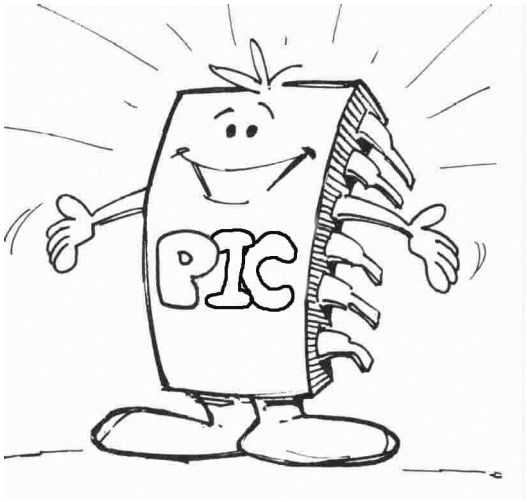
COMPILATEUR

Microchip C18 v18



MICROCHIP www.microchip.com

Ecole Nationale Supérieure des Mines SAINT-ETIENNE
CMP - Georges Charpak

Christian Dupaty
Professeur de génie électrique
Académie d'Aix-Marseille
christian.dupaty@ac-aix-marseille.fr

Jean Yves Rebouche
Professeur de génie électrique
Académie d'Aix-Marseille
jrebouche@ac-aix-marseille.fr

SOMMAIRE

1. CARACTERISTIQUES GENERALES DE MCC18 4

1.1. SCHEMA GENERAL DU PROCESSUS DE COMPILATION 4
1.2. DIRECTIVES DU PRE-PROCESSEUR 5
1.3. ROLE DU PRE-PROCESSEUR 6
1.4. ROLE DES FICHIERS D'INCLUSION (*.H) 6
1.5. FICHIER P18FXXX.H 7
1.6. LA DIRECTIVE #PRAGMA CONFIG 7

2. SPECIFICITES DU COMPILATEUR MCC18 8

2.1. TYPE DE DONNEES 8

2.2. MACROS EN C POUR MICRO PIC 8

2.3. ASSEMBLEUR EN LIGNE 8

3. PRISE EN MAIN DU COMPILATEUR 9

3.1. CREATION D'UN PROJET 9
3.2. LES DEBBUGERS 10
3.2.1. ICD2 ET ICD3 10
3.2.2. PICKIT3 11
3.2.3. PICKIT3 – CARTE DE DEMO 12
3.2.4. PROTEUS – VSM 13

3.3. GESTION DES PORTS PARALLELES 14

3.4. MISE AU POINT D'UN PROGRAMME ECRIT EN C DANS MPLAB 15

3.5. CREATION D'UNE FONCTION 16

3.6. ANALYSE D'UN PROGRAMME ECRIT EN C18 : DECALAGES 17

4. BIBLIOTHEQUES MCC18 18

4.1. EDITEUR DE LIENS MPLINK 18

4.1.1. ROLE ET CONTENU DES FICHIERS D'EDITION DE LIEN 18

4.1.2. CODE DE DEMARRAGE (CRT – C RUN TIME) 19

4.2. BIBLIOTHEQUES SPECIFIQUES D'UN PROCESSEUR 19

4.3. FONCTIONS C ANSI 20

4.4. MATH.H 22

4.5. BIBLIOTHEQUES : LES SORTIES DE TEXTE 23

4.5.1. FTOA 24

4.5.2. SORTIE DE TEXTE SUR LE SIMULATEUR DE MPLAB 24

10.1.1. EXEMPLE DE SORTIE TEXTE, LE CALCUL DE RACINES CARREES 26

10.1.2. EXEMPLE SUR MATH.H 27

1. GESTION DE LA MEMOIRE 28

10.2. DIRECTIVES DE GESTION DE LA MEMOIRE 28

5.2. QUALIFICATIFS DE MEMORISATION 29

6. PIC18F4620 – CONFIGURATION DE L'HORLOGE INTERNE 32

7. GESTION DES INTERRUPTIONS 33

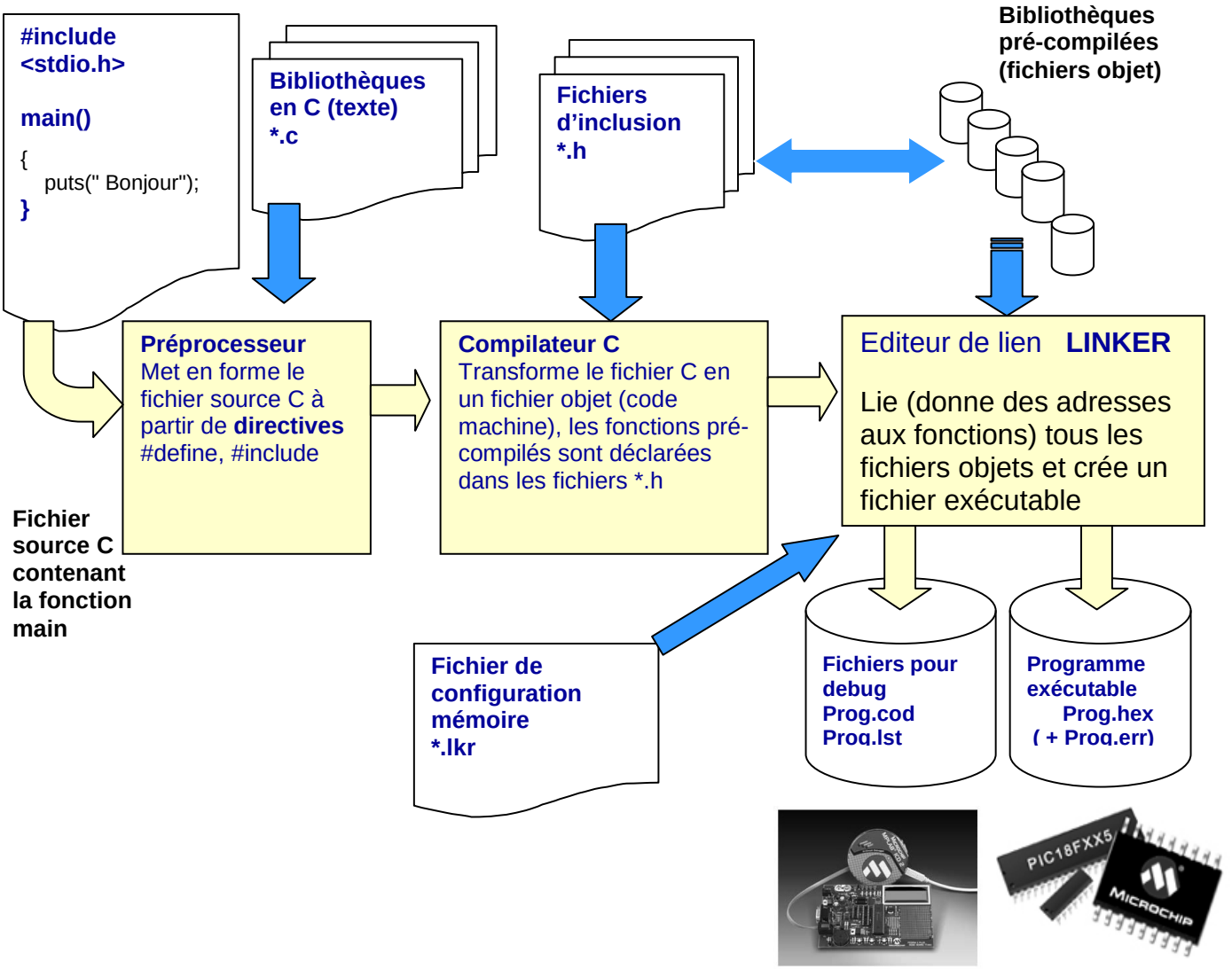
7.1. DIRECTIVES DE GESTION DES INTERRUPTIONS 33
7.2. EXEMPLE DE PROGRAMMES FONCTIONNANT EN IT 34
7.2.1. AVEC LE PORTB : PROGRAMME DEMO_IT_RB0.C 34
7.2.2. AVEC LE TIMER 0 : PROGRAMME FLASHIT.C 35

8. PERIPHERIQUES ET C18 36

8.1. TIMER1 PRODUCTION DE TEMPS 36
8.2. TIMER1 MESURE DE TEMPS 37
8.3. CONVERSION ANALOGIQUE/NUMERIQUE 38
8.4. COMMUNICATIONS SERIES ASYNCHRONES (P18FXX20) 39
8.5. PWM 46
8.5.1. PRINCIPES 46
8.5.2. STRUCTURE DE LA FONCTION PWM SUR PIC 47
8.5.3. EXEMPLE DE MISE EN ŒUVRE DU PWM SUR PIC45K20 48
8.6. INTERFACE I2C 49
8.6.1. Description 49
8.6.2. Structure interface I²C (MSSP : Master Synchronous Serial Port) 50
8.7. BUS SPI 55

1. Caractéristiques générales de MCC18

1.1. Schéma général du processus de compilation



1.2. Directives du pré-processeur

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

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)** **{** **…** **}** **#line 1 "affichage.c"** Idem avec affichage.c
#error	Permet de générer un message d'erreur qui s'affichera pendant la compilation	**#error message** **#error Type d'écran non défini**

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)

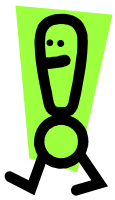
Les fichiers d'inclusion ou d'en tête ***.h** (*header*) contiennent pour l'essentiel cinq types d'informations :

- 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"**

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

#include <p18f4620.h>

Voir page suivante

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 :

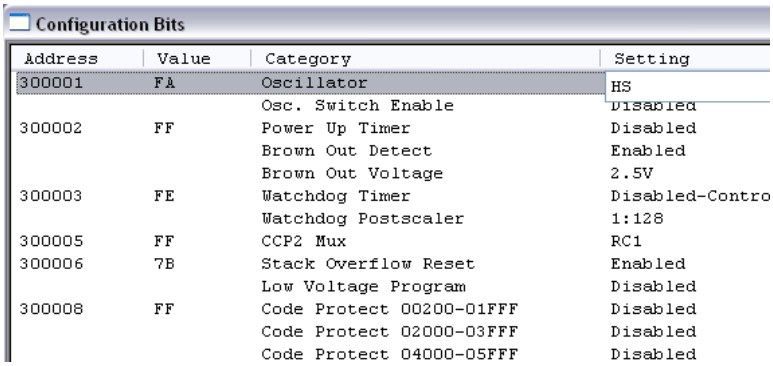
Nom_union.nom_bit

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

1.6. La directive #pragma config

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

Exemple pour un P18F4620:
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
#pragma config DEBUG = ON



