (DS51297A)** □ Sous répertoire **\doc** du répertoire d'installation**:**

- **Chapitre 2 :** Hardware Peripheral Functions □ Fonctions de gestion des périphériques matériels:
  - ADC
  - Capture
  - I2C
  - Ports d'E/S //
  - PWM
  - SPI
  - Timer
  - USART

  Le code source correspondant se trouve dans les sous répertoires suivants du répertoire d'installation :
  Src\pmc\ADC [CCP, I2C, PORTB, PWM, SPI, Timers, USART]

- **Chapitre 3 :** Software Peripheral Library □ Gestion de périphériques externs et interfaces logiciels.
  - Afficheur lcd
  - CAN2510
  - I2C logiciel
  - SPI logiciel
  - UART logiciel

  Le code source correspondant se trouve dans les sous répertoires suivants du répertoire d'installation :
  Src\pmc\XLCD [CAN2510, swI2C, SW SPI, SW UART]

La reconstruction de la bibliothèque s'effectue à l'aide d'un fichier commande (DOS) du répertoire \src pour l'ensemble des processeurs de la famille PIC18 (c'est long) et par un fichier particulier pour un processeur unique

exemple : pour reconstruire la librairie du PIC18F4620 , P18F4620.LIB :
makeonep18f242. 18f4620

2.3. Fonctions C ANSI

Elles sont contenues dans la bibliothèque **" clib.lib "**.

Les fonctions de cette bibliothèque sont décrites dans le document **MPLAB® C18C COMPILER LIBRARIES (DS51297A)** □ Sous répertoire **\doc** du répertoire d'installation**:**

- **Chapitre 4 :** General Software Library
- **Chapitre 5 :** Math Libraries

Le code source correspondant se trouve dans les sous répertoires suivants du répertoire d'installation

- **Src\math** □ fonctions mathématiques
- **Src\stdclib** □ Classification des caractères, Fonctions de conversion de données standard C ANSI(atof, itoa etc.), Fonctions de mémorisation et de manipulation de chaînes de caractères (printf etc…)
- **Src\delays** □ Temporisations

Les bibliothèques existent en deux version "traditionnal" et "extended". Extended concerne les nouveaux PIC 18 avec un jeu d'instructions étendu.

La reconstruction de la bibliothèque **" clib.lib "** s'effectue à l'aide de l'utilitaire **makeall.bat** du répertoire \src.

ANSI 1989 standard C library

ctype.h

Function	Description
isalnum	Determine if a character is alphanumeric.
isalpha	Determine if a character is alphabetic.
iscntrl	Determine if a character is a control character.
isdigit	Determine if a character is a decimal digit.
isgraph	Determine if a character is a graphical character.
islower	Determine if a character is a lower case alphabetic character.
isprint	Determine if a character is a printable character.
ispunct	Determine if a character is a punctuation character.
isspace	Determine if a character is a white space character.
isupper	Determine if a character is an upper case alphabetic character.
isxdigit	Determine if a character is a hexadecimal digit.

stdlib.c

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

string.h

Function	Description
Memchr	Search for a value in a specified memory region
memcmp memcmppgm memcmppgm2ram memcmpram2pgm	Compare the contents of two arrays.
Memcpy memcpypgm2ram	Copy a buffer from data or program memory into data memory.
Memmove memmovepgm2ram	Copy a buffer from data or program memory into data memory.
Memset	Initialize an array with a single repeated value.
Strcat strcatpgm2ram	Append a copy of the source string to the end of the destination string.
Strchr	Locate the first occurrence of a value in a string.
Strcmp strcmppgm2ram	Compare two strings.
Strcpy strcpypgm2ram	Copy a string from data or program memory into data memory.
Strcspn	Calculate the number of consecutive characters at the beginning of a string that are not contained in a set of characters.
Strlen	Determine the length of a string.
Strlwr	Convert all upper case characters in a string to lower case.
Strncat strncatpgm2ram	Append a specified number of characters from the source string to the end of the destination string.
Strncmp	Compare two strings, up to a specified number of characters.
Strncpy strncpypgm2ram	Copy characters from the source string into the destination string, up to the specified number of characters.
Strpbrk	Search a string for the first occurrence of a character from a set of characters.
Strrchr	Locate the last occurrence of a specified character in a string.
Strspn	Calculate the number of consecutive characters at the beginning of a string that are contained in a set of characters.
Strstr	Locate the first occurrence of a string inside another string.
Strtok	Break a string into substrings, or tokens, by inserting null characters in place of specified delimiters.
Strupr	Convert all lower case characters

delays.h

Function	Description
Delay1TCY	Delay one instruction cycle.
Delay10TCYx	Delay in multiples of 10 instruction cycles.
Delay100TCYx	Delay in multiples of 100 instruction cycles.
Delay1KTCYx	Delay in multiples of 1,000 instruction cycles.
Delay10KTCYx	Delay in multiples of 10,000 instruction cycles.

reset.h

Function	Description
isBOR	Determine if the cause of a RESET was the Brown-Out Reset circuit.
isLVD	Determine if the cause of a RESET was a low voltage detect condition.
isMCLR	Determine if the cause of a RESET was the MCLR pin.
isPOR	Detect a Power-on RESET condition.
isWDTTO	Determine if the cause of a RESET was a watchdog timer time out.
isWDTWU	Determine if the cause of a wake-up was the watchdog timer.
isWU	Detects if the microcontroller was just waken up from SLEEP from the MCLR pin or an interrupt.
StatusReset	Set the POR and BOR bits.

2.3.1. math.h

La librairie math.h le fichier de définition des constantes mathématiques

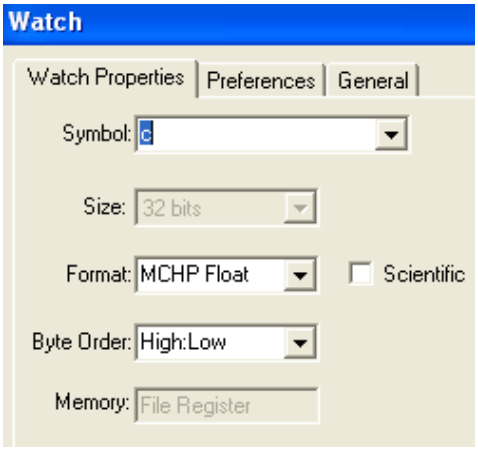
math.h

Fonction	Description
acos	Compute the inverse cosine (arccosine).
asin	Compute the inverse sine (arcsine).
atan	Compute the inverse tangent (arctangent).
atan2	Compute the inverse tangent (arctangent) of a ratio.
ceil	Compute the ceiling (least integer).
cos	Compute the cosine.
cosh	Compute the hyperbolic cosine.
exp	Compute the exponential e .
fabs	Compute the absolute value.
floor	Compute the floor (greatest integer).
fmod	Compute the remainder.
frexp	Split into fraction and exponent.
ieeetomchp	Convert an IEEE-754 format 32-bit floating point value into the Microchip 32-bit floating point format.
ldexp	Load exponent – compute x * 2 .
log	Compute the natural logarithm.
log10	Compute the common (base 10) logarithm.
mchptoieee	Convert a Microchip format 32-bit floating point value into the IEEE-754 32-bit floating point format.
modf	Compute the modulus.
pow	Compute the exponential x .
sin	Compute the sine.
sinh	Compute the hyperbolic sine.
sqrt	Compute the square root.
tan	Compute the tangent.
tanh	Compute the hyperbolic tangent.

mathdef.h

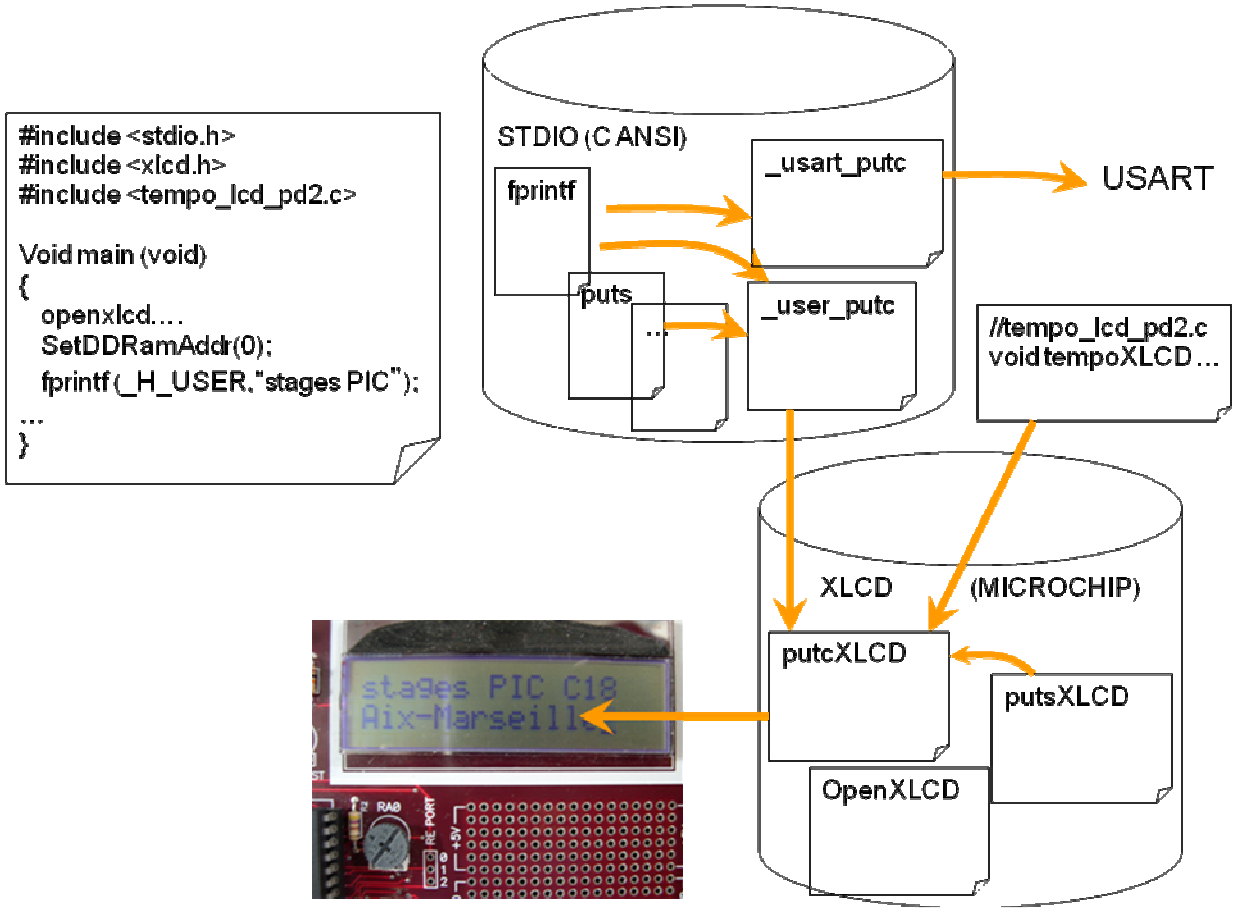
Definitions		
#define PI	3.141592653589793	// Constante Pi
#define PI_2	6.283185307179586	// Constante 2 Pi
#define PI_DIV2	1.570796326794896	// Constante Pi/2
#define INV_PI	0.318309886183790	// Constante 1/Pi
#define INV_PI_2	0.159154943091895	// Constante 1/2Pi
#define INV_PI_DIV2	0.636619772367581	// Constante 2/Pi
#define LN2	0.693147180559945	// Constante Log[2]
#define INV_LN2	1.442695040888963	// Constante 1/Log[2]
#define LN2_2	1.386294361119890	// Constante 2 Log[2]
#define INV_LN2_2	0.346573590279973	// Constante 1/2Log[2]
#define INV_LN10	0.434294481903252	// Constante 1/Log[10]
#define E	2.718281828	// Constante e
// degre - radian et radian - degre #define deg2rad(x) ((x)*1.7453293e-2) #define rad2deg(x) ((x)*57.296)		

Microchip n'utilise pas le format IEEE pour coder les réels,
pour visualiser ceux-ci dans une fenêtre WATCH,
il faut demander le format MCHP Float



2.4. Les sorties de texte

Le formatage du texte repose sur les fonctions C ANSI « printf » (voir page20) de la bibliothèque stdio.h
Les sorties bas-niveau (matériels) utilisent xlcd.h (bibliothèque crée avec l'utilitaire MAESTRO voir page21) pour l'afficheur LCD et la sortie par défaut (RCREG) pour l'USART.
La bibliothèque libusart.h (voir page 41) pour n'est utile que pour les entrées de caractères.



stdio.h est une librairie de gestion de sortie des caractères qui définit stdout (la sortie standard). Elle permet le formatage simple de chaînes de caractères vers différentes sorties (output stream)
Sur les PIC la sortie par défaut est l'USART. L'utilisateur peut définir sa propre sortie de caractères.
_H_USART est le nom du flux vers l'USART, il utilise la fonction _usart_putc
_H_USER est le nom du flux utilisateur. Il utilise la fonction _usart_putc

Pour rediriger stdout vers l'afficheur LCD d'un KIT PICDEM2+ il faut rediriger _user_putc vers l'afficheur LCD. (XLCDPut envoie un caractère vers l'afficheur LCD)

```
int _user_putc(char c)
{
      XLCDPut(c) ;
}
```

```
// fprintf.c  demo pour fprintf C18
#include <p18fxxx.h>
#include <stdio.h>        // pour fprintf
#include <xlcd.h>
#include <tempo_lcd_pd2.c> // tempo pour xlcd.h

// dirige user_putc vers l'afficheur LCD du PD2+
int _user_putc (char c)
{
      XLCDPut(c);
}

void main(void)
{
      SPBRG = 25; /* configure la vitesse (BAUD) 9600 N 8 1*/
      TXSTA = 0x24;
      RCSTA = 0x90;       /* active l'USART*/
      XLCDInit();//       /*initialise AffLCD*/
      XLCDL1home() ; //ligne 0 de l'afficheur
      fprintf (_H_USART, "fprintf USART\n");  // vers USART
      fprintf (_H_USER, "fprintf USER\n" );   // vers LCD
      while(1);
}
```

En déclarant #include <stdio..h> on dispose des fonctions :

Fonction	Description
fprintf	Envoie une chaîne formatée vers le flux défini fprintf(_H_USER, « vers l'afficheur LCD ») ; fprintf(_H_USART, « vers l'afficheur l'USART ») ;
fputs	Envoie une chaîne terminée par un passage à la ligne (newligne) vers le flux défini fputs(« Bonjour USART »,_H_USART) ;
printf	Envoie une chaîne formatée vers stdout. Exemples page suivante
putc	Envoie un caractère vers le flux défini putc('A', _H_USART) ; envoie A sur l'USART
puts	Envoie une chaîne terminée par un passage à la ligne (newligne) vers stdout. puts(« Bonjour ») ; envoie Bonjour vers stdout
sprintf	Envoie une chaîne formatée vers une zône mémoire RAM. Exemples page suivante
vfprintf	Comme fprintf mais en utilisant les arguments de stdarg (compatibilité CANSI)
vprintf	Comme printf mais en utilisant les arguments de stdarg (compatibilité CANSI)
vsprintf	Comme sprintf mais en utilisant les arguments de stdarg (compatibilité CANSI)
_usart_putc	Envoie un caractère vers l'USART
_user_putc	Envoie un caractère vers la sortie utilisateur (doit être écrit par l'utilisateur)

xlcd contient les fonctions :
XLCDInit() XLCDPutc (char c) XLCDL1home() et XLCDL2home() (voir page 21)

Afficheur LCD : Adresses curseur pour SetDDRamAddr

0x00 à 0x0F

0,0	1,0	2,0	3,0	4,0	5,0	6,0	7,0	8,0	9,0	10,0	11,0	12,0	13,0	14,0	15,0
0,1	1,1	2,1	3,1	4,1	5,1	6,1	7,1	8,1	9,1	10,1	11,1	12,1	13,1	14,1	15,1

0x40 à 0x4F

CODE ASCII (American Standard Code for Information Interchange)

code	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	NP	CR	SO	SI
0x10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0x20	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0x30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0x40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0x50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0x60	`	a	b	c	d	e	f	G	h	i	j	k	l	m	n	o
0x70	p	q	r	s	t	u	v	W	x	y	z	{	|	}	~	DEL

1.1.1. ftoa

ftoa (float to ascii) est une fonction standard du C ANSI mais elle n'est pas fournie avec MCC18. Pour afficher des nombres réels, utiliser ftoa.c qu'il suffit d'inclure dans le projet

unsigned char ***ftoa*** (float x, unsigned char *str,char prec,char format);

unsigned char chaine[10];
EX: ftoa(3.1415,chaine,2,'s')

ftoa convertit un réel en ACSII
prec indique la précision, 0 pour avoir le maximum
si **format** ='s' affichage scientifique 1.6666666E3
si **format** ='f' affichage classique 1666.6666

1.1.2. printf, fprintf, sprintf

printf permet la sortie formatée de chaînes de caractères (ici i=23 et c='A')

Format :

```
printf("un int %d un caractere %c",i,c);
```

```
un int 23 un caractere A
```

Transmission des arguments :

```
printf("%dh %dm %ds",heu,min,sec);
```

```
12h 41m 20s
```

%	Affichage
%c	Caractère ASCII
%d	Décimal signé pour entiers 8 ou 16 bits
%o	Octal pour entiers 8 ou 16 bits
%u	Décimal non signé pour entiers 8 ou 16 bits
%b	Binaire pour entiers 8 ou 16 bits (b)
%B	Binaire pour entiers 8 ou 16 bits (B)
%x	Hexadécimal pour entiers 8 ou 16 bits (minuscules)
%X	Hexadécimal pour entiers 8 ou 16 bits (majuscules)
%s	Chaîne ASCII en RAM
%S	Chaîne ASCII en ROM
%p	Pointeur Hexadécimal 16 bits (minuscules)
%P	Pointeur Hexadécimal 16 bits (majuscules)

Formats binaire et hexadécimal	
%X	AB
%#x	0xab
%#X	0XAB
%#06X	0X00AB
%B	1010
%#b	0b1010
%#B	0B1010
%#010B	0B00001010

```
int a = -27;
int b = 0xB5;
char c = 'A';
float r=31.416e-5;
char chram[ ]="en RAM";
rom const char chrom[ ]="en ROM" ;
char *pram=0x1cd;
rom char *prom=0x12Ab;
```

Script	Affichage
printf("Dec : %d %u",a,a);	Dec : -27 65509
printf("Hex: %#06X %x ",b,b);	Hex: 0X00B5 b5
printf("Bin: %16b",b);	Bin: 0000000010110101
printf("Bin: %#010B",b);	Bin: 0B10110101
printf("%c %c %d",'b',c,(int)c);	b A 65
printf("J habite %S",chrom);	J habite en ROM
printf("J habite %s",chram);	J habite en RAM
printf("pointeur RAM:%p %04P",pram,pram);	pointeur RAM:1cd 01CD
printf("pointeur ROM:%p %P",prom,prom);	pointeur ROM:12Ab 12AB

fprintf est identique à printf et permet de choisr la destination du flux

```
fprintf (_H_USER, "fprintf USER\n" );
```

sprintf est identique à printf, la sortie étant une zone RAM. La chaîne constituée peut-être envoyée ensuite sur n'importe quelle sortie.

```
unsigned char tampon[20] ;
sprintf(tampon,"Dec : %d %u",a,a);
```

Pour sortir une chaine contenant des réels (le qualificatif %f n'est pas reconnu par C18) on convertit le réel en chaine de caractères avec ftoa puis on inclut cette dernière dans fprintf avec le qualificatif %s

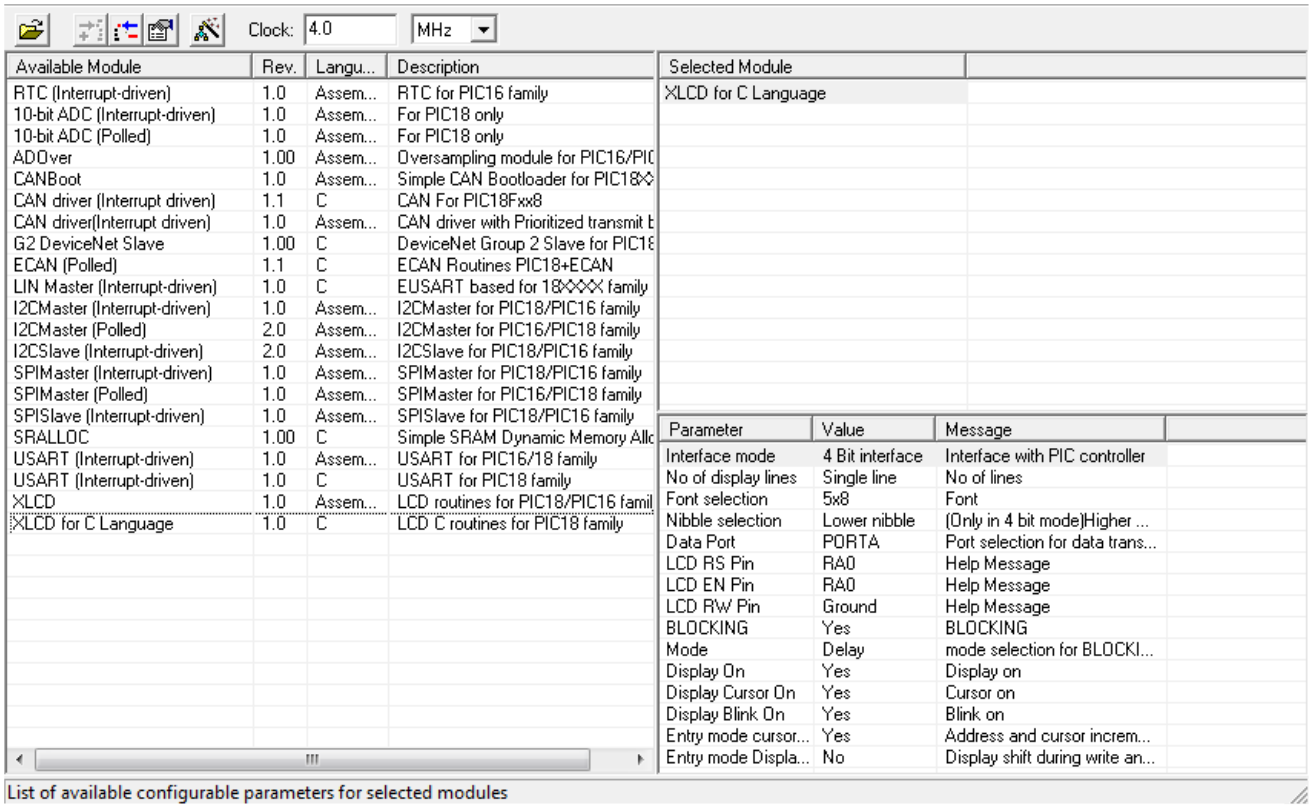
```
unsigned char tampon[20] ;  // on reserve 20 octets en mémoire RMA
ftoa(3.1415,tampon,2,'s') ; // ftoa converti 3.1415 en une chaine de caractères
fprintf (_H_USER, "La valeur de PI est %s",tampon);//fprintf affiche cette chaine
```

1.1.3. Fonctions de la bibliothèque XLCD:

Crées à partir de l'utilitaire MAESTRO © Microchip

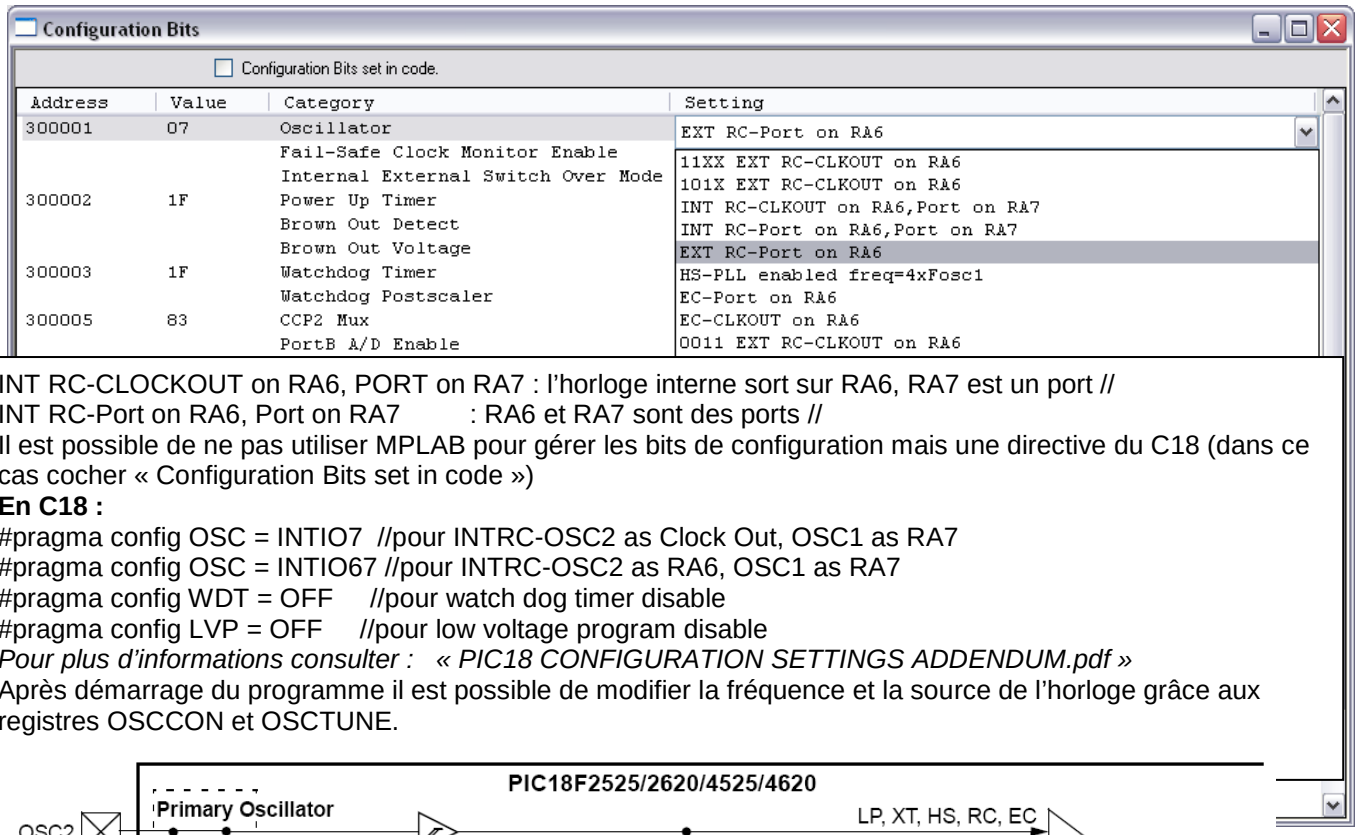
XLCDInit() Initialise l'afficheur LCD (type 1602) en function de la description faite dans MAESTRO
XLCDCommand(Command) Envoie une commande
XLCDPut(data Envoie une donnée à afficher (code ASCII)
XLCDIsBusy() Lit le drapeau 'busy' et retourne "vrai" si l'afficheur est occupé
XLCDGet() Lit la donnée dans l'afficheur à l'adresse courante
XLCDL1home() Positionne le curseur à gauche (ligne 1)
XLCDL2home() Positionne le curseur à gauche (ligne 2)
XLCDClear() Efface la RAM et positionne le curseur en haut à gauche
XLCDReturnHome() Positionne le curseur en haut à gauche
XLCDGetAddr() Lit la DDRAM
XLCDPutRomString(addr) Affiche une chaine depuis la ROM
XLCDPutRamString(addr) Affiche une chaine depuis la RAM

Ces fonctions sont décrites dans la documentation MAESTRO : XLCDc.readme.pdf



1.2. PIC18Fx620 – Configuration de l'horloge interne

Dans MPLAB :