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
```

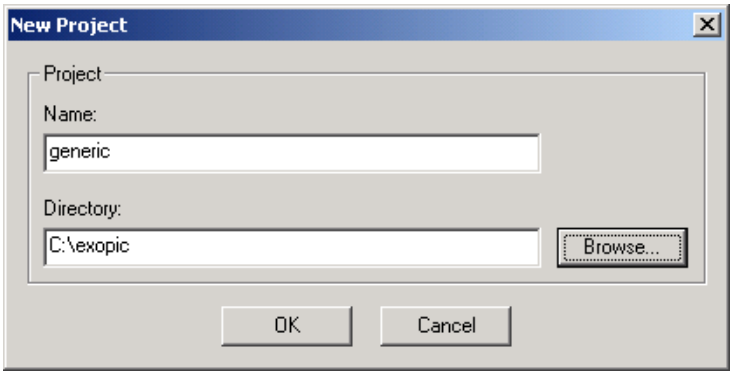
compte est une variable déclarée dans le fichier C

3. Prise en main du compilateur

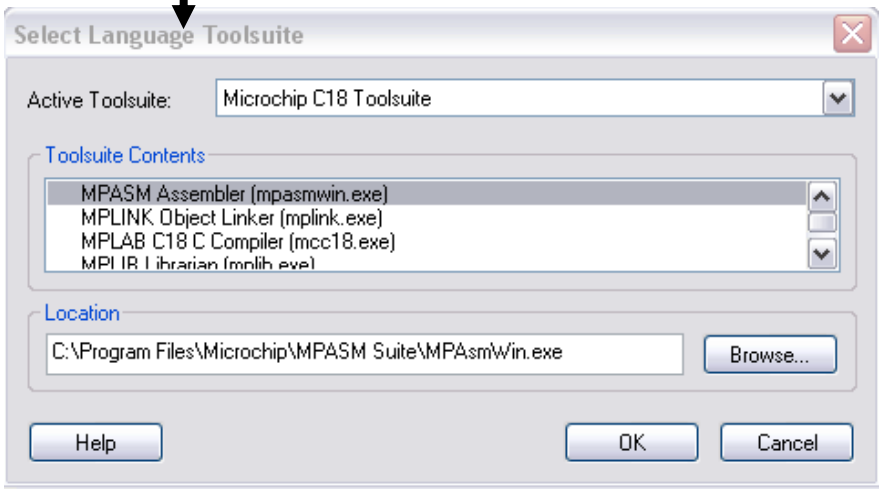
3.1. Création d'un projet

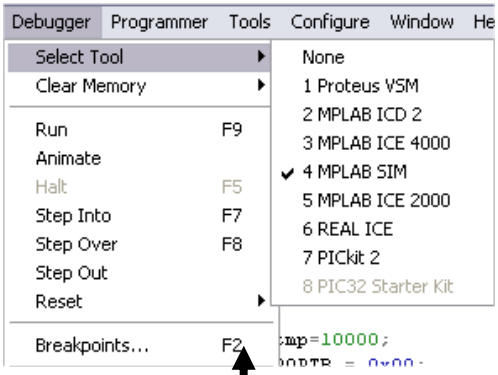
Création d'un **projet** générique en C, ce projet pourra servir pour tester **tous** les programmes exemples et effectuer les exercices.

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 »





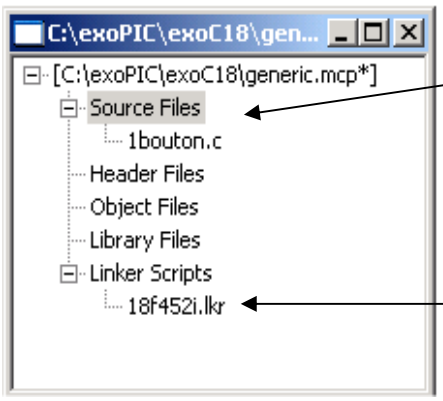
Sélectionner le type de debugger (en général ICD2, ICD3, PICKIT3)

Chemins des programmes : (habituellement)
Assembleur : C c:\mcc18\mpasm\mpasmwin.exe
Linker : C:\mcc18\bin\mplink.exe
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 »

Project - Build options - project - directory , cliquer defaults



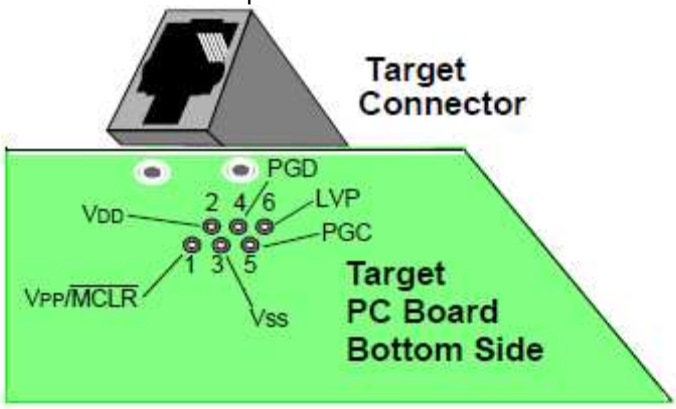
Sources Files contient les fichiers sources en C à compiler. Pour essayer les exemples qui suivent. Placer ici le fichier à compiler.

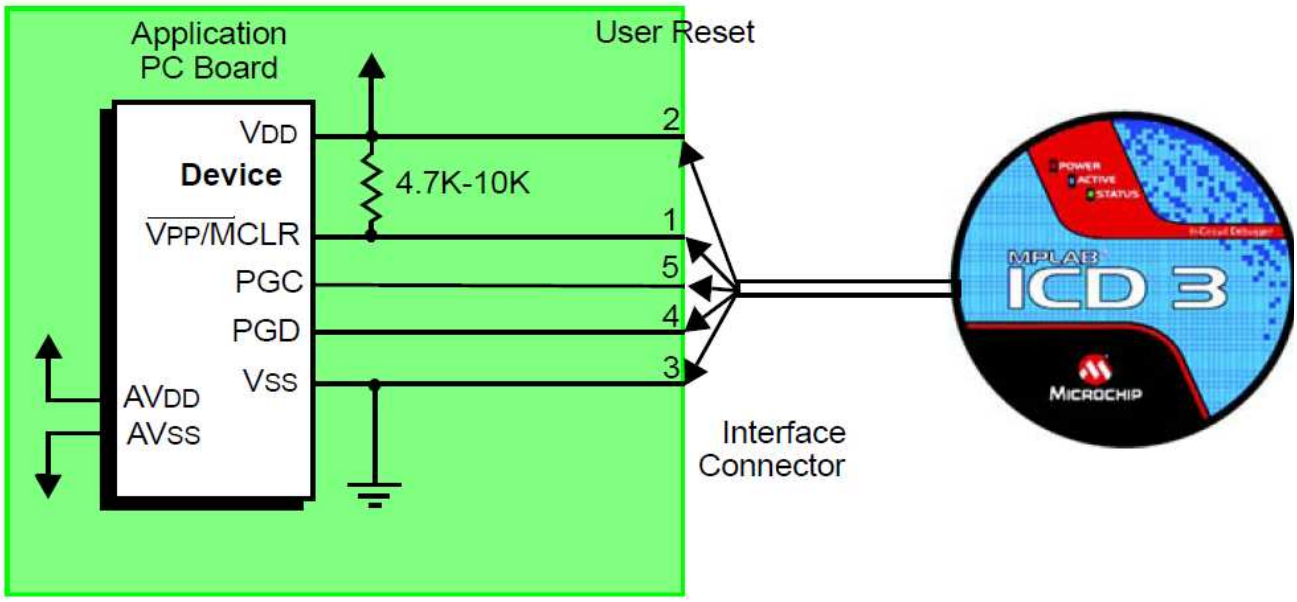
Pour les anciennes versions de MPLAB.
Un fichier d'édition de lien (.lkr) est un fichier de commande pour contrôler les opérations d'édition de lien par MPLINK pour un processeur cible donné ; ici un 18f452 (le i indique une configuration pour ICD2).
CETTE DECLARATION EST INUTILE A PARTIR DE LA VERSION 8.50

3.2. Les debbugers

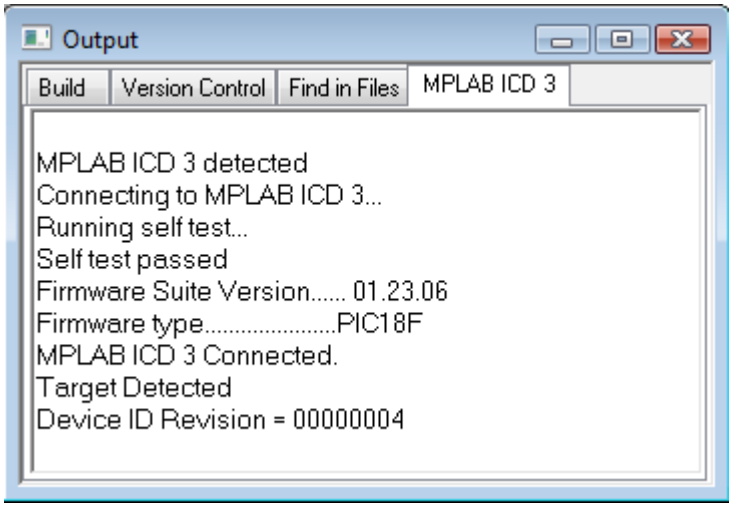
3.2.1. ICD2 et ICD3

Le développement de programmes sur une cible physique peut s'effectuer à l'aide de l'ICD (In Circuit Debug)
Dans ce cas un petit programme de communication (protocol Microchip) est ajouté au programme de l'utilisateur.