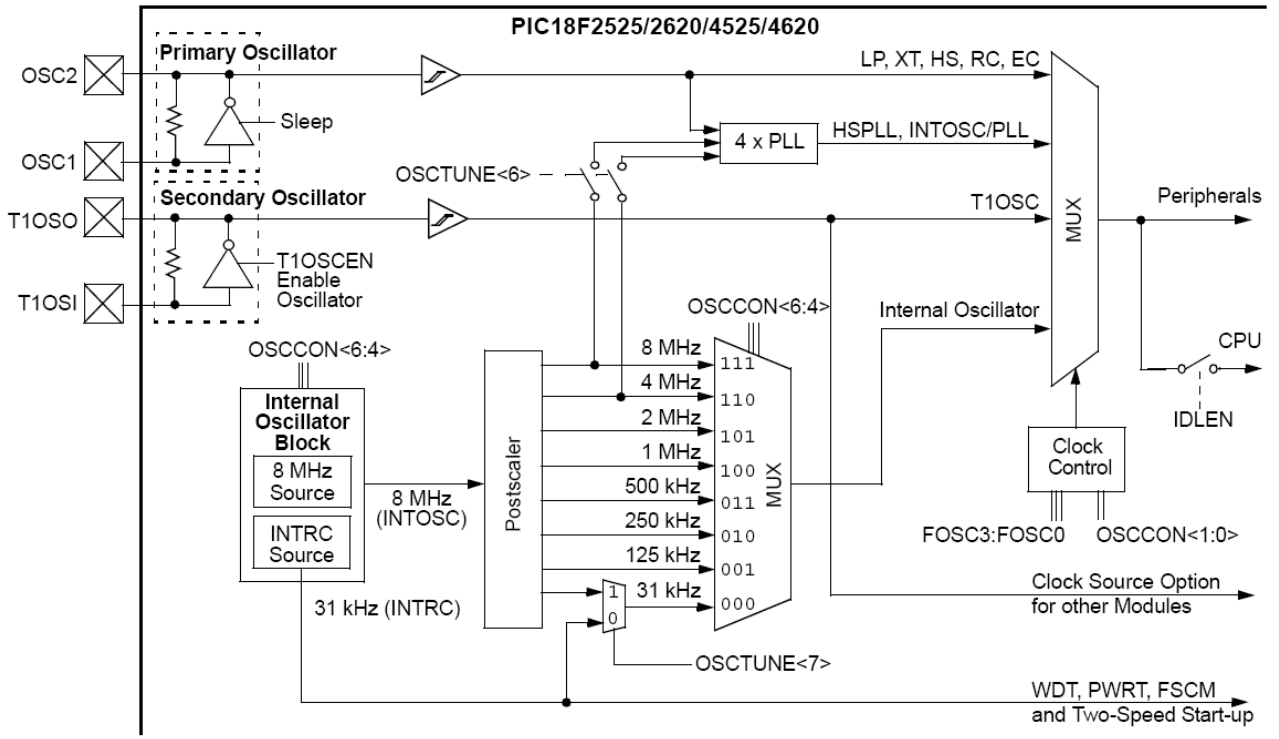
INT RC-CLOCKOUT on RA6, PORT on RA7 : l'horloge interne sort sur RA6, RA7 est un port //
INT RC-Port on RA6, Port on RA7 : RA6 et RA7 sont des ports //
Il est possible de ne pas utiliser MPLAB pour gérer les bits de configuration mais une directive du C18 (dans ce cas cocher « Configuration Bits set in code »)

En C18 :

```
#pragma config OSC = INTIO7  //pour INTRC-OSC2 as Clock Out, OSC1 as RA7
#pragma config OSC = INTIO67 //pour INTRC-OSC2 as RA6, OSC1 as RA7
#pragma config WDT = OFF     //pour watch dog timer disable
#pragma config LVP = OFF     //pour low voltage program disable
```

Pour plus d'informations consulter : « PIC18 CONFIGURATION SETTINGS ADDENDUM.pdf »

Après démarrage du programme il est possible de modifier la fréquence et la source de l'horloge grâce aux registres OSCCON et OSCTUNE.



Configuration de l'horloge interne, on suppose que les bits de configuration active la base de temps de 8MHz :

OSCCON: OSCILLATOR CONTROL REGISTER

R/W-0	R/W-1	R/W-0	R/W-0	R(1)	R-0	R/W-0	R/W-0
IDLEN	IRCF2	IRCF1	IRCF0	OSTS	IOFS	SCS1	SCS0

OSCTUNE: OSCILLATOR TUNING REGISTER

R/W-0	R/W-0(1)	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INTSRC	PLLEN(1)	—	TUN4	TUN3	TUN2	TUN1	TUN0

Les bits IRCF de OSCCON permettent de choisir la fréquence de base (31KHz à 8MHz). (1MHz par défaut)
Les bits TUN de OSCTUNE permettent d'ajuster la fréquence interne (en cas de variation de température par exemple)
Le bit PLLEN de OSCTUNE permet d'activer la PLL qui multipliera par quatre la fréquence de base (donc max 32MHz), désactivée par défaut)
Les bits SCS de OSCCON permettre de choisir la source de l'horloge des périphériques et du CPU du PIC. (Interne par défaut)
Les bits OSTS et IOFC de OSCCON permettent de connaître l'état de l'horloge (active, stable …)

Un exemple pour configurer l'horloge interne d'un PIC18FXX20 à 8MHz

```
OSCCONbits.IRCF2=1; // H interne 8MHz
OSCCONbits.IRCF1=1;
OSCCONbits.IRCF0=1;
OSCTUNEbits.PLLEN=1;    // PLL  (x4)
OSCCONbits.SCS1=0;      // sortie sur osc interne
OSCCONbits.SCS1=0;
```

1.3. TP N°2 Utilisation des bibliothèques

(Travail individuel , Durée : 2h30)

Objectifs :

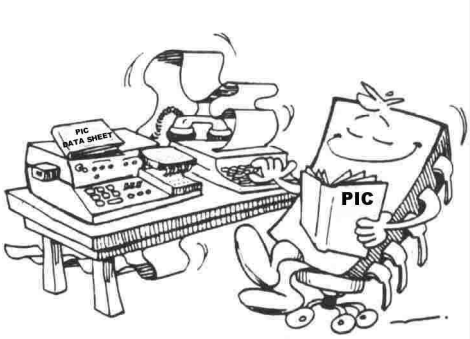
- Gérer l'afficheur LCD sur PICDEM2
- Mettre en oeuvre des fonctions de conversion btoa, itoa, ftoa et fonctions mathématiques
- Mettre en œuvre stdio.h, rediriger printf vers l'afficheur LCD du KIT PICDEM2+
- Utiliser des bibliothèques de composants logiciels MCC18
- Associer plusieurs fichiers sources dans un projet.
- Installer et mettre en œuvre la bibliothèque mathématique

Prérequis :

- Caractéristiques générales du compilateur MCC18 - Connaissance élémentaire du langage C
- Notions d'algorithmique
- Architecture du µcontrôleur PIC 18F4620

Données :

- Documentation minimale PIC 18F4620
- Guide d'utilisation de la carte PICDEM2 PLUS
- Guide d'utilisation des bibliothèques MCC18 : ***Ccompliler librairies DS51297a.pdf***



Travail demandé :

Exercices sur les librairies

- Installation de la bibliothèque xlcd.h
- Prise en main, essais d'affichage de différents types de donnée
- Test de la fonction mathématique « srqt » (racine carée)
- Intégration de la fonction « exp » (exponentielle) et création de la fonction « abs » (valeur absolue)

1.3.1. Exercice, sorties LCD ou USART

La sortie standard (std_out) est l'USART du PIC. On peut donc envoyer simplement des messages ASCII vers un PC (par exemple) avec printf ou fprintf(_H_USART, « ….)

Il faut alors initialiser l'USART du PIC, par exemple pour un format 9600,n,8,1 on introduira les lignes :

```
SPBRG = 25;    /* configure la vitesse (BAUD) 9600 N 8 1*/
TXSTA = 0x24;
RCSTA = 0x90;  /* active l'USART*/
```

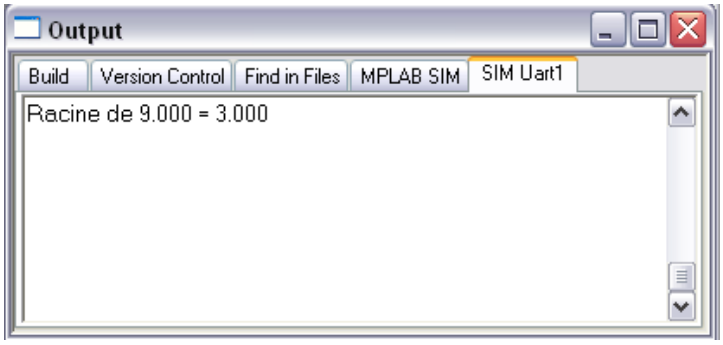
Brancher un câble série entre le KIT PD2+ et le port série d'un PC , et lancer un émulateur de terminal (le TERMINAL WINDOWS par exemple) et tester le programme ci-dessous.

```
// fprintf.c  demo pour fprintf C18
#include <p18fxxx.h>
#include <stdio.h>      // pour fprintf
#include <xlcd.h>       // pour OpenXLCD et putcXLCD
#include <tempo_lcd_pd2.c> // tempo pour xlcd.h
// dirige user_putc vers l'afficheur LCD du PD2+
int _user_putc (char c)
{
      XLCDputc (c);
}

void main(void)
{
      SPBRG = 25; /* configure la vitesse (BAUD) 9600 N 8 1*/
      TXSTA = 0x24;
      RCSTA = 0x90;     /* active l'USART*/
      XLCDInit();// LCD sur PD2
      XLCDL1home() ; //ligne 0 de l'afficheur
      fprintf (_H_USART, "fprintf USART\n");  // vers USART
      fprintf (_H_USER, "fprintf USER\n" );    // vers LCD
      while(1);
}
```

Le simulateur de MPLAB peut également afficher les sorties USART.

Activer le simulateur comme debugger (debugger-select tool-MPLAB sim)

Puis debugger-setting, la fenêtre « output » possède maintenant un onglet « Sim UART »



- Réaliser un programme comptant les appuis sur S2 (à partir du programme page9) et envoyant le nombre d'appuis vers l'USART (à partir du programme ci-dessus) sous cette forme :
« le bouton S2 a été enfoncé x fois. »
(\n\r permet de passer à la ligne et de revenir sur la première colonne)

1.3.2. Tester et analyser le programme de calcul de racines carrées

tstsqrt.c sur LCD et tstsqrtUSART.c sur USART))

```
// Test fonction Math
// calcul les racines carrées avec l'algo d'héron
// CD Lycée Fourcade 13120 Gardanne 5/2003
// evolution USART 11/2005

#include <p18fxxx.h>
#include <stdio.h>
#include "ftoa.c"

char chaine1[15],chaine2[15];

float sqrt(float f)
{
float xi,xi1;
char i;
      xi=1;
            for (i=0;i<8;i++)
      {
      xi1=(xi+f/xi)/2.0;
      xi=xi1;
    }
    return xi;
}

void main(void)  // la sortie s'effectue sur l'USART
{float f,r;
      SPBRG = 25; /* configure la vitesse (BAUD) 9600 N 8 1*/
      TXSTA = 0x24;
      RCSTA = 0x90;     /* active l'USART*/
      f=9.0;
      r=sqrt(f);        // on utilise Heron (pas math.h)
      ftoa(f,(unsigned char *)chaine1,3,'S');
      ftoa(r,(unsigned char *)chaine2,3,'S');
      fprintf(_H_USART,"Racine de %s = %s \n",chaine1,chaine2);
      while(1);
}
```

Le CAST évite les affichages lors de la compilation:
`Warning [2054] suspicious pointer conversion`

Exercice : A partir du programme « tstsqrt.c» ci-dessus, réaliser un programme de test pour la fonction « exp » ci dessous retournant l'exponentielle d'un nombre

La fonction abs retournant la valeur absolue de l'argument ***est à créer***

Pour les rapides : écrire la fonction mathématique réalisée par exp() ;

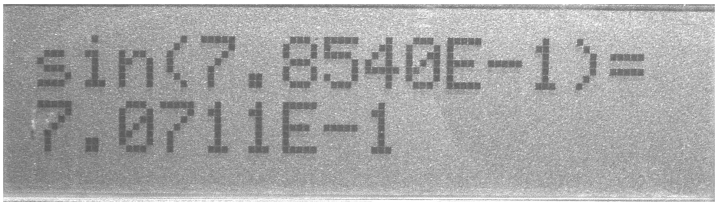
```
// fonction exponentielle sur nombres entiers
float exp(float f)
{
float s=1.0,u=1.0;
int n;
      for (n=1;abs(u)>0.001;n++)
      {
            u=u*f/n;
            s+=u;
      }
    return s;
}
```

1.3.3. Exercices sur math.h

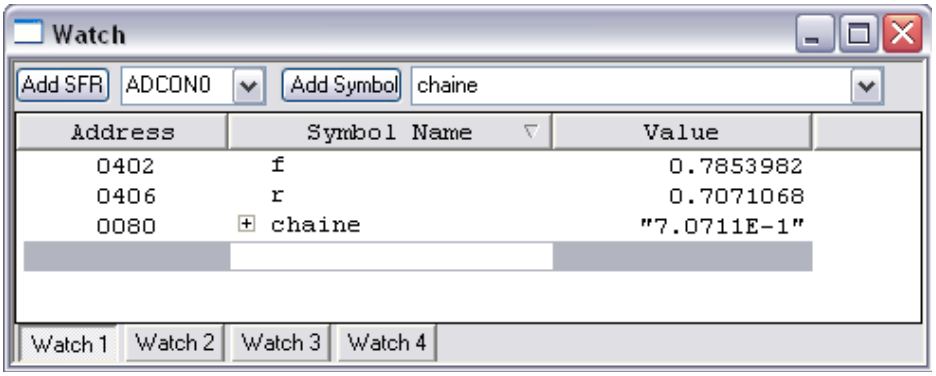
A partir du programme tstmath.c, tester diverses fonctions mathématiques de la librairie (sin, cos, log etc…)
Ici test de la fonction sinus. (Attention les angles doivent être donnés en radians)

```
#include <xlcd.h>
#include <stdio.h>
#include <math.h>
#include <mathdef.h>
#include "ftoa.c"

char chaine[10];

void main(void)
{float f,r;
      SPBRG = 25;
      TXSTA = 0x24;
      RCSTA = 0x90;     /* active l'USART*/
                  f=PI/4.0;
                  ftoa(f,chaine,4,'S');
                  fprintf(_H_USART,"sin(%s)=",chaine);
                  r=sin(f);
                  ftoa(r,chaine,4,'S');
                  fprintf(_H_USART,"%s  \n\r",chaine);

      while(1);
}
```

Consulter les .h
dans c:\mcc18\h



Visualiser les résultats sur l'afficheur LCD et dans une fenêtre "WATCH"



- Tester de même log, exp,pow, sqrt

2. Spécificités du compilateur MCC18

2.1. Type de données

- **Entiers**

Type	Size	Minimum	Maximum
char	8 bits	-128	127
signed char	8 bits	-128	127
unsigned char	8 bits	0	255
int	16 bits	-32768	32767
unsigned int	16 bits	0	65535
short	16 bits	-32768	32767
unsigned short	16 bits	0	65535
short long	24 bits	-8,388,608	8,388,607
unsigned short long	24 bits	0	16,777,215
long	32 bits	-2,147,483,648	2,147,483,647
unsigned long	32 bits	0	4,294,967,295

- **Réels**

Type	Size	Minimum Exponent	Maximum Exponent	Minimum Normalized	Maximum Normalized
float	32 bits	-126	128	2^{-126} = 1.17549435e - 38	$2^{128} * (2-2^{-15})$ = 6.80564693e + 38
double	32 bits	-126	128	2^{-126} = 1.17549435e - 38	$2^{128} * (2-2^{-15})$ = 6.80564693e + 38

2.2. Macros en C pour micro PIC

Instruction Macro[1]	Action
Nop()	Executes a no operation (NOP)
ClrWdt()	Clears the watchdog timer (CLRWDT)
Sleep()	Executes a SLEEP instruction
Reset()	Executes a device reset (RESET)
Rlcf(*var, dest, access*)[2,3]	Rotates *var* to the left through the carry bit.
Rlncf(*var, dest, access*)[2,3]	Rotates *var* to the left without going through the
carry bit	
Rrcf(*var, dest, access*)[2,3]	Rotates *var* to the right through the carry bit
Rrncf(*var, dest, access*)[2,3]	Rotates *var* to the right without going through the
carry bit	
Swapf(*var, dest, access*)[2,3]	Swaps the upper and lower nibble of *var*

Note 1: Using any of these macros in a function affects the ability of the MPLAB C18 compiler to perform optimizations on that function.
2: *var* must be an 8-bit quantity (i.e., char) and not located on the stack.
3: If *d*est is 0, the result is stored in WREG, and if *dest* is 1, the result is stored in *var*. If *access* is 0, the access bank will be selected, overriding the BSR value. If access is 1, then the bank will be selected as per the BSR value.