Connecter un ICD sur le port USB du PC. Coté cible l'ICD doit être connecté comme ci-dessous

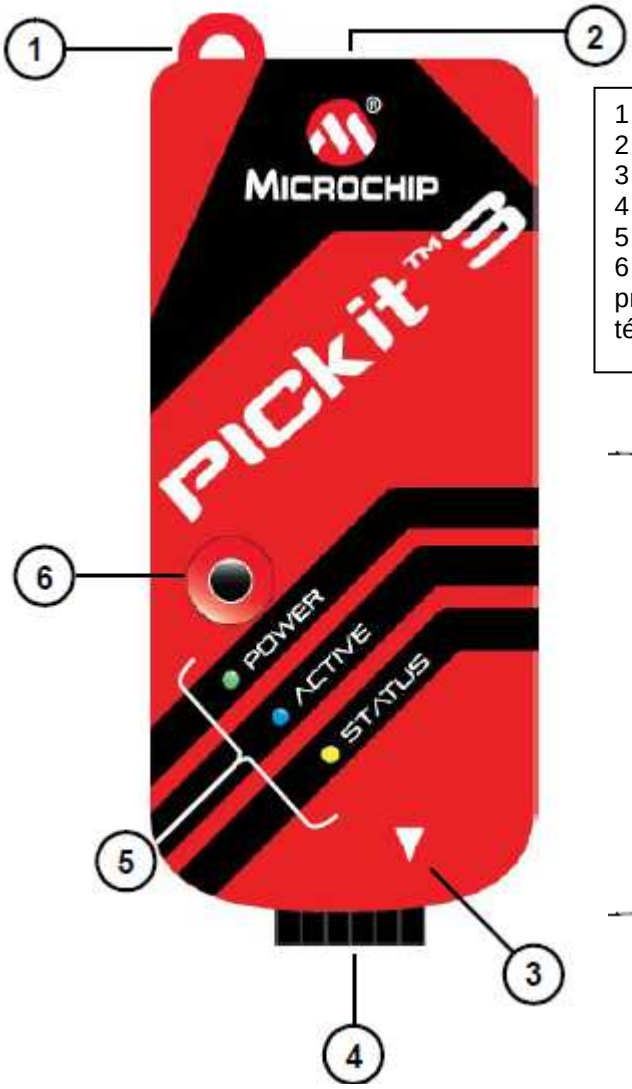




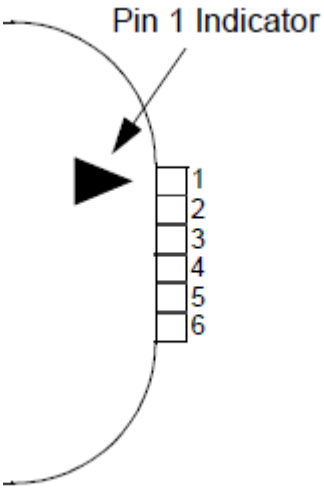
Dans MPLAB, sélectionner le debugger ICD2 ou ICD3
La fenêtre « OUTPUT » donne



3.2.2. PICKIT3

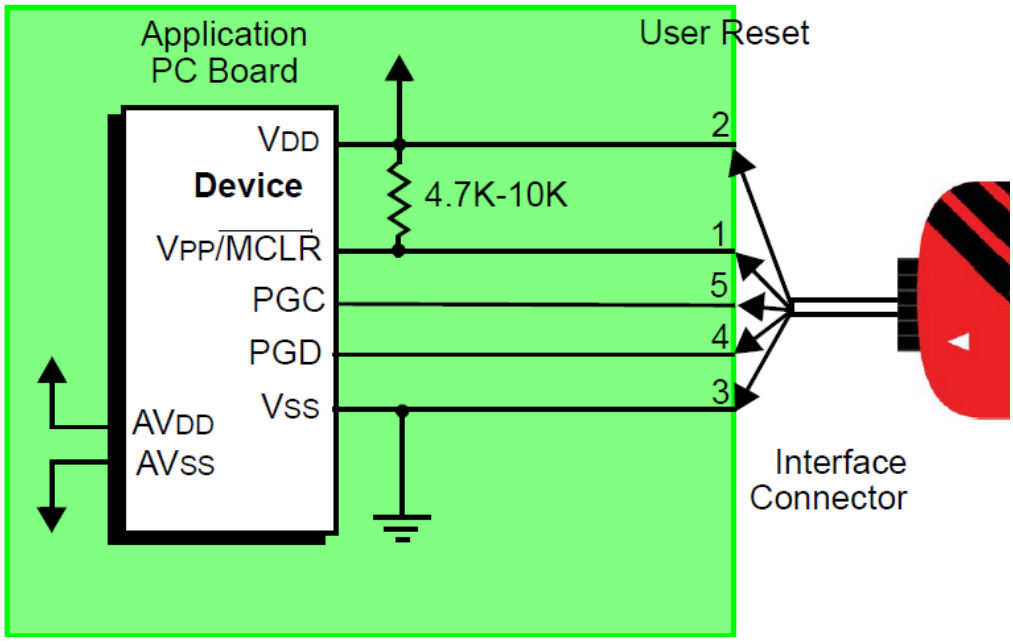


1 : Boucle pour dragonne
2 : USB vers PC
3 : Repère broche 1 du connecteur 4
4 : Connecteur vers le uC cible
5 : LEDs d'états
6 : Fonction « Programmer-To-Go », permet de programmer un PIC avec une image préalablement téléchargée dans le PICKIT3



Pin Description*

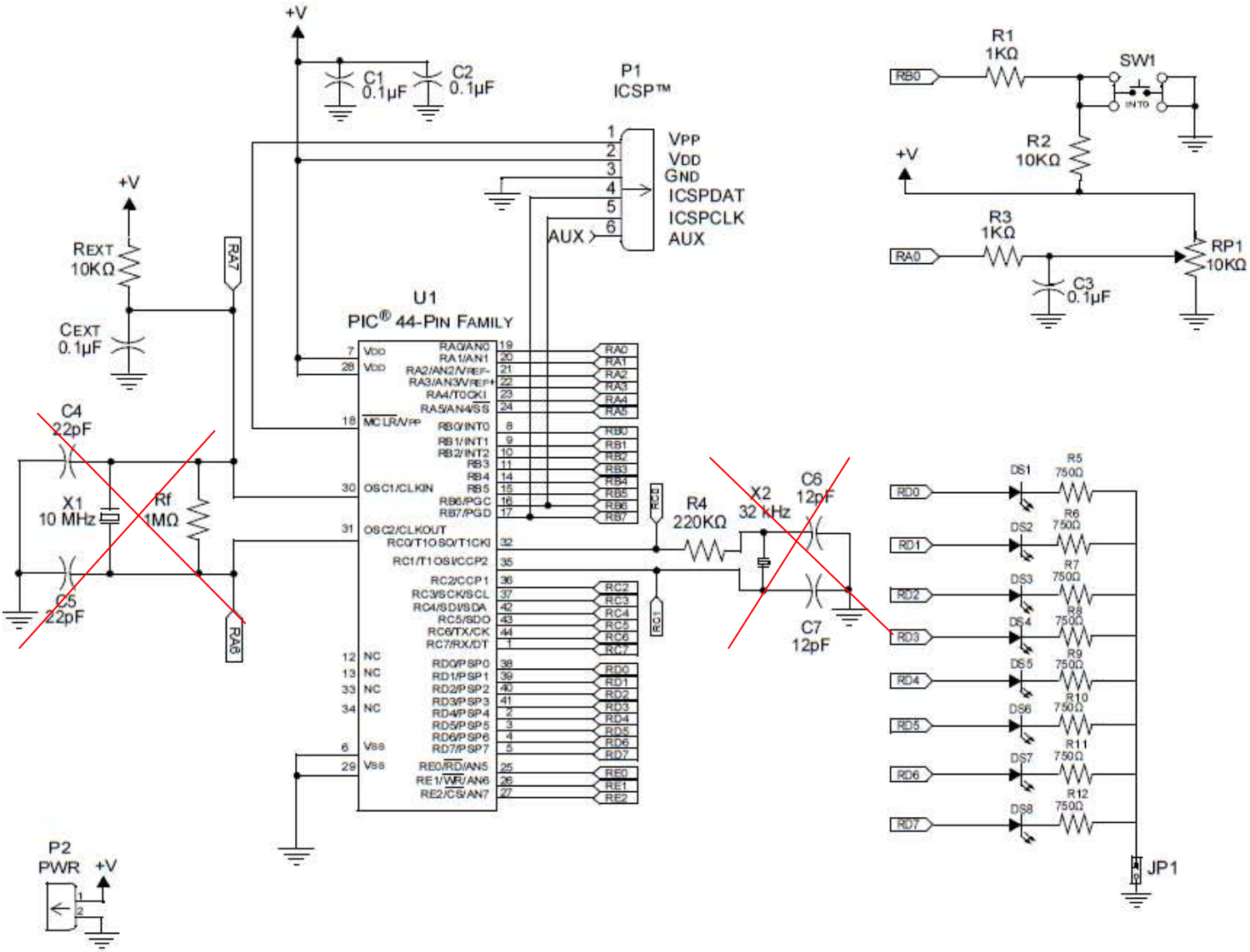
1 = $\overline{\text{MCLR}}$/VPP
2 = VDD Target
3 = VSS (ground)
4 = PGD (ICSPDAT)
5 = PGC (ICSPCLK)
6 = PGM (LVP)



3.2.3. PICKIT3 – CARTE DE DEMO

Le PICKIT3 est accompagné d'une carte de démonstration équipée d'un PIC18F45K20 (similaire au PIC18F4520 mais pouvant fonctionner à 64Mhz avec une alimentation de 3,3v

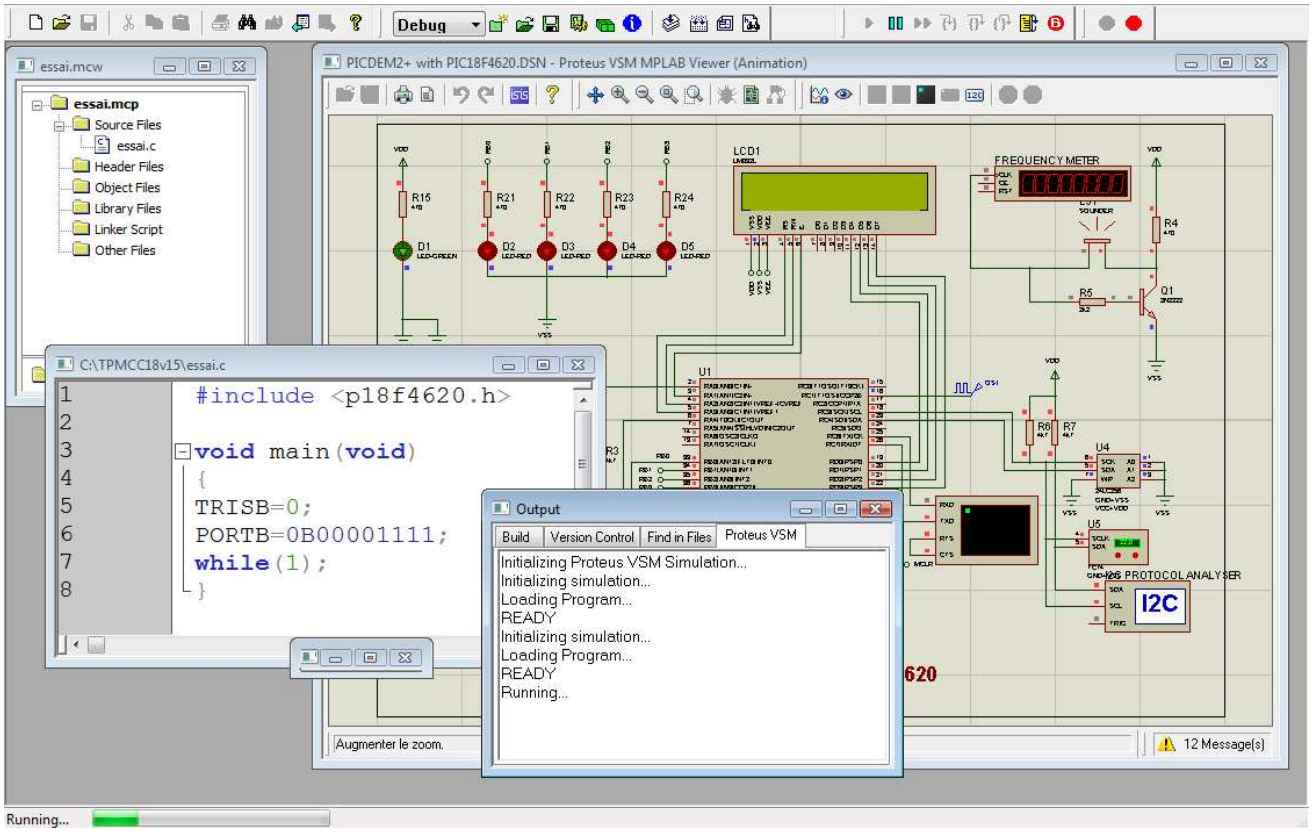
Les oscillateurs 10Mhz et 32Khz ne sont pas câblés sur la carte de démonstration. Il est donc obligatoire d'activer l'oscillateur interne du PIC18F45K20 (par défaut FOSC=1Mhz, TCY = 4*1/TOSC=4us.)



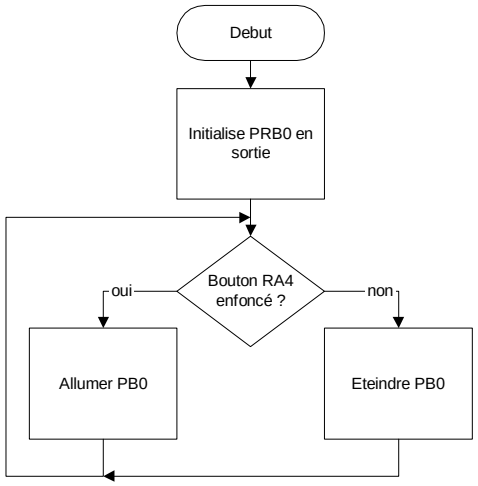
3.2.4. PROTEUS – VSM

Le logiciel de CAO PROTEUS-ISIS peut être utilisé comme debbuger et servir de maquette virtuelle
Dans MPLAB sélectionner le debugger PROTEUS-VSM

Une fenêtre ISIS s'ouvre dans MPLAB, dans cette fenêtre charger le schéma de la maquette virtuelle.
Cliquer sur le bouton vert pour activer la simulation, compiler le programme et l'exécuter sur la maquette virtuelle.



3.3. Gestion des ports parallèles



Fichier « bouton.c »

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

```
/* Bouton et LED LED sur PICKIT3+*/

#include <p18f45K20.h>

#pragma config FOSC = INTIO67, FCMEN = OFF, IESO = OFF
#pragma config PWRT = OFF, BOREN = OFF, BORV = 30
#pragma config WDTEN = OFF, WDTPS = 32768
#pragma config MCLRE = ON, LPT1OSC = OFF, PBADEN = OFF, CCP2MX = PORTC
#pragma config STVREN = ON, LVP = OFF, XINST = OFF

void main(void)
{

TRISB=0xFF;       // PORTB en entrée pour bouton SW1
TRISD = 0;        /* PORTD en sortie pour LED DS1*/
       while(1)         // une boucle infinie
      {
            if (PORTB & 0x01) PORTD=1;
            else PORTD=0;
      }
}
```

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**

Rappels sur le masquage :

- **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

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

Améliorer la lecture du programme :

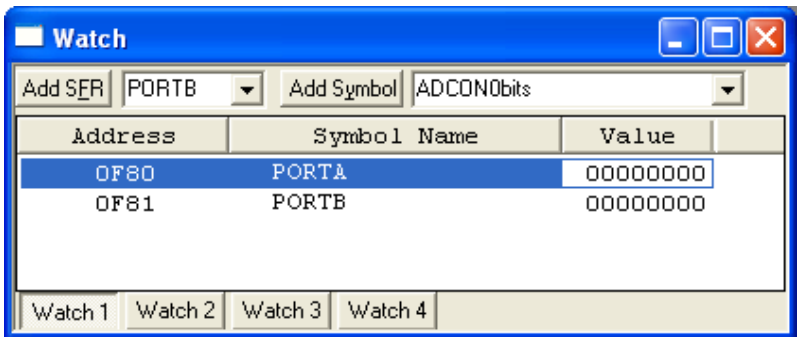
```
#define PORTBbits.RB0 sw1
#define PORTDbits.RD0 sd1
```

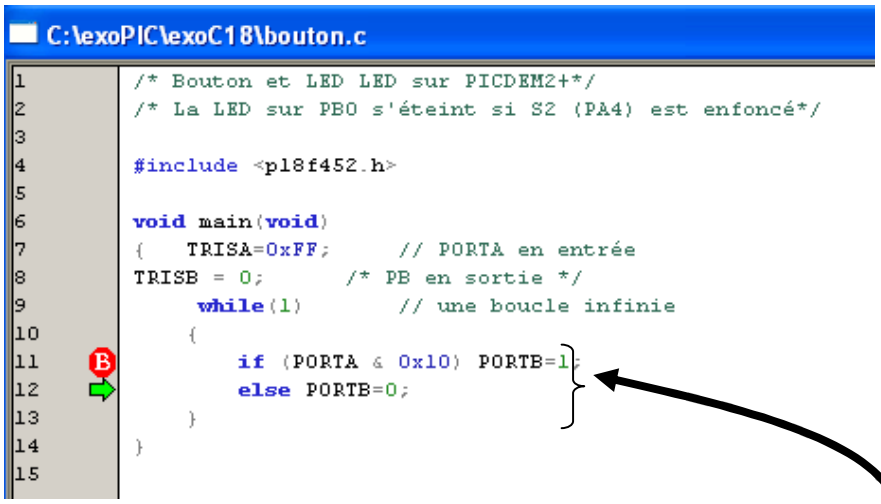
On peut alors écrire dans le programme principal:

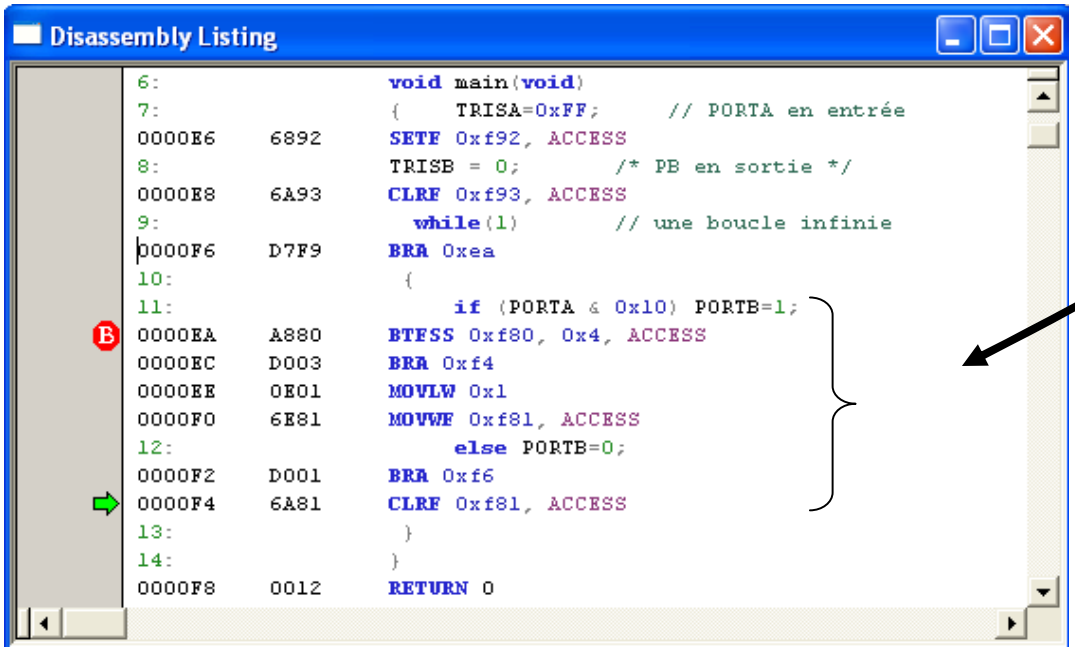
```
if (sw1) sd1=1 ;
Else sd1=0 ; // on ne modifie ici que le bit '0' du port D
```

3.4. 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,







.

3.5. Création d'une fonction

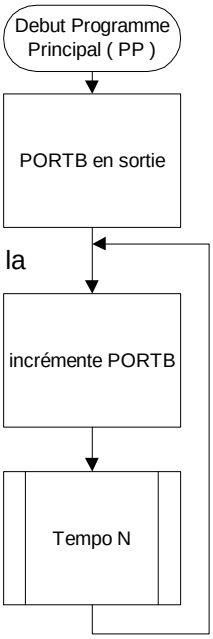
Recopier le programme led.c

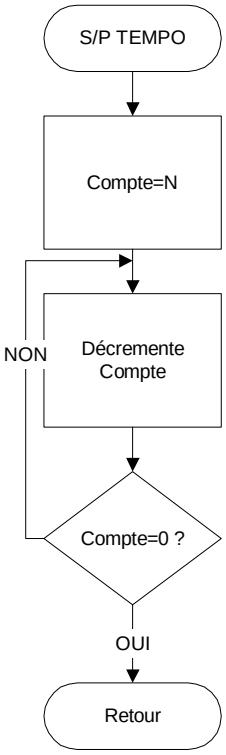
```
#include <p18f45K20.h>