2.3. Assembleur en ligne

MCC18 contient un assembleur qui utilise une syntaxe identique à MPASM. Un module assembleur dans un programme en C commence par _**asm** et se termine par _**endasm**

```
char compte ;
_asm
     /* Code assembleur utilisateur   */
     MOVLW 10
     MOVWF compte, 0
     /* boucle jusqu'à 0 */
     debut:
     DECFSZ compte, 1, 0
     GOTO fin
     BRA debut
     fin:
_endasm
```

compte est une variable déclarée dans le fichier C

3. Gestion de la mémoire

3.1. Directives de gestion de la mémoire

Elles sont décrites dans le tableau ci-dessous.

Directive/ Rôle	Syntaxe / exemple
#pragma *sectiontype* Cette directive permet de changer la section dans laquelle MCC18 va allouer les informations associées. Une section est une partie de l'application localisée à une adresse spécifique. Ces directives permettent le contrôle total par l'utilisateur de l'allocation des données et du code en mémoire (optimisation, mise au point). **Sections par défaut :** (en l'absence de directive) **Type / nom par défaut** code /.code_nomfichier romdata / .romdata_nomfichier udata / .udata_nomfichier idata / .idata_nomfichier	***code : #pragma code [overlay] [nom[=adresse]]*** Contient des instructions exécutables ***romdata : #pragma romdata [overlay] [nom[=adresse]]*** Contient des constantes et des variables (normalement déclarées avec le qualificatif **rom**). ***udata : #pragma udata [overlay/access] [nom[=adresse]]*** Contient des variables utilisateur statiques non initialisées (uninitialized) ***idata : #pragma idata [overlay/access] [nom[=adresse]]*** Contient des variables utilisateur statiques non initialisées (initialized) **Access :** Localisation dans la zone access ram (256 octets : 00h à 7Fh De la Bank 0 et 80h à FFh de la Bank 15. **Overlay :** Permet de localiser plusieurs sections à la même adresse physique. On peut ainsi économiser de la mémoire en plaçant plusieurs variables au même emplacement. Cela fonctionne tant qu'on ne les utilise pas en même temps.

#pragma *varlocate*
 Indique au compilateur dans quel bloc mémoire (bank) placer une variable.
 Cela permet d'optimiser le code généré.
	#pragma *varlocate bank nom_variable*
	 ou
	 #pragma *varlocate nom_section nom_variable [, nom_variable …]*
	 Par exemple, dans un fichier c1 et c2 sont affectées en bank 1.
	 `#pragma udata bank1=0x100`
	 `signed char c1;`
	 `signed char c2;`
	 Dans un second fichier le compilateur est informé que c1 et c2 sont localisées en bank 1.
	 `#pragma varlocate 1 c1`
	 `extern signed char c1;`
	 `#pragma varlocate 1 c2`
	 `extern signed char c2;`
	 `void main (void)`
	 `{`
	 `c1 += 5;`
	 `/* No MOVLB instruction needs to be generated here. */`
	 `c2 += 5;`
	 `}`
	 Lorsque c1 et c2 sont utilisées dans le second fichier, le compilateur sait que les variables sont dans le même bloc et ne génère pas une instruction MOVLB supplémentaire.

5.2. Qualificatifs de mémorisation

Les microcontrôleurs PIC possèdent deux espaces mémoires (RAM et ROM) d'accès différents en raison de l'architecture Harvard, donc deux types d'instructions pour y accéder. Les constantes peuvent être en ROM ou en RAM (zone interdite en écriture dans le fichier *.lkr). Par défaut les constantes sont recopiées dans la RAM lors de l'initialisation. Pour éviter cela, il faut les déclarer « ROM ».
Remarque : Il est possible de placer des variables en ROM sur les microcontrôleurs équipés de ROM FLASH (cette procédure est complexe et nécessite l'ajout d'une procédure en assembleur propre au microcontrôleur, qui n'est pas encore implantée automatiquement par C18)

Localisation des données en fonction des qualificatifs :

	rom	**ram**
far	N'importe ou en mémoire programme (flash)	N'importe ou en mémoire RAM (default)
near	N'importe ou en mémoire programme (flash) sous 64KO	Dans access memory

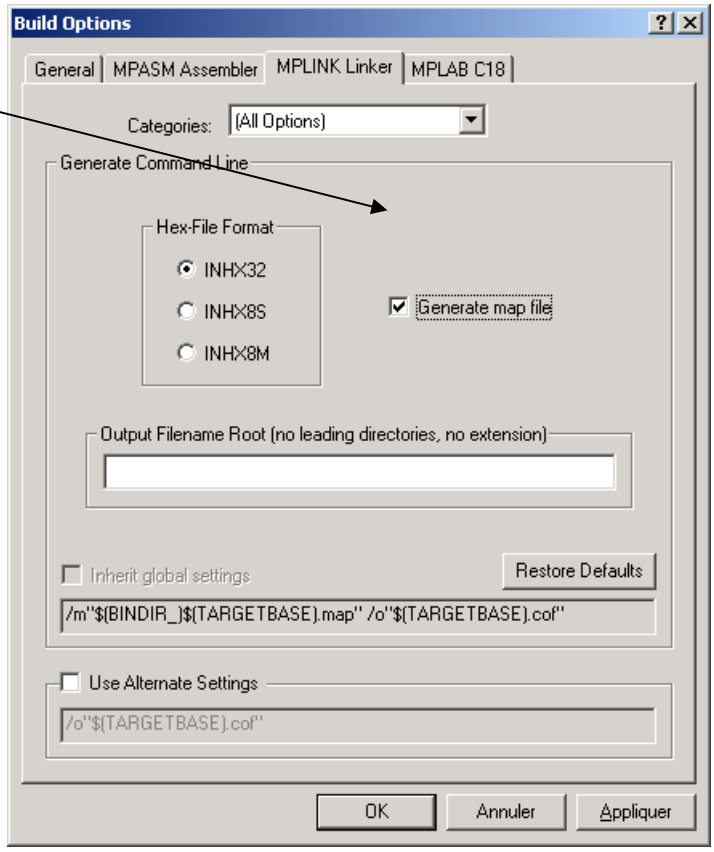
Taille des pointeurs :

Pointer Type	**Example**	**Size**
Pointeur sur RAM	char * dmp;	16 bits
Pointeur sur ROM <64KO	rom near * npmp;	16 bits
Pointeur sur ROM	rom far * fpmp;	24 bits

Fichier de routage mémoire (fichier map)

Pour générer ce fichier il faut activer l'option « Generate map file »

Dans le menu « Project Build Option Project » allez sous l'onglet MPLINK linker et activer la génération du fichier map.



Ce fichier rapporte l'occupation mémoire.

Fichier tstmem.map (partiel)

Symbols - Sorted by Name

Name	Address	Location	Storage	File
c	0x000128	program	extern	
d	0x000129	program	extern	
main	0x0000f6	program	extern	
r	0x00012b	program	extern	
s	0x00012d	program	extern	
a	0x00008f	data	extern	
b	0x000090	data	extern	
f	0x00008e	data	static	
p	0x00008a	data	extern	
q	0x00008c	data	extern	

5.3. TP N°3 : Gestion de la mémoire

(Travail individuel , Durée : 1h30)

Objectifs :

- Apprendre à utiliser les pointeurs
- Accéder à n'importe quelle partie de la mémoire

Ex6 :

1. Essayer les trois exemples et valider leur fonctionnement en affichant les contenus des mémoires programme et données du PIC.

2. Réaliser un programme « dump ». Affichage sur l'USART du contenu mémoire ROM puis RAM des 256 premiers octets par lignes de 16 octets sous la forme :

```
00    73 6B 64 68 66 66 73 64 6F 75 67 79 6F 73 75 66
10    6E 76 62 6B 75 73 64 68 66 62 73 6F 75 64 62 68
20    73 67 62 73 67 3A 3B 62 2C 73 6B 62 2C 6D 73 6B
30    66 67 6E 66 6E 73 73 6E 73 3E 77 00 00 00 00 9E
40    00 00 00 00 00 73 66 6E 6E 73 66 66 66 66 66 66
50    73 64 6E 73 6E 73 6E 73 6E 77 67 6E 73 6E 6E 73
60    6E 68 6E 73 6E 6E 77 73 73 66 68 73 7A 74 72 68
70    2C 64 2C 64 2C 64 2C 64 64 2C 64 64 2C 00 00 9E
80    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Exemple 1 : Qualificatifs de mémorisation (tstmem.c)

```
ram char a=0;           // a est en ram (le mot ram est facultatif
const ram char b=5;     // b est en ram mais ne peut être modifiée
rom char c=6;           // c est en rom et est modifialble si rom flash
const rom d=7;          // d est en rom et non modifialble

char *p;                // p est en ram et pointe en ram
rom char *q;            // q est en ram et pointe en rom
char *rom r;            // r est en rom et pointe en ram (rarement utile)
rom char *rom s;        // s est en rom et pointe en rom (rarement utile)

void fonction (void)
{
auto ram char e;  //e est dans la pile (ram et auto sont facultatifs)
static ram char f; // f est locale à la fonction mais à une adresse fixe
}

void main(void)
{
      a=4;
      c=0xAA;
}
```

Exemple 2 : utilisation de données en ROM (ledtbl.c).

Gestion d'un tableau

```
// exemple d'utilisation d'un tableau de constantes en ROM ! CD 01-2003
// seuls les 4 bits de poids faibles du PORTB commandent des LEDs
#include <p18fxxx.h>
#define tbltaille 16    // taille de la table


const rom unsigned char sortie[]={0b00000000,0b00000001,0b00000010,
0b00000100,0b00001000,0b00000100,0b00000010,0b00000001,0b00000000,0b00000001,0b00
000011,0b00000111,0b00001111,0b00001110,0b00001100,0b00001000 };

void wait(int cnt)
{
      for (;cnt>0; cnt--);
}

void main(void)
{     char c=0;
      TRISB = 0;                          // PB = sortie
      while(1)                            // boucle infinie
      { for(c=0;c<tbltaille;c++)
                  {     PORTB=sortie[c];  // c indexe la table
                        wait(5000);
                  }
      }
}
```

Sortie est un tableau de constantes rangées en ROM

cnt est une variable auto de la fonction, elle reçoit le paramètre d'entrée

c est local à "main" donc rangée dans la pile

Exemple 3 : Utilisation des directives de gestion de la mémoire (gestmem.c)

Copie ROM RAM

```
// Copie une chaîne de caractères localisée en rom à 0x1000 dans une chaîne
// en ram à 0x300 sur µcontrôleur PIC 18F4620.
// RT le 17/12/02

#include <p18fxxx.h>

#pragma romdata mamemoire=0x1000
rom unsigned char chaine1[]="bonjour",*ptr1;

#pragma udata mesdonnees=0x300
unsigned char chaine2[20],*ptr2;

void copyRom2Ram(rom unsigned char *Romptr,unsigned char *Ramptr)
{

while(*Romptr)
   {
      *Ramptr++=*Romptr++;
   }
}

void main(void)
{
      ptr1=chaine1;
      ptr2=chaine2;
      copyRom2Ram(ptr1,ptr2);
}
```

pragma romdata : les données en ROM seront rangées à partir de l'adresse 0x1000

pragma udata : les données en RAM seront rangées à partir de l'adresse 0x300

Le contenu du pointeur Romptr est recopié dans le contenu du pointeur Ramptr jusqu'à ce que ce dernier égale 0x00

ptr1 pointe sur la chaine1 en ROM et ptr2 sur la chaine2 en RAM

6. Gestion des interruptions

- Le C ne sait pas traiter les sous programmes d'interruption. Ceux-ci se terminent par l'instruction assembleur RETFIE et non par RETURN (dépilements différents)
- Le C ne laisse pas le contrôle des adresses au programmeur. Les interruptions renvoient à une adresse fixée par MICROCHIP (0x08 ou 0x18)

6.1. Directives de gestion des interruptions

#pragma *interruptlow*

Déclaration d'une fonction en tant que programme de traitement d'interruption **non prioritaire. (vect = 0x18) l'instruction de retour sera RETFIE**

#pragma *interrupt*

Déclaration d'une fonction en tant que programme de traitement d'interruption **prioritaire. (vect = 0x08) l'instruction de retour sera RETFIE FAST. *Seuls les registres W, BRS et STATUS sont restaurés***

```
#include <p18fxxx.h>
//Initialisation du vecteur d'interruption
#pragma code adresse_it=0x08 //Place le code à l'adresse 0x08
void int_toto(void)
{
  _asm it_prioritaire _endasm // Branchement au S/P d'it
}
#pragma code       // retour à la section par défaut

//Sous programme de traitement de l'interruption
#pragma interrupt it_prioritaire
void it_prioritaire (void)
{
/* placer le code de traitement de l'IT ici */
     if (INTbitx) {  … traitement de l'ITx …
                     INTbitxF=0;  // efface drapreau d'ITx
                  }
     if (INTbity  {  … traitement de l'ITy …
                     INTbityF=0;  // efface drapreau d'ITy
                  }
}
```

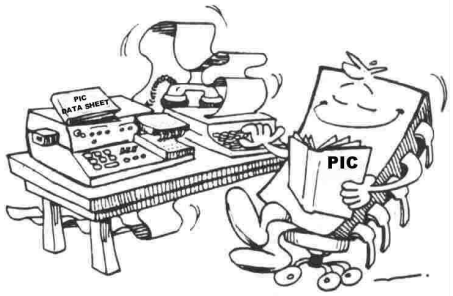
Rappel : Si le bit IPEN(RCON)=1 (0 par défaut) les priorités d'interruptions sont activées.
Si IPEN=0; GIE=1 active toutes les interruptions autorisés
Si IPEN=1; GIEH=1 active les interruptions prioritaires et GIEL=1 les interruptions non prioritaires. Les registres PIR permettent de définir les priorités.

6.2. TP N°4 : Gestion des Timers en interruption

(Travail individuel , Durée : 2h00)

Objectifs :

- Mettre en œuvre les interruptions des Timers du PIC18F4620 en C18



Prérequis :

- Caractéristiques générales du compilateur MCC18
- Connaissance élémentaire du langage C

Données :

- Documentation minimale PIC 18F4620
- Guide d'utilisation de la carte PICDEM2 PLUS

Travail demandé :

1 Production de temporisation

Le programme flashit.c ne permet pas de produire une durée de 500mS, pour cela il est nécessaire d'utiliser la fonction COMPARE associée au TIMER1 par exemple **Programme itcomp.c** en annexe .
Ex 7 : Modifier le programme **itcomp.c** pour à obtenir un rapport cyclique ¼ si S2 est enfoncé et ½ sinon.

2 Mesure de durée

Ex 8 : A partir du programme **itcapt.c** fourni en annexe, réaliser un fréquencemètre sur afficheur LCD. Définir en fonction des paramètres de TIMER1 les fréquences max et min mesurables.

3 Exercices pour les plus rapides

Ex 9 : A partir de flashit.c et lcdtst.c, construire une horloge affichant heures, minutes, secondes. La mise à l'heure se fera dans le débogueur MPLAB
Dans un deuxième temps la mise à l'heure se fera par S2 pour les minutes et S3 pour les heures.

6.3. Exemple de programme fonctionnant en IT

6.3.1. Avec le PORTB : Programme demo_it_rb0.c