#pragma config FOSC = INTIO67, FCMEN = OFF, IESO = OFF
#pragma config PWRT = OFF, BOREN = OFF, BORV = 30
#pragma config WDTEN = OFF, WDTPS = 32768
#pragma config MCLRE = ON, LPT1OSC = OFF, PBADEN = OFF, CCP2MX = PORTC
#pragma config STVREN = ON, LVP = OFF, XINST = OFF

#define led PORTDbits.RD0
#define duree 10000

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


void main(void)
{
      TRISD = 0x00;
      while(1) {
                        led=!led;
                        tempo(duree);

                  }
}
```

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

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.*



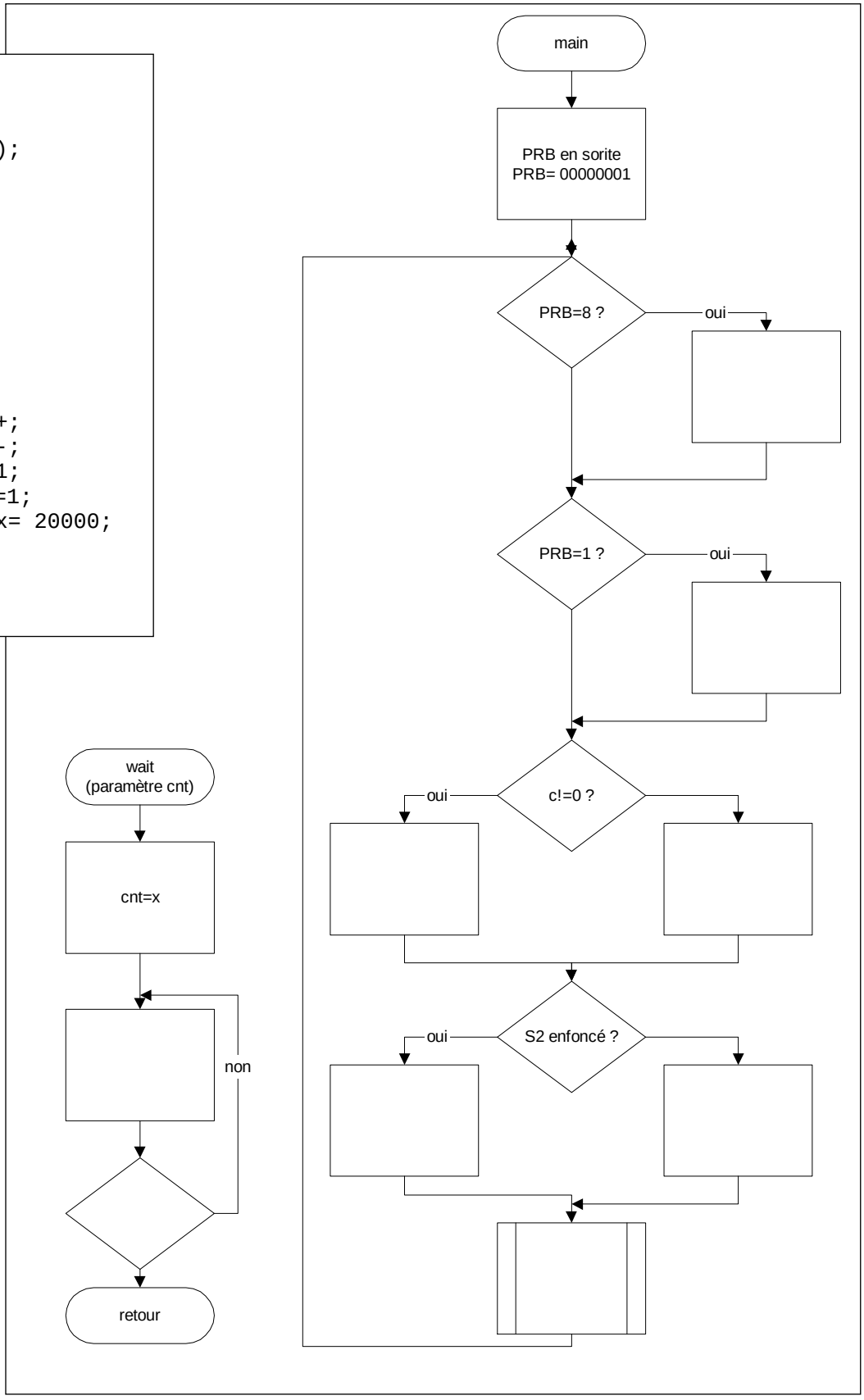


3.6. Analyse d'un programme écrit en C18 : décalages

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 !



4. 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)

4.1. Editeur de liens MPLINK

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

4.1.1. 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   NAME=idlocs   START=0x200000   END=0x200007   PROTECTED
CODEPAGE   NAME=config   START=0x300000   END=0x30000D   PROTECTED
CODEPAGE   NAME=devid    START=0x3FFFFE   END=0x3FFFFF   PROTECTED
CODEPAGE   NAME=eedata   START=0xF00000   END=0xF000FF   PROTECTED

ACCESSBANK  NAME=accessram  START=0x0      END=0x7F
DATABANK    NAME=gpr0       START=0x80     END=0xFF
DATABANK    NAME=gpr1       START=0x100    END=0x1FF
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.**

4.1.2. 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**

4.2. 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 :
makeonep 18f4620

4.3. 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.

4.4. math.h

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

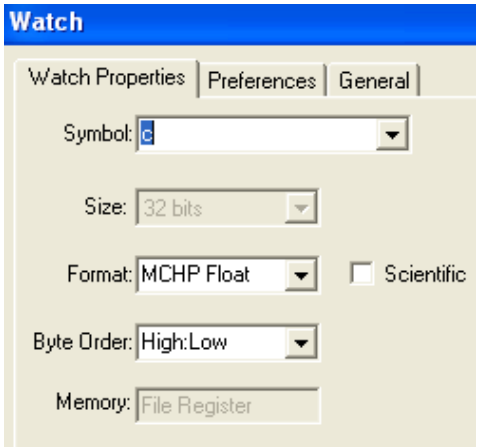
math.h

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

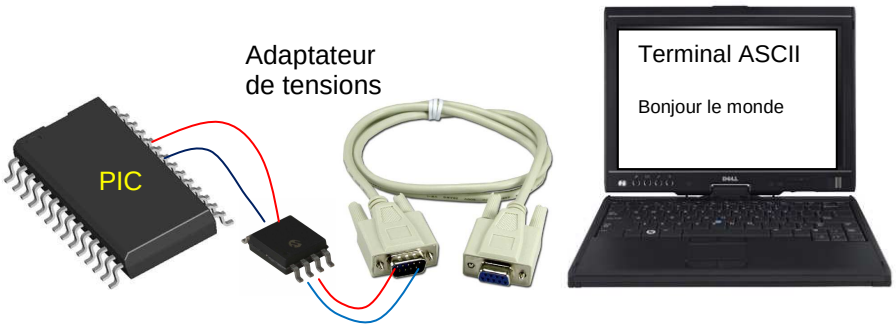


4.5. Bibliothèques : Les sorties de texte

Le formatage du texte repose sur les fonctions C ANSI « printf » (voir page25) de la bibliothèque stdio.h.
La bibliothèque libusart.h (voir page 44) 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)
En C18 la sortie par défaut utilisée par printf est l'USART. L'utilisateur peut définir sa propre sortie de caractères en utilisant fprintf
_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

```
// printf.c  demo pour printf C18
#include <p18fxxx.h>
#include <stdio.h>        // pour printf

void main(void)
{
       SPBRG = 25; /* configure la vitesse (BAUD) 9600 N 8 1*/
       TXSTA = 0x24;
       RCSTA = 0x90;       /* active l'USART*/
       printf "Bonjour le monde\n");  // vers USART
       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)

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

4.5.1. 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

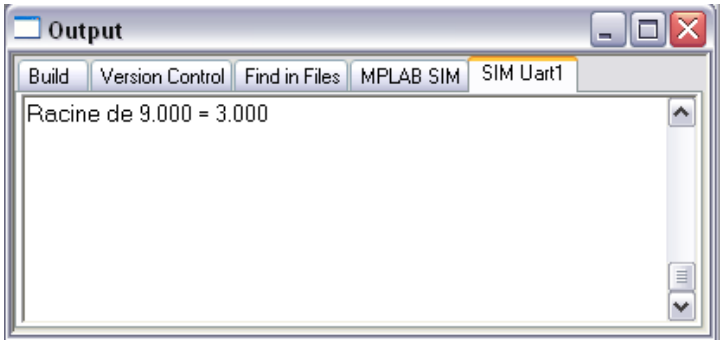
unsigned char ***ftoa*** (float x, unsigned char *str,char prec,char format);

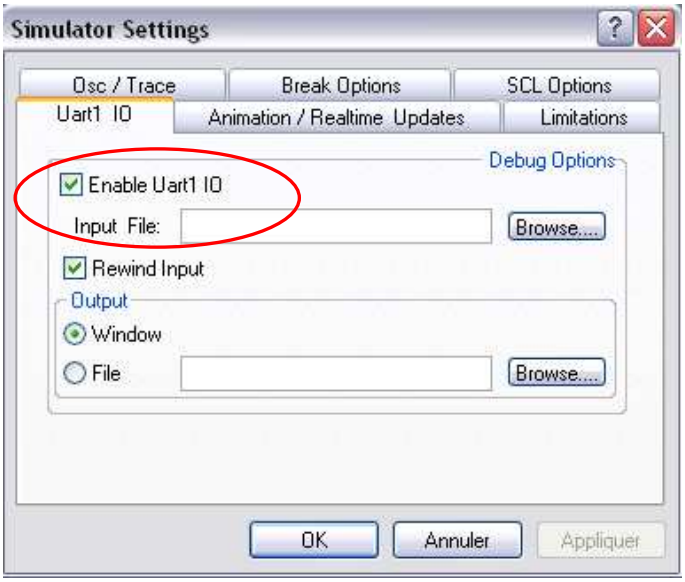
unsigned char chaine[10];
EX: ftoa(3.1415,chaine,2,'s')