```
// demo_it_rb0.c demo de mise en oeuvre des interruptions
// ce programme incrémente une mémoire à chaque appuis sur S3
// attention : sur PICDEM2+ retirer le strap J6

#include <p18f452.h>

unsigned int cpt=0;     // compteur d'interruption

// sous programme d'interruption
#pragma interrupt it_sur_rb0
void it_sur_rb0(void)
{
      if (INTCONbits.INT0IF)  // vérifie que l'IT INT0, origine PB0=0 (bouton S3)
                  {
                        cpt++;  //utiliser un WATCH pour visualiser cpt
                        INTCONbits.INT0IF=0;    //efface le drapeau d'IT
                  }
}

#pragma code vecteur_d_IT=0x08      // vecteur d'IT
void une_fonction(void)
{
_asm goto it_sur_rb0 _endasm
}
#pragma code


void main (void)
{
      // configure IT sur PB0
      TRISBbits.TRISB0=1;           // PRB0 en entrée
      INTCONbits.INT0IE=1;    // INT0 activée , front descendant
      INTCONbits.GIE=1;       // Toutes les IT démasquées autorisées

      while(1);               // attente d'un évènement, le programme ne fait plus rien
}
```

Le port B est configuré en entrée, l'interruption sur front descendant de RB0 est activée.
Lors d'un appui sue S3 (front descendant sur RB0) , il y a interruption, le processeur exécute l'instruction à l'adresse 0x08 (intrinsèque au PIC18) . L'espace d'instruction à cet endroit étant réduit, on place un saut absolu sur le sous programme d'interruption (`void it_sur_rb0(void)`).

Le sous programme d'interruption vérifie l'origine de l'interruption puis (pour cet exemple) incrémente un compteur (cpt). Le drapeau de l'interruption est effacé avant le retour vers la boucle while(1) du programme principal.

6.3.2. Avec le TIMER 0 : Programme flashit.c

Ce programme fait clignoter la LED sur PB0 par interruption sur le TIMER0 T=1.048s (TIMER0 produit des temps de 2expN). Il s'agit d'une mise en oeuvre simple du TIMER 0, chaque débordement provoque une IT.
Le timer est en mode 16 bits avec horloge Fosc/4 soit 1MHz, prédiviseur par 8
La période des débordements est donc 1uS * 8 * 65536 = 524.288 mS

```
#include <p18fxxx.h>

void traiteIT(void);
```

le vecteur d'IT prioritaire se trouveà l'adresse 8. Cette pragma force le compilateur à placer le code à l'adresse indiquée

```
#pragma code it=0x08
void saut_sur_spIT(void)
{
_asm
      goto traiteIT
_endasm
}
#pragma code
```

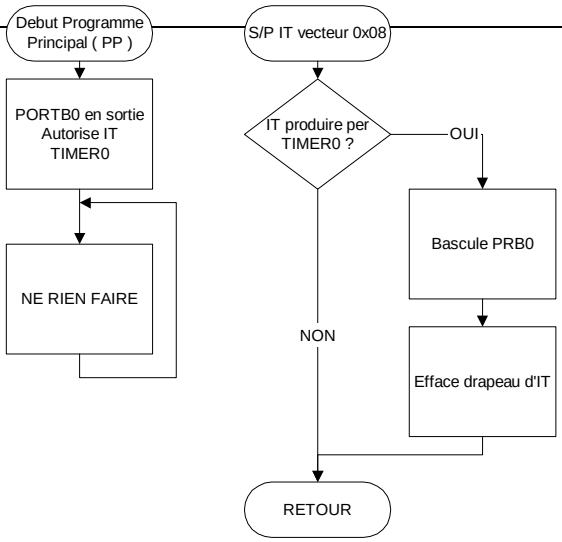
Facultatif ici, il n'y a pas d'IT en 0x18

Saut sur le S/P de traitement de l'interruption

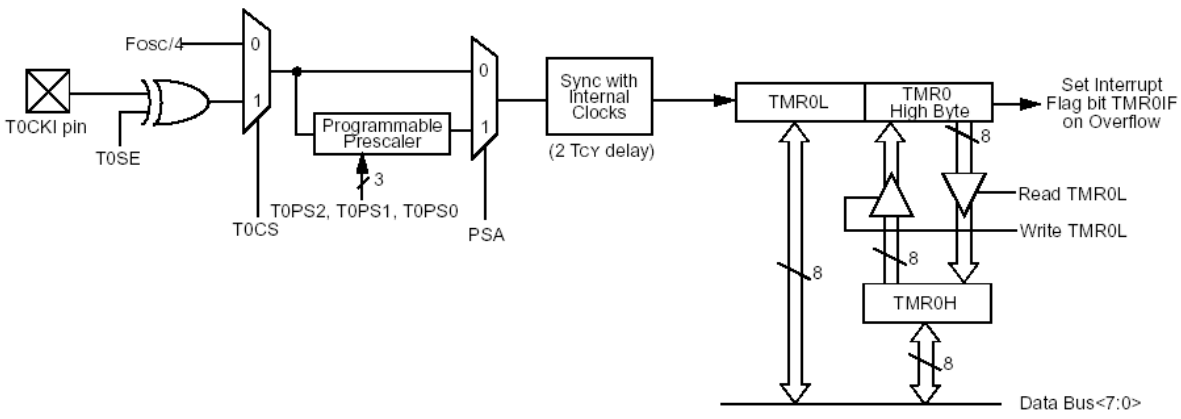
le compilateur peut à nouveau gérer les adresses

```
#pragma interrupt traiteIT
void traiteIT(void)
{
      if(INTCONbits.TMR0IF)   //vérifie un débordement sur TMR0
                  {INTCONbits.TMR0IF = 0; //efface le drapeau d'IT
                  PORTBbits.RB0 = !PORTBbits.RB0;     //bascule LED sur RB0
                  }
}
void main()
{
      PORTBbits.RB0 = 0;
      TRISBbits.TRISB0 = 0;
      T0CON = 0x82;
      INTCONbits.TMR0IE = 1;
      INTCONbits.GIEH = 1;
      while(1);
}
```

traiteIT est un SP d'IT, il finit donc par retfie et non par return. Il n'y a aucun paramètre pour un SP d'IT car son appel est asynchrone



Ex : tester le programme, mesurer la période du signal sur PRB0 et conclure par rapport à la configuration du TIMER0



6.4. Timers

6.4.1. Production de temps

Programme itcomp.c

Le programme flashit.c ne permet pas de produire une durée de 500mS, pour cela il est nécessaire d'utiliser la fonction COMPARE associée au TIMER1 par exemple .

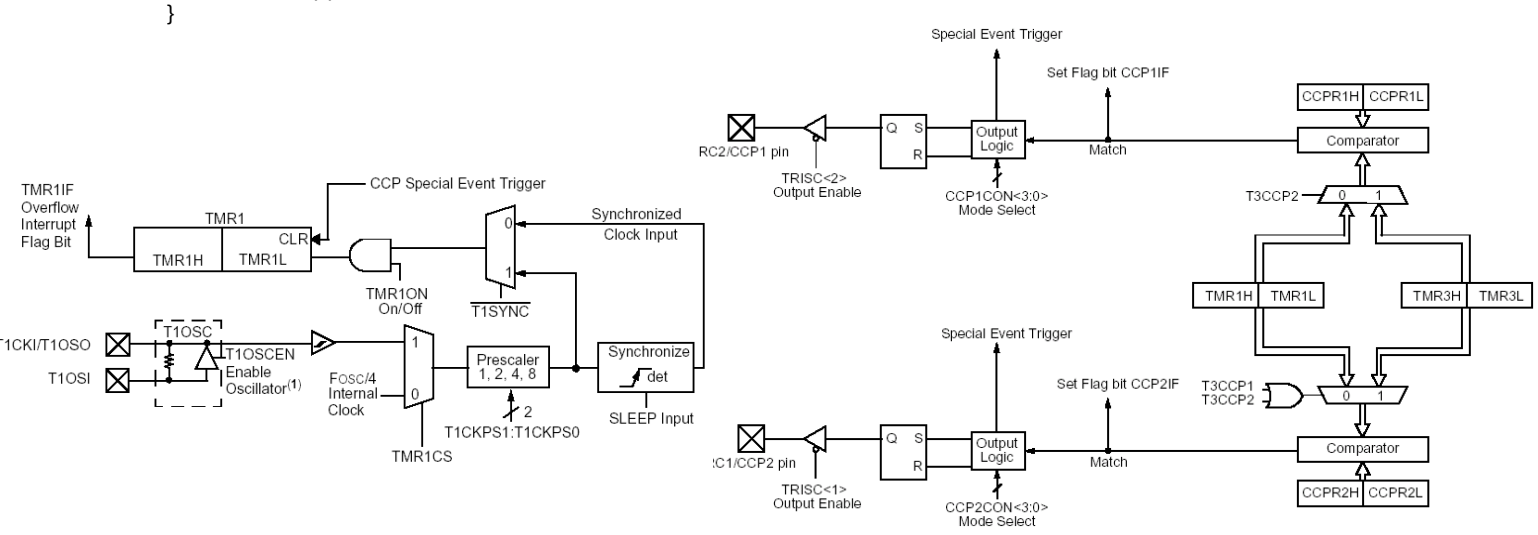
Il y a une IT toutes les 125mS, il faut attendre 4 IT avant de basculer PB0. le compteur d'IT tictac est local, il doit également être statique pour ne pas être perdu à la

```
#include <p18fxxx.h>
// sous programme d'interruption
#pragma interrupt traite_it
void traite_it(void)
{
static char tictac;     // IT toutes les 125mS, 4 IT avant de basculer PB0
        if( PIR1bits.CCP1IF)            // l'IT provient d'une comparaison
                        {
                        if (++tictac>=4) {
                                        PORTBbits.RB0=!PORTBbits.RB0;       //bascule PB0
                                        tictac=0;
                                                }
                        PIR1bits.CCP1IF=0;
                        }
}

#pragma code vec_it=0x08
void vect8 (void)
{
_asm goto traite_it  _endasm
}
#pragma code

void main(void)
{
        //configure PORTB
        PORTBbits.RB0=0;            // RB0 e
        TRISBbits.TRISB0=0;
        // configure le TIMER1
        T1CONbits.RD16=0;           // TMR1 mode simple (pas de RW)
        T1CONbits.TMR1CS=0;         // compte les impulsions sur internal clock
        T1CONbits.T1CKPS1=1;        // prédiviseur =1/8 periode sortie = 8uS
        T1CONbits.T1CKPS0=1;
        T1CONbits.T1SYNC=1;
        T1CONbits.TMR1ON=1;         // TMR1 Activé

// configure le mode comparaison sur le TIMER1 avec IT sur CCP1 toutes les 62500 périodes de 8us soit 125ms
        T3CONbits.T3CCP2=0;         // mode comparaison entre TMR1 et CCPR1
        CCP1CON=0x0B;               // Trigger special event sur comparaison (RAZ TIMER1 lors de l'égalité)

        CCPR1H=0x3d;                // égalité après 15625 périodes de 8ms (125mS)
        CCPR1L=0x09;

        PIE1bits.CCP1IE=1;          // active IT sur mode comparaison CCP1

        RCONbits.IPEN=1;            // Interruption prioritaires activées
        INTCONbits.GIE=1;           // Toutes les IT démasquées autorisées

        while(1);                   // une boucle infinie, tout fonctionne en IT
}
```

configure le mode comparaison sur le TIMER1 avec IT sur CCP1 toutes les 62500 périodes de 8us soit 125ms

Ex7 : Modifier itcomp.c de manière à obtenir un rapport cyclique ¼ si S2 est enfoncé et ¾ sinon en fonction de TICTAC ou des valeurs dans CCPR1

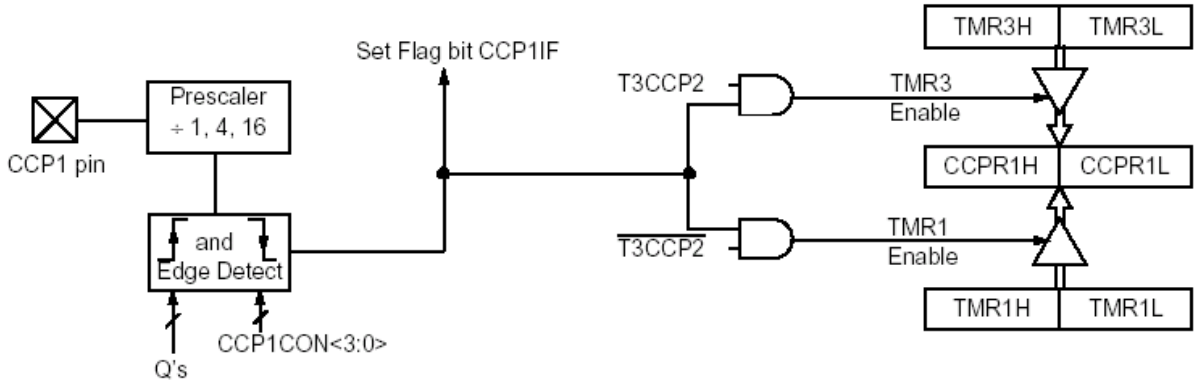
Exercices pour les plus rapides :
A partir de itcomp.c et tstxlcd.c, construire une horloge affichant heures, minutes, secondes. La mise à l'heure se fera dans le débogueur MP-LAB
Dans un deuxième temps la mise à l'heure se fera par S2 pour les minutes et S3 pour les heures. ***(EX10)***



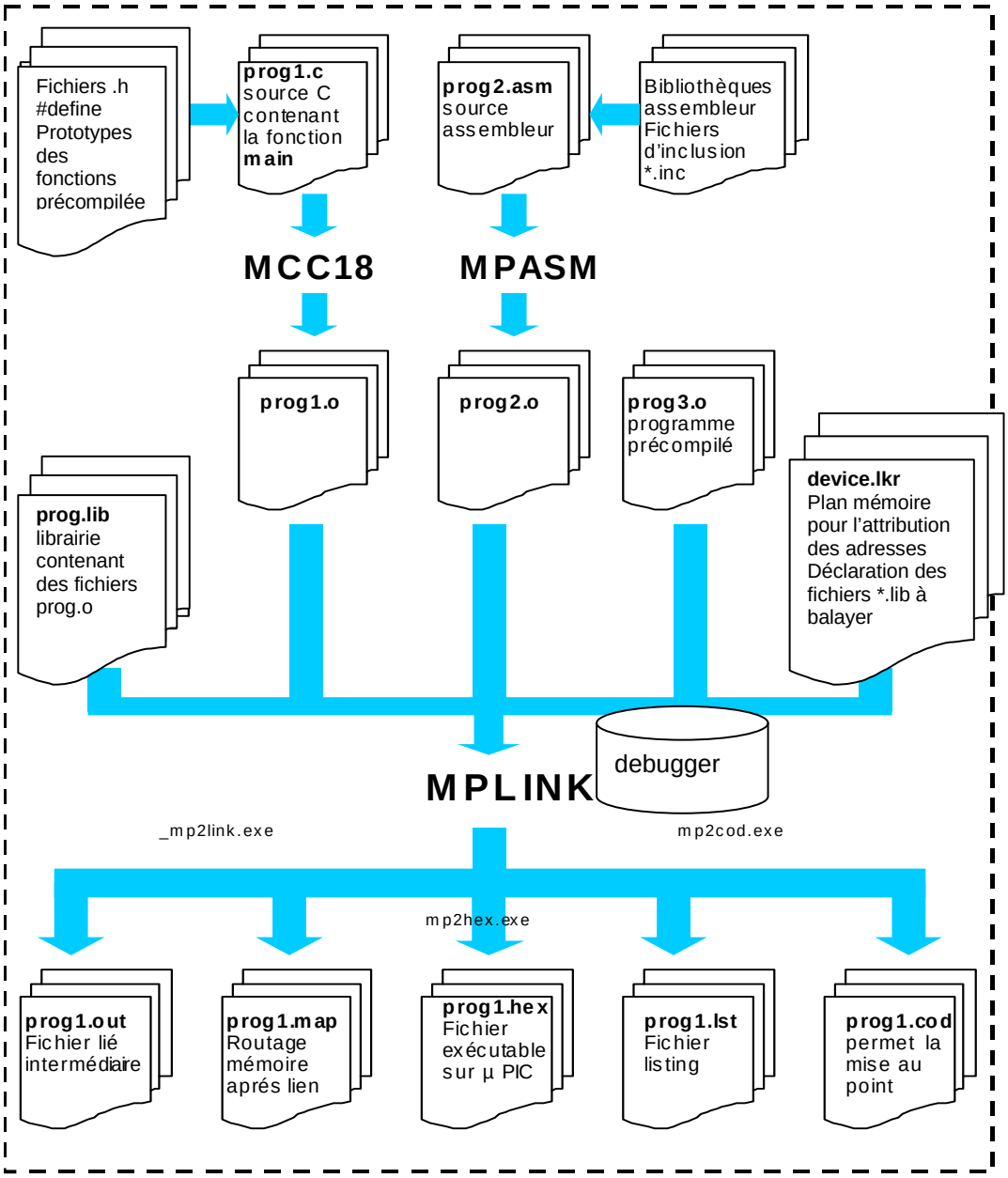
6.4.2. Mesure de temps

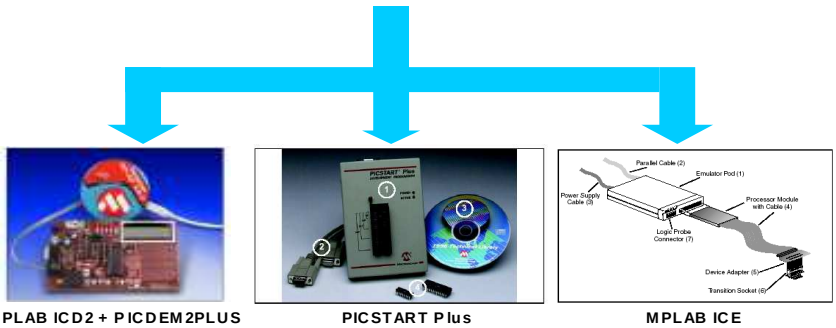
```
// mesure de période sur CCP1 (RC2) (TIMER1 et fonction capture) (fichier itcapt.c)
// initxlcd.c doit être compilée dans le projet pour la gestion de l'afficheur LCD
#include <p18fxxx.h>
#include <xlcd.h>
#include <stdio.h>
#include <tempo_lcd_pd2.c> // tempo pour xlcd.h

unsigned int duree=5555;      // représente le comptage entre 2 fronts
char maj=1;                   // indique qu'une nouvelle mesure est prête

// sous programme d'interruption
#pragma interrupt itcomp
void itcomp(void)
{
unsigned static int ancien;
        if(PIR1bits.CCP1IF)           // l'IT provient d'une capture
                        {
                        duree=CCPR1;
                        maj=1;                  // nouvelle mesure pr^te
                        }
        PIR1bits.CCP1IF=0;              //efface le drapeau d'IT
}

#pragma code interruption=0x8
void ma_fontion (void)
{
_asm goto itcomp  _endasm
}
#pragma code

void main(void)
{
// configure PORTC CCP1
        DDRCbits.RC2=1;  // RC2/CCP1 en entree

// configure le TIMER1
        T1CONbits.RD16=0;               // TMR1 mode simple (pas de RW)
        T1CONbits.TMR1CS=0;             // compte les impulsions sur internal clock
        T1CONbits.T1CKPS1=1;    // prédiviseur =1/8 periode sortie = 8uS
        T1CONbits.T1CKPS0=1;
        T1CONbits.T1SYNC=1;             // pas de synchronisation sur sleep/Reset
        T1CONbits.TMR1ON=1;             // TMR1 Activé

// configure le mode capture sur le TIMER1 avec IT sur CCP1
        T3CONbits.T3CCP2=0;             // mode comparaison entre TMR1 et CCPR1
        CCP1CON=0x05;                   // capture mode sur fronts montants

        PIE1bits.CCP1IE=1;              // active IT sur mode capture/comparaison CCP1

        RCONbits.IPEN=1;        // Interruption prioritaires activées
        INTCONbits.GIE=1;       // Toutes les IT démasquées autorisées
        XLCDInit( );
        while(1)
                {
                if (maj) {
                                XLCDL1home();
                                fprintf(_H_USER,"ccp1= %u    " ,duree);
                                maj=0;
                                }
                }
}
```

Ex9 : Réaliser un fréquencemètre sur afficheur LCD.
Définir en fonction des paramètres de TIMER1 les fréquences max et min mesurables.



7. Structure d'un projet dans MPLAB, gestion des bibliothèques





MPLAB ICD2 + PICDEM2PLUS MPLAB SIM | PICSTART Plus | MPLAB ICE

Simulateur/ In Circuit Debugger/ Emulateur/ Programmateur

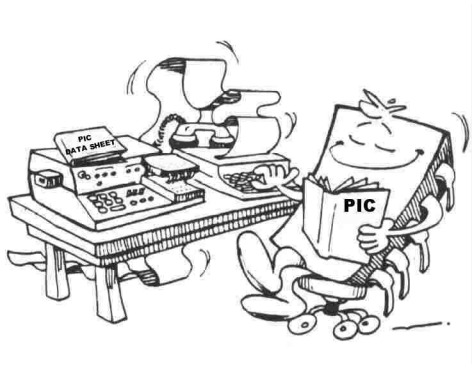
MPLAB IDE est un environnement de développement intégré de projets logiciels. Associé au compilateur MCC18 il permet de compiler et/ou d'assembler ensemble des fichiers sources d'origines différentes puis de lier les fichiers objet obtenus entre eux et avec des bibliothèques précompilées, à partir d'un fichier d'édition de lien caractéristique du processeur utilisé, pour obtenir le fichier exécutable.

7.1. TP N°6 : Gestion des périphériques intégrés

(Travail individuel, Durée :1h30)

Objectifs :

- Mettre en œuvre le convertisseur analogique numérique (voltmètre)
- Utiliser l'EEPROM interne en lecture/écriture
- Mettre en œuvre les communications séries asynchrones (USART liaison avec terminal.exe sur PC)
- Mettre en oeuvre l'interface I2C (mesure de température)



Prérequis :

- Caractéristiques générales du compilateur MCC18
- Connaissance élémentaire du langage C
- Notions d'algorithmique
- Architecture du µcontrôleur PIC 18F4620

Données :

- Documentation minimale PIC 18F4620
- Guide d'utilisation de la carte PICDEM2 PLUS

Remarque préalable :

La durée prévue pour ce TP ne permet pas de solutionner les exercices proposés. Il s'agit donc dans un premier temps de mettre en œuvre les différentes interfaces avec les programmes proposés puis dans un deuxième temps de solutionner un ou plusieurs exercices.

Travail demandé :

1 Convertisseur analogique numérique
2 EEPROM interne
3 Communications asynchrones
4 BUS I2C
5 BUS SPI

7.2. Conversion analogique/Numérique

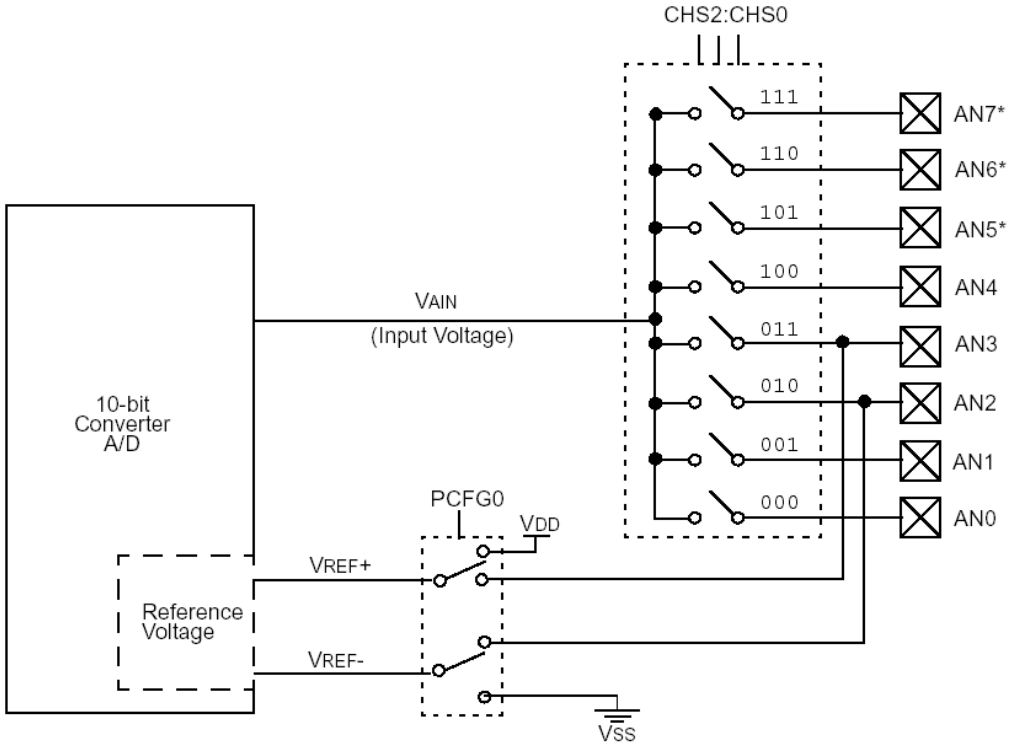
Etre capable de créer une fonction paramétrée sur un périphérique d'E/S

Programme **atod.c**
atod.c montre comment mettre en œuvre le convertisseur analogique numérique du PIC18F4620.
La tension sur l'entrée AN0 est affichée en volts.

```
#include "xlcd.h"
#include <stdio.h>
#include <tempo_lcd_pd2.c> // tempo pour xlcd.h

#include "ftoa.c"

#define q 4.8828e-3     // quantum pour un CAN 10bits 0v-5v

char chaine[30];

void main(void)

{
float res;
      XLCDInit();
      XLCDL1home() ;    // positionne le curseur en x,y
      ADCON0=1;                    // CAN on. CLOCK=FOSC/2. CANAL0 (RA)
      ADCON1=0x8E;                 // justification à droite, seul AN0 est
activé, VREF+=VDD VREF-=VSS

      while(1){
                  ADCON0bits.GO_DONE=1;         // SOC
                  while(ADCON0bits.GO_DONE);    // attend EOC
                  res=(float)ADRES*q;                 // calcule la tension
                  ftoa(res,chaine,3,'f');       // convertit en chaine
                  XLCDL1home();
                  XLCDPutRamString(chaine)
            }
}
```

Ex8 : Réaliser un voltmètre affichant la tension sur AN0 en volts en introduisant une fonction int mesvolt(char canal) retournant la valeur mesurée sur l'entrée « canal » et en utilisant xlcd.h et stdio.h

Envoie vers l'afficheur LCD une chaîne depuis la RAM



7.3. Accès EEPROM interne

Etre capable d'utiliser l'EEPROM interne en lecture/écriture

Programme **eeprom.c**

```
#include <p18fxxx.h>

char chaine1[]="j'ecris en EEPROM";
char *chaine2;
unsigned int adresse;
char c;

char eeplit(unsigned int ad)       // lecture de l'adresse ad
{
      EEADR=ad;
      EECON1bits.EEPGD=0;
      EECON1bits.RD=1;
      return(EEDATA);
}

void eepecr(unsigned int ad,unsigned char c) // ecrit  c à l'adresse ad
{
      EEADR=ad;
      EEDATA=c;

      EECON1bits.EEPGD=0;
      EECON1bits.WREN=1;
      EECON2=0x55;
      EECON2=0xAA;
      EECON1bits.WR=1;
      EECON1bits.WREN=0;
}

void eepmess(unsigned int ad, unsigned char *p) // écrit une chaine p à l'adresse ad
{
      while (*p) eepecr(ad++,*p++);
}

void main(void)
{
      eepmess(0,chaine1); // ecrit chaine1 à l'adresse 0 de l'EEPROM
      adresse=0;
      while(c=eeplit(adresse++)) *chaine2++=c;      //recopie en RAM l'EEPROM
      while(1);