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

4.5.2. Sortie de texte sur le simulateur de MPLAB

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 »



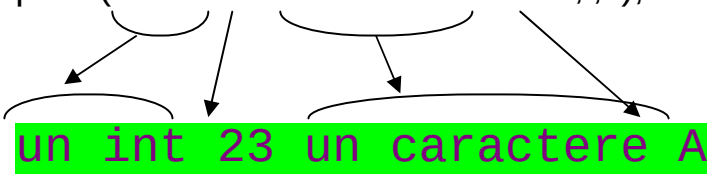


printf, fprintf, sprintf

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

Format :

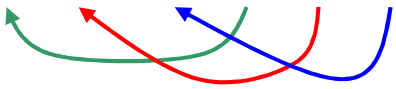
```
printf("un int %d un caractere %c",i,c);
```



```
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;
```

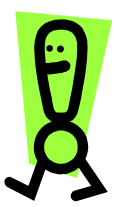
Script	Affichage
printf("Dec : %d %u",a,a);	Dec : -27 65509
printf("Hex: %#06X %x ",b,b);	Hex: 0X00B5 b5
printf("Bin: %16b",b);	Bin: 0000000010110101
printf("Bin: %#010B",b);	Bin: 0B10110101
printf("%c %c %d",'b',c,(int)c);	b A 65
printf("J habite %S",chrom);	J habite en ROM
printf("J habite %s",chram);	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
```

-

10.1.1. Exemple de sortie texte, le 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) // algorithme d'Héron Ier siècle ap. J.C.
{
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`

Annexe : la fonction exponentielle

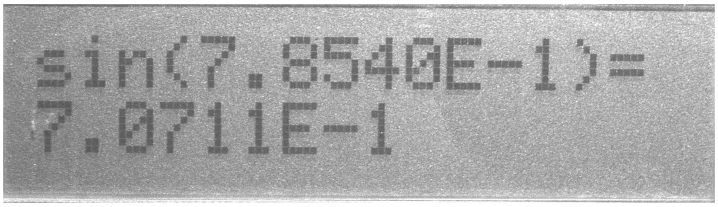
```
// fonction exponentielle sur nombres entiers
float exp(float f)
{
float s=1.0,u=1.0;
int n;
      for (n=1;abs(u)>0.001;n++)
      {
            u=u*f/n;
            s+=u;
      }
    return s;
}
```

10.1.2. Exemple sur math.h

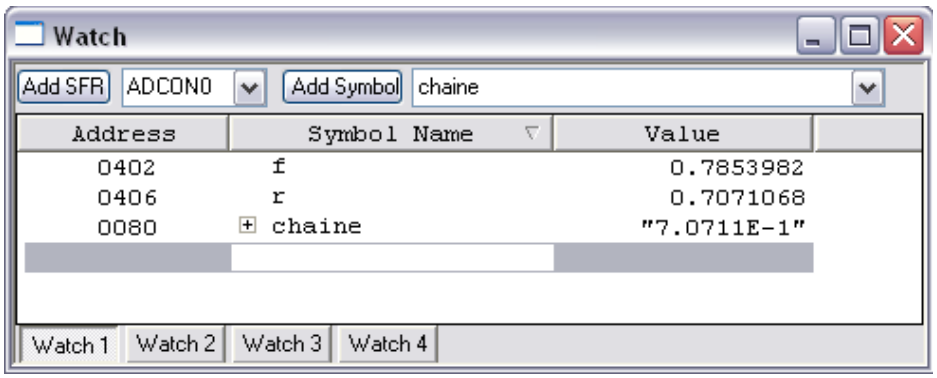
Ici test de la fonction sinus, fonction de la bibliothèque math.h. (Attention les angles doivent être donnés en radians)

Consulter les .h
dans c:\mcc18\h

```
#include <p18f4620.h>
#include <stdio.h>
#include <math.h>
#include "ftoa.c" // indispensable pour l'affichage des reels

char chaine[10];

void main(void)
{float f,r;
      SPBRG = 25;
      TXSTA = 0x24;
      RCSTA = 0x90;     /* active l'USART*/
            f=3.14/4.0; // PI/4
                  ftoa(f,chaine,4,'S');
                  printf("sin(%s)=",chaine);
                  r=sin(f);
                  ftoa(r,chaine,4,'S');
                  printf("%s  \n\r",chaine);

      while(1);
}
```



Visualisation des résultats sur l'afficheur LCD et dans une fenêtre "WATCH"



- Il est possible de faire de même avec log, exp,pow, sqrt

1. Gestion de la mémoire

10.2. 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

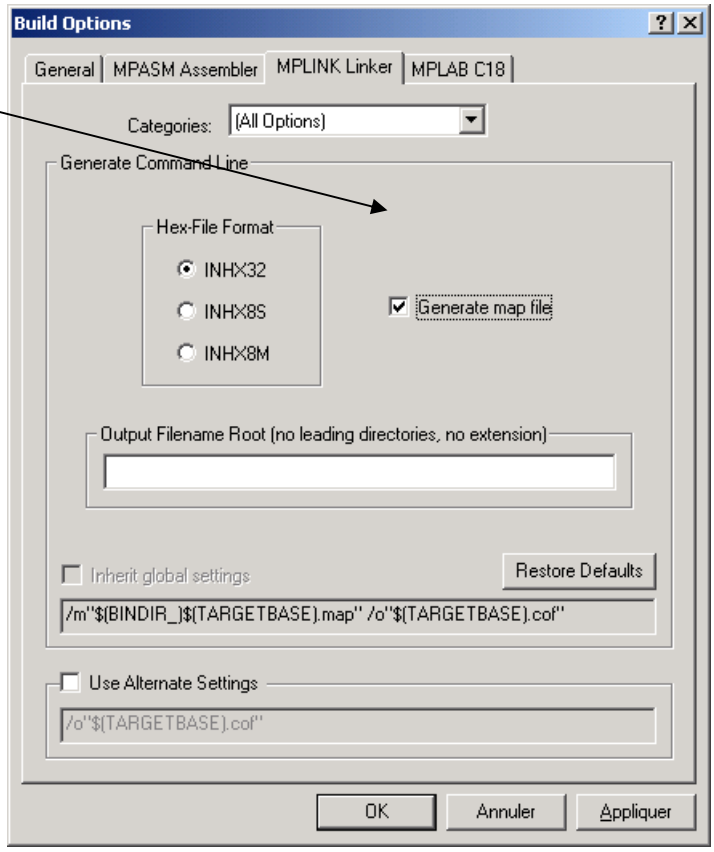
Taille des pointeurs :

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	

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
```

Sortie est un tableau de constantes rangées en ROM

```
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--);
}
```

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

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

```
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);
                  }
      }
}
```

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. PIC18F4620 – Configuration de l'horloge interne

Dans MPLAB :

Configuration Bits

Configuration Bits set in code.

Address	Value	Category	Setting
300001	07	Oscillator	EXT RC-Port on RA6
		Fail-Safe Clock Monitor Enable	
		Internal External Switch Over Mode	
300002	1F	Power Up Timer	
		Brown Out Detect	
		Brown Out Voltage	
300003	1F	Watchdog Timer	
		Watchdog Postscaler	
300005	83	CCP2 Mux	
		PortB A/D Enable	
		Low Power Timer1 Osc enable	
		Master Clear Enable	
300006	05	Stack Overflow Reset	
		Low Voltage Program	Enabled
		Extended CPU Enable	Disabled
300008	0F	Code Protect 00800-03FFF	Disabled
		Code Protect 04000-07FFF	Disabled
		Code Protect 08000-0BFFF	Disabled
		Code Protect 0C000-0FFFF	Disabled
300009	C0	Data EEPROM Code Protect	Disabled
		Code Protect Boot	Disabled
30000A	0F	Table Write Protect 00800-03FFF	Disabled
		Table Write Protect 04000-07FFF	Disabled
		Table Write Protect 08000-0BFFF	Disabled
		Table Write Protect 0C000-0FFFF	Disabled
30000B	E0	Data EEPROM Write Protect	Disabled
		Table Write Protect Boot	Disabled
		Config. Write Protect	Disabled
30000C	0F	Table Read Protect 00800-03FFF	Disabled

EXT RC-Port on RA6 — options :
11XX EXT RC-CLKOUT on RA6
101X EXT RC-CLKOUT on RA6
INT RC-CLKOUT on RA6,Port on RA7
INT RC-Port on RA6,Port on RA7
EXT RC-Port on RA6
HS-PLL enabled freq=4xFosc1
EC-Port on RA6
EC-CLKOUT on RA6
0011 EXT RC-CLKOUT on RA6
HS
XT
LP