}
```

Exercice : tester ce programme et constater l'écriture et la recopie de la chaîne dans les fenêtres « files registers » et « EEPROM »

Exercice : : réaliser le même programme à l'aide de la bibliothèque eep.h

Etre capable de mettre en œuvre les communications séries asynchrones

7.4. Communications séries asynchrones

Programme tstusart.c

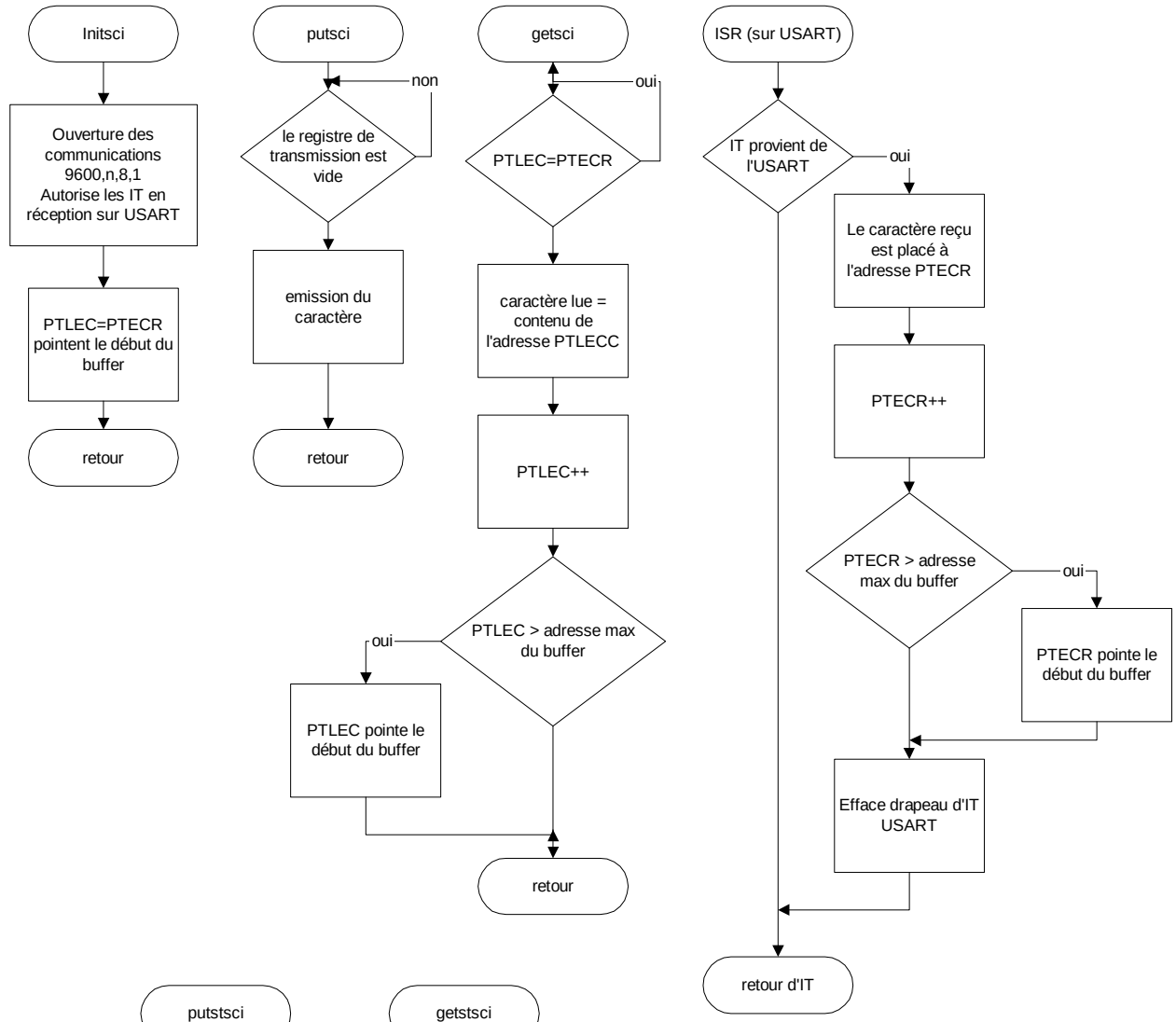
Tstusart montre la mise en œuvre des communications asynchrones.
Ce programme **ne traite pas la perte de données en réception par écrasement**

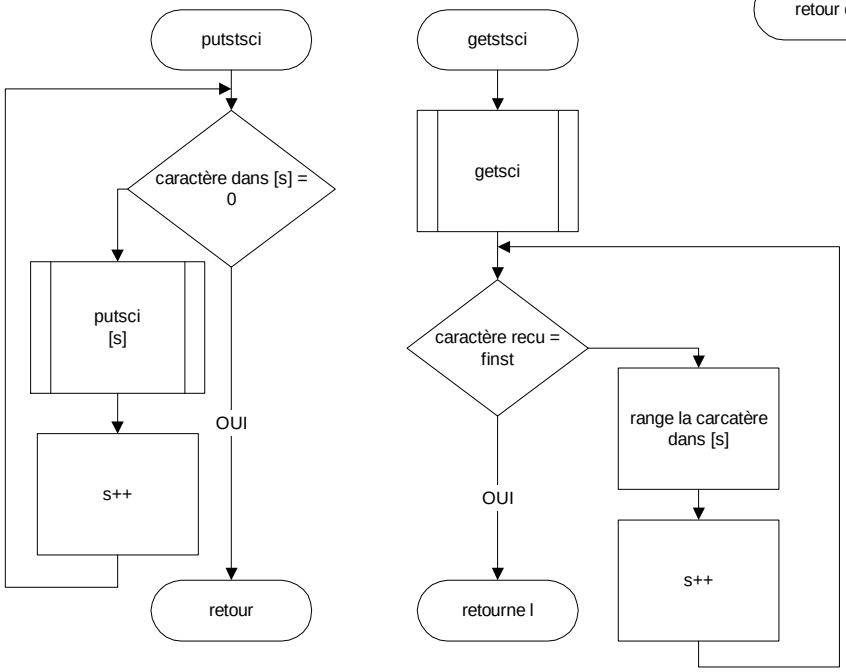
```
// CD 03/03
// Test des communications asynchrones sans IT
// connecter un émulateur de terminal sur le pour série de PICDEM2+
// Attention au cable PC (brochage RX/TX)
#include <p18fxxx.h>

rom char mess[]="\nLes communications sont ouvertes\nTapez une touche
...\n\n";

// indique qu'un caractère est dans RCREG de l'USART
char data_recue(void)              // reception d'une interruption
{
      if (PIR1bits.RCIF)           /* char recu en reception*/
      {
            PIR1bits.RCIF=0;  // efface drapeau
            return (1); // indique qu'un nouveau caractère est dans RCREG
      }
      else return (0);       // pas de nouveau caractère reçu
}

// envoie un caractère sur USART
void putch(unsigned char c)        //putch est défini sur le port série
{
      while(!TXSTAbits.TRMT); // pas de transmission en cours ?
    TXREG=c;                          /* envoie un caractère */
    while(!PIR1bits.TXIF);
}

// envoie une chaine en ROM
void putchaine(rom char* chaine)
{
      while (*chaine) putch(*chaine++);
}

void main(void)
{
    SPBRG = 25;              /* configure la vitesse (BAUD) 9600 N 8 1*/
    TXSTA = 0x24;
    RCSTA = 0x90;                  /* active l'USART*/

    putchaine(mess);                    // intro
    while(1)                            // echo
          {
                  if (data_recue()) putch(RCREG }
          }
}
```

Bibliothèque libusart.h

Cette bibliothèque contient toutes les fonctions de gestion de l'USART et **évite la perte de donnée en réception par écrasement.**
Chaque caractère reçu déclenche une interruption qui stocke ce dernier dans un tampon mémoire. getsci lit dans ce tampon le plus ancien caractère reçu.

Bibliothèque libusart.c – Algorigrammes





Etre capable d'analyser un programme de traitement de données par pointeurs. Utiliser la librairie libpd2.h

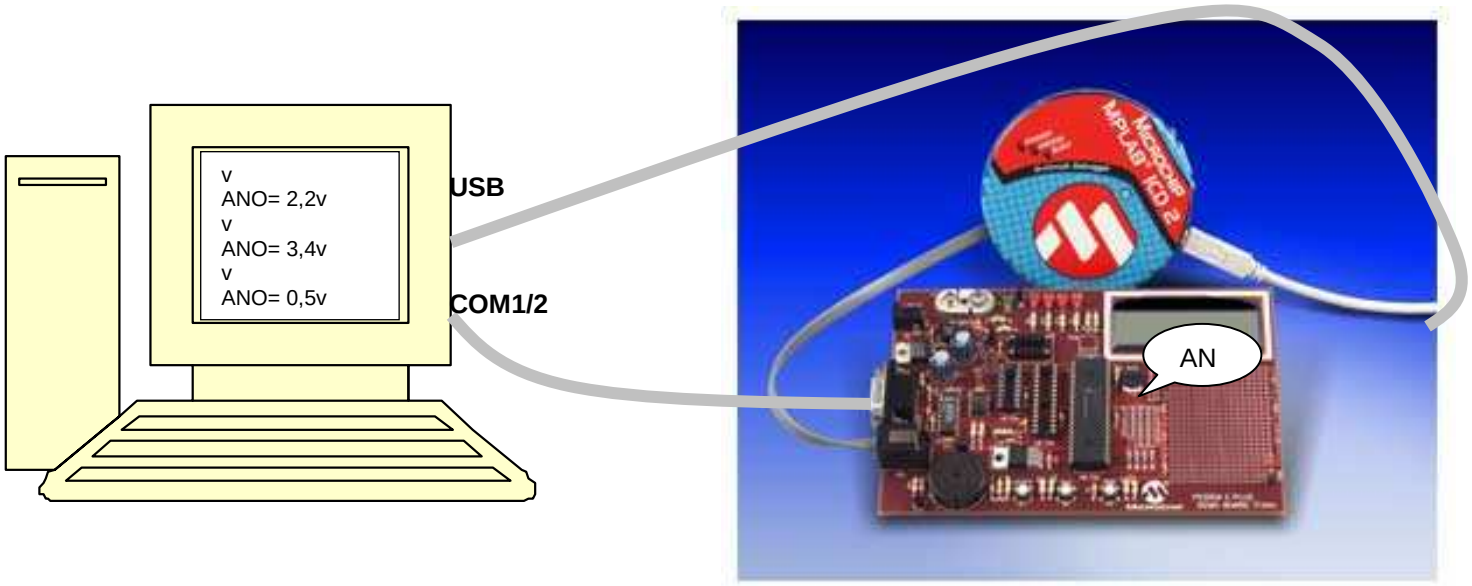
Tableau de réception :
Les caractères reçus sont rangés dans un tableau (buffer) à l'adresse d'un pointeur PTECR qui est ensuite incrémenté. getsci retourne le caractère ponté par PTRECR puis PTRECR est incrémenté.
Lorsque tous les caractères reçus ont été lus, PTERC=PTLEC. Si l'un des pointeurs dépasse l'adresse max du tableau il pointera à nouveau le début
[PTLEC-1] représente le dernier caractère lu et [PTECR-1] le dernier caractère reçu.

0	1	2	3	4	5	6	7	8	9	…	38	39
36	54	4A	41	6D	43	8B	55	65	30	…	69	7E
			PTLEC					PTECR				

Programme **tstusartlib.c**
Exemple d'utilisation de usartlib.c

```
/* gestion SCI en IT
/* test de la librairie libpd2*/
#include "initxlcd.c"
#include <p18fxxx.h>
#include "libusart.c"

void main(void)
{
unsigned char chaine[]="Bonjour les communications sont ouvertes\n";
unsigned char maj[]="Afficheur LCD mis à jour \n";
    OpenXLCD(FOUR_BIT & LINES_5X7 );
      initsci();
      putstsci(chaine);
    while(1)
    {       // mettre une des 2 lignes ci dessous en commentaires
 //   putsci(getsci()+1);           // emmision / réception d'un caractère
//    putstsci(getstsci(chaine,'*'));// emmision / réception d'une chaine
      getstsci(chaine,'*');
      SetDDRamAddr(0);
      putsXLCD(chaine);
      putstsci(maj);
      }
}
```

Ex11 : réaliser un voltmètre sur PC mesurant de tension sur AN0 et la transférant sur USART lors de la réception du caractère 'v'



Etre capable de mettre en oeuvre l'interface I2C

7.5. Bus I2C

Exemple de gestion du module MSSP (Master Synchronous Serial Port) en mode I2C. Lecture de la température sur le capteur TC74 de PICDEM2+ (fichier I2Ctc74.C)

Programme **i2cTC74.c**

Read Byte Format

S	Address	WR	ACK	Command	ACK	S	Address	RD	ACK	Data	NACK	P
	7 Bits			8 Bits			7 Bits			8 Bits		

Slave Address — Command Byte: selects which register you are reading from. — Slave Address: repeated due to change in data-flow direction. — Data Byte: reads from the register set by the command byte.

```
// test TC74 sur picdem2+
// C.D 02/2003
#include <p18fxxx.h>
#include "initxlcd.c"

#define adrtc74 0b1001101
#define regtemp 0
#define config 1
signed char temp;                       // variable température
unsigned char tampon[3];                // mémoire pour les chaines converties avec BTOA

void ack(void)                                  // attend acknowledge (I2C) de l'esclave
{
while(SSPSTATbits.R_W);                         // attend fin de transmission
while (SSPCON2bits.ACKSTAT);  // attend fin ACK esclave
}