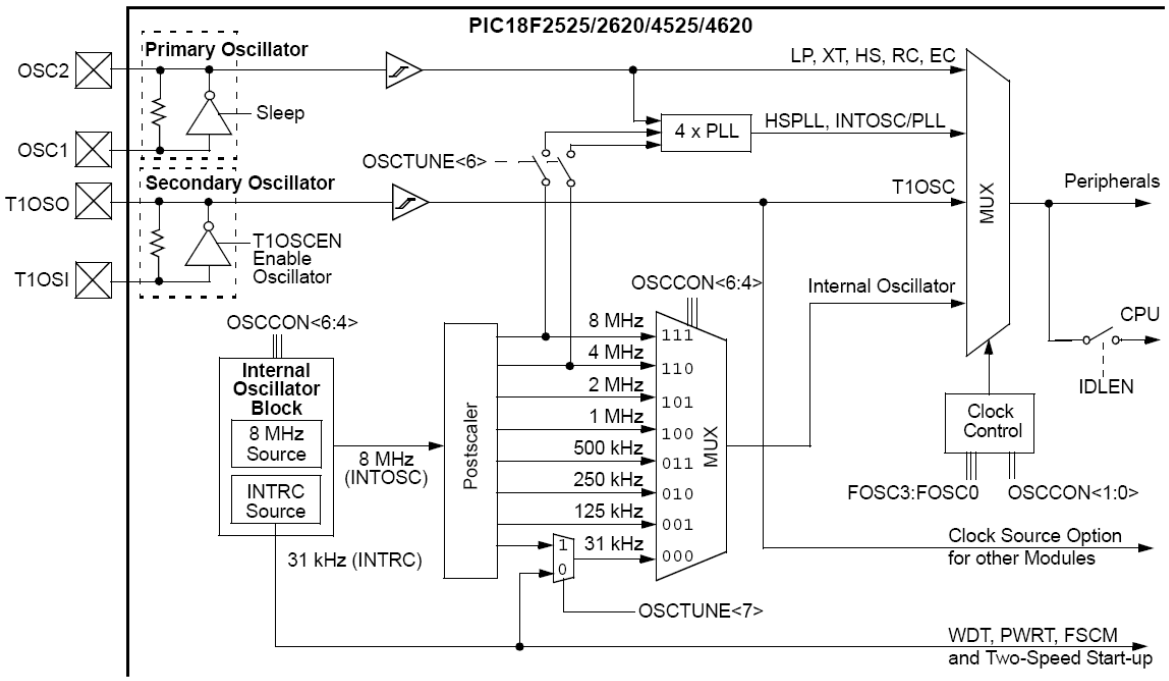
INT RC-CLOCKOUT on RA6, PORT on RA7 : l'horloge interne sort sur RA6, RA7 est un port //
INT RC-Port on RA6, Port on RA7 : RA6 et RA7 sont des ports //
Il est possible de ne pas utiliser MPLAB pour gérer les bits de configuration mais une directive du C18 (dans ce cas cocher « Configuration Bits set 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 (DOC PIC18F4620) , les bits de configuration activant 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 IRCFx de OSCCON permettent de choisir la fréquence de base (31KHz à 8MHz). (1MHz par défaut)
Les bits TUNx de OSCTUNE permettent d'ajuster la fréquence interne (en cas de variation de température)
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 SCSx de OSCCON permettre de choisir la source de l'horloge des périphériques et du CPU du PIC. (Interne par défaut)
ATTENTION, pour activer l'horloge interne AVEC PLL, il faut sélectionner INTOSC/PLL en positionnant les libts SCS1 et SCS0 à 0 !!!!!
Les bits OSTS et IOFC de OSCCON permettent de connaître l'état de l'horloge (active, stable …)
Exemple de configuration de l'horloge interne d'un PIC18Fxx20 8Mhz avec PLLx4

```
OSCCONbits.IRCF2=1;
OSCCONbits.IRCF1=1;
OSCCONbits.IRCF0=1;
OSCTUNEbits.PLLEN=1;
OSCCONbits.SCS1=0;
OSCCONbits.SCS0=0;
```

7. 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)

7.1. 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.
.

7.2. Exemple de programmes fonctionnant en IT

7.2.1. Avec le PORTB : Programme demo_it_rb0.c

```
#include <p18f45K20.h>

#pragma config FOSC = INTIO67, FCMEN = OFF, IESO = OFF
#pragma config PWRT = OFF, BOREN = OFF, BORV = 30
#pragma config WDTEN = OFF, WDTPS = 32768
#pragma config MCLRE = ON, LPT1OSC = OFF, PBADEN = OFF, CCP2MX = PORTC
#pragma config STVREN = ON, LVP = OFF, XINST = OFF

#define leds PORTD

// sous programme d'interruption
#pragma interrupt it_sur_rb0       // choix du niveau d'interruption
void it_sur_rb0(void)
{
      if (INTCONbits.INT0IF)  // vérifie que l'IT est INT0, origine PB0=0
            {
                  leds++;     // incremente PORTD
                  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)
{     TRISD = 0x00;           // PORTD en sortie pour visu sur LEDs
      TRISBbits.TRISB0=1;     // PRB0 en entrée
      INTCON2bits.INTEDG0=0;  // IT sur front descendant
      INTCONbits.INT0IE=1;    // INT0 activée
      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 sur SW1 (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 le PORTD. Le drapeau de l'interruption est effacé avant le retour vers la boucle while(1) du programme principal.

7.2.2. Avec le TIMER 0 : Programme flashit.c

Ce programme fait clignoter la LED sur PORTD0 (ex PICKIT3) 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 ici 250KHz, prédiviseur par 2
La période des débordements est donc 4uS * 2 * 65536 = 524.288 mS

```
#include <p18f45K20.h>

#pragma config FOSC = INTIO67, FCMEN = OFF, IESO = OFF
#pragma config PWRT = OFF, BOREN = OFF, BORV = 30
#pragma config WDTEN = OFF, WDTPS = 32768
#pragma config MCLRE = ON, LPT1OSC = OFF, PBADEN = OFF, CCP2MX = PORTC
#pragma config STVREN = ON, LVP = OFF, XINST = OFF

void traiteIT(void);
```

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

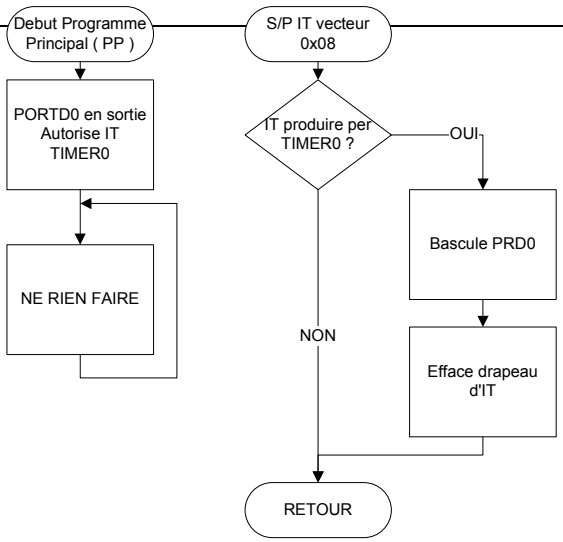
```
#pragma code it=0x08
void saut_sur_spIT(void)
{
_asm
      goto traiteIT
_endasm
}
#pragma code
```

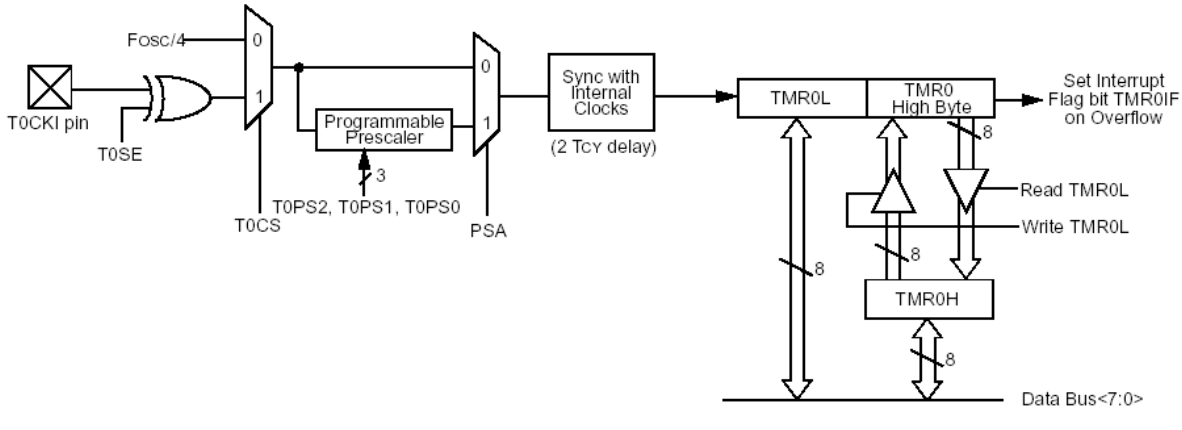
Saut sur le S/P de traitement de l'interruption

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
                  PORTDbits.RD0 = !PORTDbits.RD0;     //bascule LED sur RD0
                  }
}
void main()
{
      PORTDbits.RD0 = 0;
      TRISDbits.TRISD0 = 0;
      T0CON = 0x80;
      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





8. Périphériques et C18

8.1. TIMER1 Production de temps

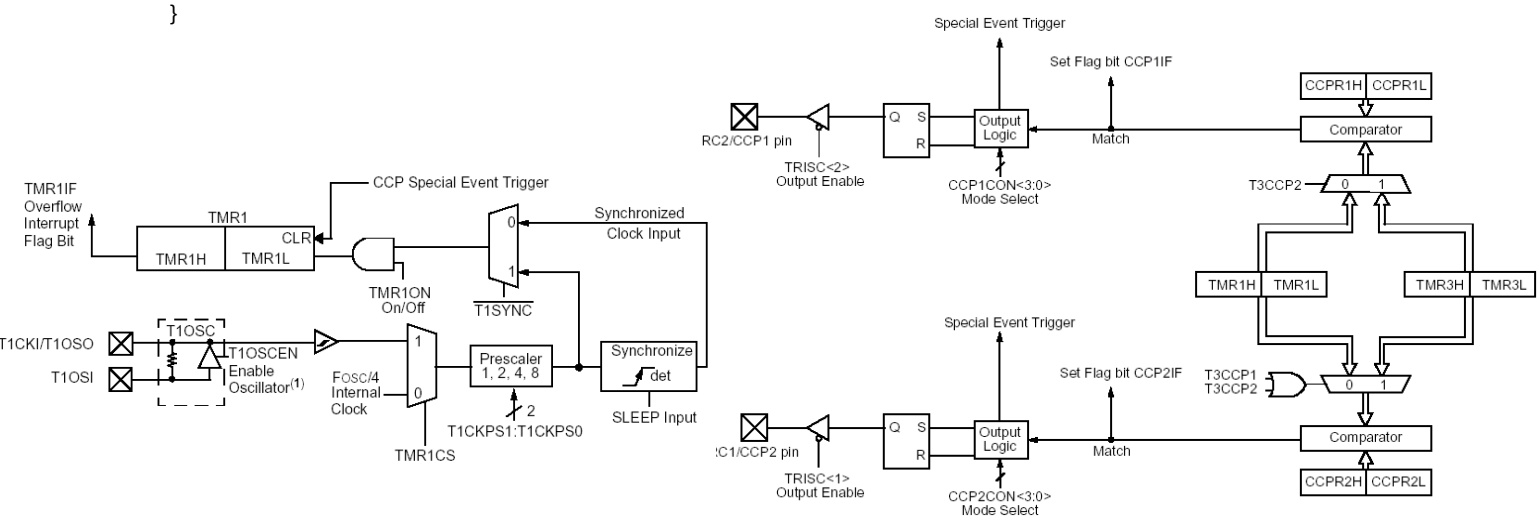
Programme itcomp.c

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 .