//retourne le contenu du registre cmd dans TC74
signed char lit_i2c(unsigned char adresse, unsigned char registre)
{ signed char t;
 SSPCON2bits.SEN=1;                     // START
 while (SSPCON2bits.SEN);
 SSPBUF=adresse<<1;             // adresse ecriture
 ack();
 SSPBUF=registre;                               // adresse registre
 ack();
 SSPCON2bits.RSEN=1;            // RESTART
 while (SSPCON2bits.RSEN);
 SSPBUF=(adresse<<1)|0b00000001;                // adresse lecture
 ack();
 SSPCON2bits.RCEN=1;            // passe ne mode lecture d'un octet
 while (SSPCON2bits.RCEN);  // attend reception terminée
 t=SSPBUF;                                      // mémorise température
 SSPCON2bits.ACKDT=1;           // NON-ACK
 SSPCON2bits.ACKEN=1;
 while(SSPCON2bits.ACKEN);
 SSPCON2bits.PEN=1;                     // STOP
 while(SSPCON2bits.PEN);
 return (t);
}

void init_i2c(void)
{       DDRCbits.RC3 = 1;               // SCL (PORTC,3) en entrée
        DDRCbits.RC4 = 1;               // SDA (PORTC,4) en entrée
        SSPCON1=0b00101000;             // WCOL SSPOV SSPEN CKP SSPM3:SSPM0
 // efface WCOL et SSPOV, active I2C, I2C mode maitre horloge=FOSC/(4*(SSPADD+1))
        SSPSTATbits.SMP=1;              // slew rate inhibé (f<400Khz)
        SSPADD=5;                       // horloge = 4Mhz / 24 = 166,66 KHz
}

void main(void)
{ OpenXLCD(FOUR_BIT & LINES_5X7 );
 init_i2c();
while (1)
        {
        while(!(lit_i2c(adrtc74,config)&0b01000000));
// attend mesure ok (Bit D6 (du registre CONFIG TC74) =1)
        temp=lit_i2c(adrtc74,regtemp);
// le résultat est direct (codé signed char), voir doc TC74
    SetDDRamAddr(0);
    putsXLCD(btoa(temp,tampon));     // écrit un byte  (8 bits)
    putchar('c');
        }
}
```

Exercices :

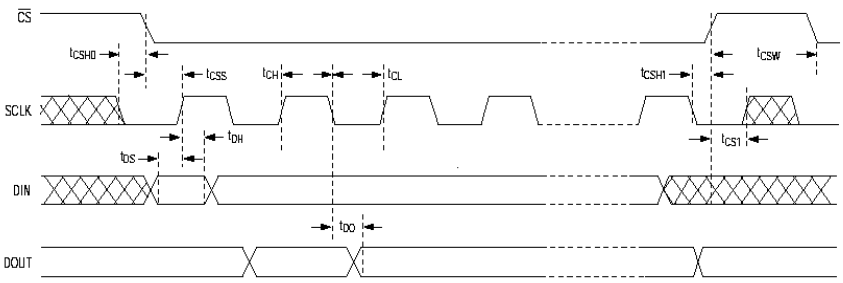
- A l'aide de la documentation du PIC18F4620 (chap 15 MSSP). Réalisez l'algorigramme de ce programme.

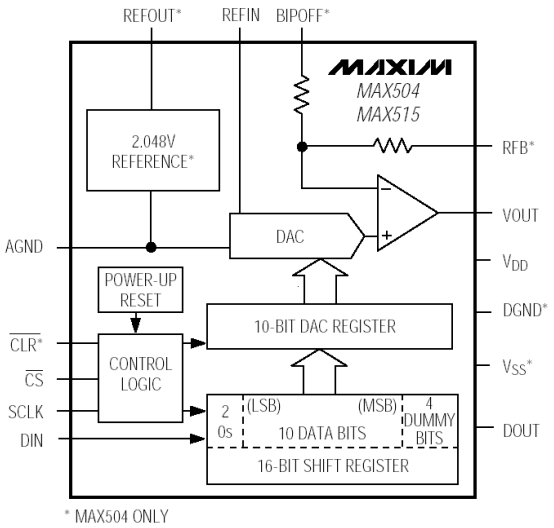
Il sera essentiel d'analyser la configuration des registres et bits utilisés

- Réaliser le MÊME programme mais en utilisant la librairie libpd2.h **Ex12**
- Ecrire un programme transmettant la température toutes les secondes sur l'USART

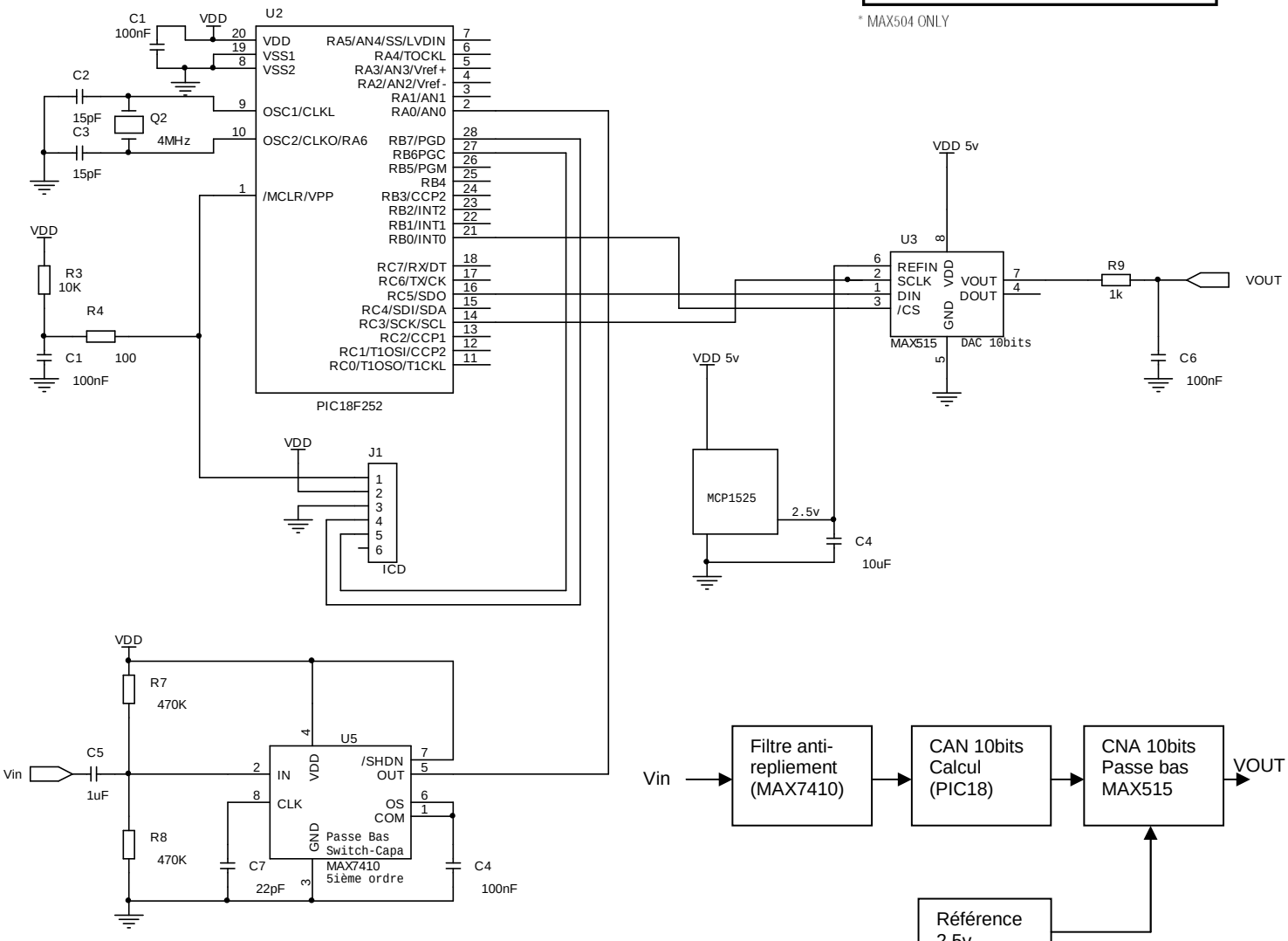
7.6. Bus SPI

Connexion d'un convertisseur analogique numérique 10 bits sur bus SPI : MAX 515





Exemple de connexion (filtre numérique):



Exercices :

- Après câblage, tester le programme page suivante
- **Ex 13 :** Créer un programme recopiant Vin sur Vout avec fe=1Khz.

```
// CD Lycee Fourcade 13120 Gardanne 01/2003
/* Librairie pour MAX515 sur BUS SPI (PIC18)
 initSPI_max515 initialise le port SPI pour MAX515 avec F=Fosc/4
 Selection boitier sur /SS (PORTA5) pas d'interruption
 void max515(unsigned int v) envoie la valeur v(0<=v<=1023) vers le max515
Brochage MAX515 CNA 10 bits
1 – DIN sur RC5/SDO
2 – SCLK sur RC3/SCK
3 - /CS sur RA5 (ou ailleur)
4 – DOUT (non connecté)
5 – GND
6 – REFIN (ref 2,5v Microchip MCP1525 par exemple)
7 – Vout (sortie 0-5v du CNA)
8 – VDD (5v)
*/

#include <p18f252.h>

void initSPI_max515(void)  // initialisise SPI sur PIC18
{
DDRAbits.RA5=0;   // PRA5 en sortie (/SS)
PORTAbits.RA5=1; // CS=1
DDRCbits.RC3=0;   //SCK en sortie
PORTCbits.RC3=0;
DDRCbits.RC5=0;   //SDO en sortie
PORTCbits.RC5=0;
PIR1bits.SSPIF=0;

SSPSTAT=0b01000000; //echantillonne au milieu de la donnée, sur front montant
SSPCON1=0b00100000;// active SPI, IDLE=0, clock=FOSC/4
PIR1bits.SSPIF=0; // SSPIF indique une fin d'emmission par un 1
}

void max515(unsigned int v)   // envoie v sur CAN MAX515
{unsigned char fort,faible;  // poids forts et faibles de v
            v<<=2;// formatage des données pour compatibilité avec MAX515
            fort=v>>8;
            faible=v & 0b0000000011111111;

            PORTAbits.RA5=0; // CS=0
            SSPBUF=fort;            // emmision poids forts
            while(!PIR1bits.SSPIF); // attend la fin de l'émission
            PIR1bits.SSPIF=0;
            SSPBUF=faible;          // emmisiion poids faibles
            while(!PIR1bits.SSPIF);// attend la fin de l'émission
            PIR1bits.SSPIF=0;
            PORTAbits.RA5=1; // CS=1
}
/* programme de test de la librairie */
void main(void)
{
int val=0x0;

initSPI_max515();
while(1)
            {
            max515(val++);  // incrémente VOUT de q, F dépend du quartz
            }
}
```

7.7. Directives du pré-processeur

Les directives de pré-compilation commencent toutes par le caractère # et ne se terminent pas par un point-virgule .

7.7.1. Directives C ANSI

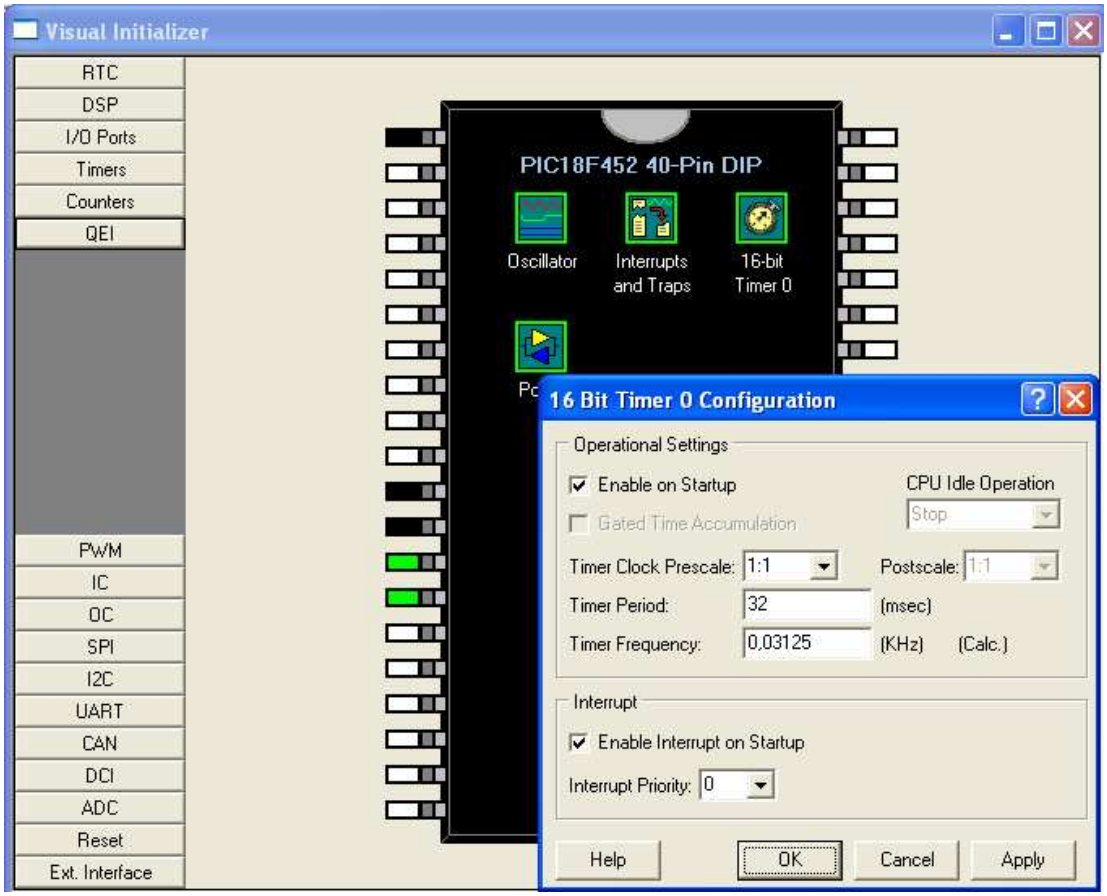
Directive	Rôle	Syntaxe / exemple
#include	Sert à inclure un fichier contenant du code source (.c ou .h) dans un autre fichier.	**#include<Nomfichier>** □ recherche du fichier dans : • Les répertoires mentionnés à l'aide de l'option de compilation /Idirectory • Les répertoires définis à l'aide de la variable d'environnement INCLUDE **#include "Nomfichier"** □ recherche du fichier dans : Idem cas précédent **+** • Le répertoire courant
#define **#undef**	Permet de définir une variable pré-processeur en lui affectant éventuellement un contenu . Partout dans la suite du fichier source, la variable (identificateur) en question sera remplacée par son contenu (valeur). La directive #undef permet de détruire une variable pré-processeur, à partir d'un endroit donné dans le code, et d'arrêter toute substitution liée à cette variable.	**#define identificateur [valeur]** #define PI 3.1416 #define OUTPUT 0x0 #define INPUT 0x1 #define LCD_DATA LATD #define lcd_clear() lcd_cmd(0x1) // efface l'afficheur #define lcd_goto(x) lcd_cmd(0x80+(x)) **Remarque :** Dans ce dernier type de substitution, il est possible d'introduire des paramètres, on parle alors de macro-fonction.

#if
#ifdef
#ifndef
#else
#elif
#endif
	Il s'agit de directives de compilation conditionnelle qui facilitent entre autre la résolution de nombreux problèmes de portabilité des codes "source".	#if type_ecran==VGA
		nb_colonnes=640;
		nb_lignes=480;
		#elif type _écran==SVGA
		nb_colonnes=800;
		nb_lignes=600;
		#elif type _écran==XGA
		nb_colonnes=1024;
		nb_lignes=768;
		endif

#line	A l'intérieur d'un fichier, en introduisant une directive #line, il est possible d'imposer une numérotation des lignes, ainsi éventuellement qu'un nouveau nom de fichier comme indication du code source compilé. Ceci sert essentiellement pour les messages d'erreur de compilation	**#line numéro_de_ligne ["nomfichier"]** **#line 1 "calcul.c"** → Numérotation des lignes à partir de 1 avec comme nom calcul.c **int calcul(int x, int y)** **{** **…** **}** Idem avec affichage.c **#line 1 "affichage.c"**
#error	Permet de générer un message d'erreur qui s'affichera pendant la compilation	**#error message** **#error Type d'écran non défini**

7.8. L'utilitaire graphique VISUAL INITIALISER

Ce module de MPLAB doit être télécharger depuis le site internet de MICROCHIP et installé après MPLAB. Il permet la création rapide d'un squellette de programme en assembleur avec les périphérques pré-initiailisés qui peut être lié simplement avec un programme en C



7.9. L'utilitaire MICROCHIP MAESTRO

Maestro permet de créer le squelette d'un programme en assembleur et en C avec des fonctions de gestion de prériphériques intégrées, (LCD, Bus CAN, I2C, etc…)