```
#include <p18f45K20.h>

#pragma config FOSC = INTIO67, FCMEN = OFF, IESO = OFF
#pragma config PWRT = OFF, BOREN = OFF, BORV = 30
#pragma config WDTEN = OFF, WDTPS = 32768
#pragma config MCLRE = ON, LPT1OSC = OFF, PBADEN = OFF, CCP2MX = PORTC
#pragma config STVREN = ON, LVP = OFF, XINST = OFF

#define led PORTDbits.RD0// 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) {
                                        led=!led;  //bascule PD0
                                        tictac=0;
                                                }
                        PIR1bits.CCP1IF=0;                      //efface le drapeau d'IT
                        }
}

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

void main(void)
{       OSCCON=0b01010010;      // horloge interne 4Mhz (TCY=1us)
        TRISDbits.TRISD0=0;     // PRD0 en sortie
        // 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
}
```

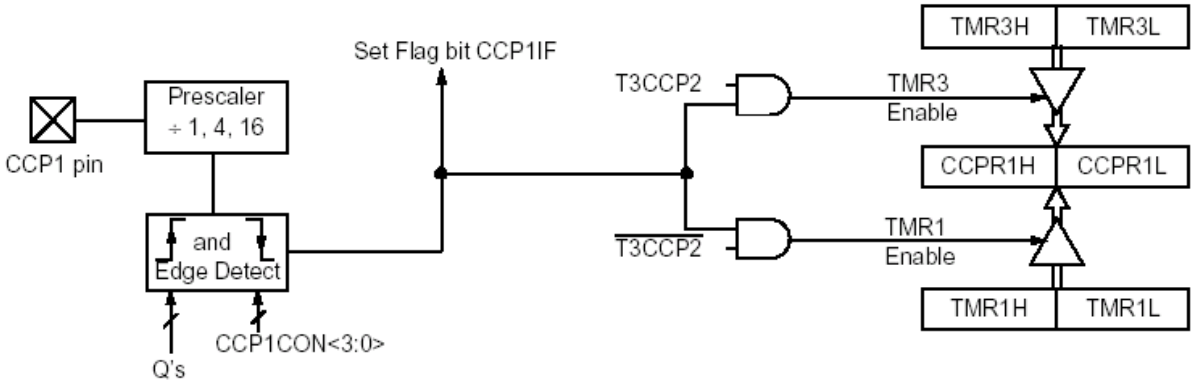
Il y a une IT toutes les 125mS, il faut attendre 4 IT avant de basculer PB0. le compteur d'IT tictac est local, il doit également être statique pour ne pas être perdu à la

configure le mode comparaison sur le TIMER1 avec IT sur CCP1 toutes les 15625 périodes de 8us soit 125ms



8.2. TIMER1 Mesure de temps

```
// mesure de période sur CCP1 (RC2) (TIMER1 et fonction capture)

#include <p18f4620.h>
#include <stdio.h>

unsigned int duree;             // 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)
{
static unsigned int ancien;
        if(PIR1bits.CCP1IF)                             // l'IT provient d'une capture
                        {
                        duree=CCPR1-ancien;
                        maj=1;
                        ancien=CCPR1;
                        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)
{// init USART
        SPBRGH= 0x00;                   // SPBRG est compose de SPBRGH et SPBRG
        SPBRG = 25;             /* configure la vitesse (BAUD) 9600 N 8 1 (QUARTZ 4MHz)*/
        TXSTA = 0b00100000;
        RCSTA = 0b10010000;             /* active l'USART*/
        BAUDCONbits.BRG16=0;    // SPBRGH = 0 pour 9600
        TXSTAbits.BRGH=1;               // High Speed
// configure PORTC CCP1
        TRISCbits.TRISC2=1;     // RC2/CCP1 en entree
// configure le TIMER1
        T1CON=0b00110001;               // prediv 1/8 osc interne Fosc/4=1MHz (PTRM1=1/125KHz=8us)
// configure le mode capture sur le TIMER1 avec IT sur CCP1
        T3CONbits.T3CCP2=0;             // mode comparaison entre TMR1 et CCPR1
        CCP1CON=0b00000101;             // 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

        while(1)
                {
                if (maj)        {
                                printf("duree= %u \n\r" ,duree);
                                maj=0;
                                }
                }
}
```



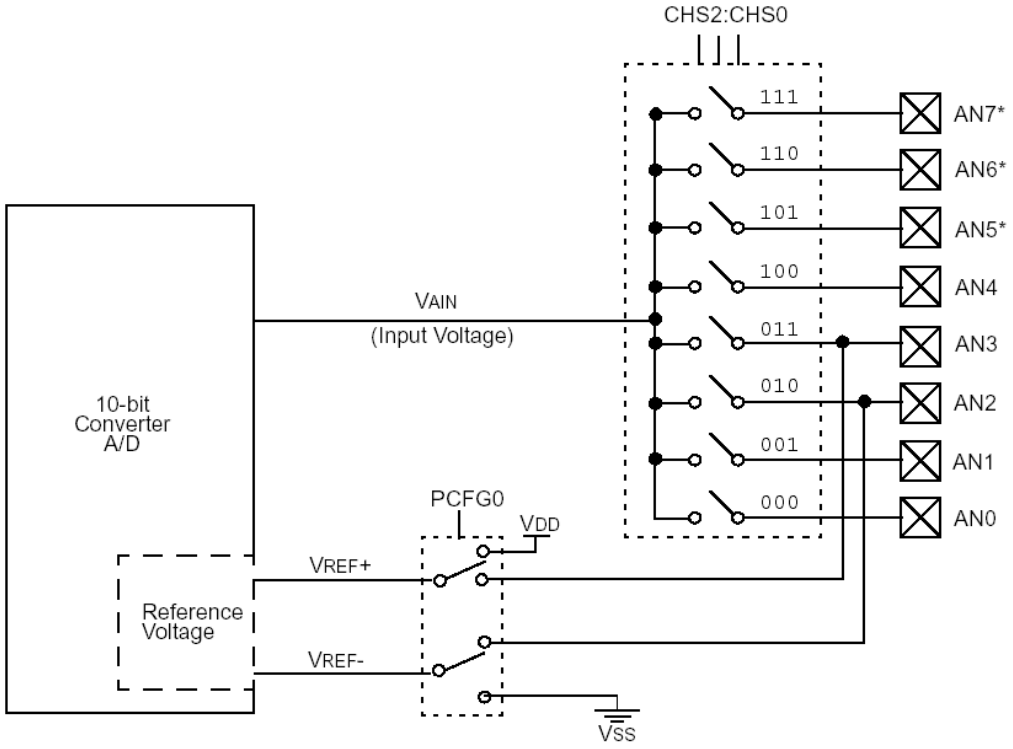
8.3. Conversion analogique/Numérique

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.

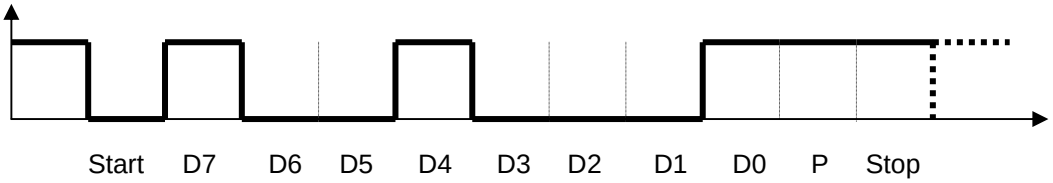
```
#include "xlcd.h"
#include <stdio.h>
#include <tempo_lcd_pd2.c> // tempo pour xlcd.h

#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)
            }
}
```

Envoie vers l'afficheur LCD une chaîne depuis la RAM



8.4. Communications séries asynchrones (P18Fxx20)

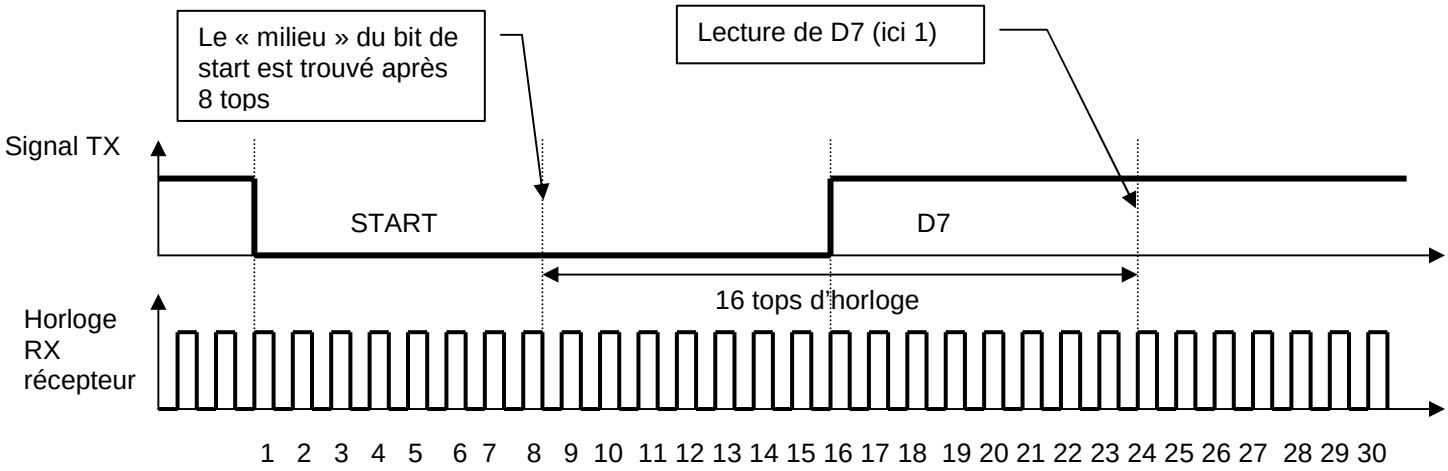
Les communications séries asynchrones suivent le format NZR (No Return to Zero). Ils communiquent avec le microprocesseur par l'intermédiaire du bus de données et en série avec l'extérieur par une liaison série asynchrone (sans horloge). Ils peuvent par ailleurs piloter un modem.
Exemple de trame : asynchrone 1start, 8 bits, parité paire, 1 stop : nombre 10010001



Parité paire : le bit de parité est positionné pour que l'ensemble des bits de donnée et le bit de parité représente un nombre de bits à 1 pair
Parité impaire : le bit de parité est positionné pour que l'ensemble des bits de donnée et le bit de parité représente un nombre de bits à 1 impair

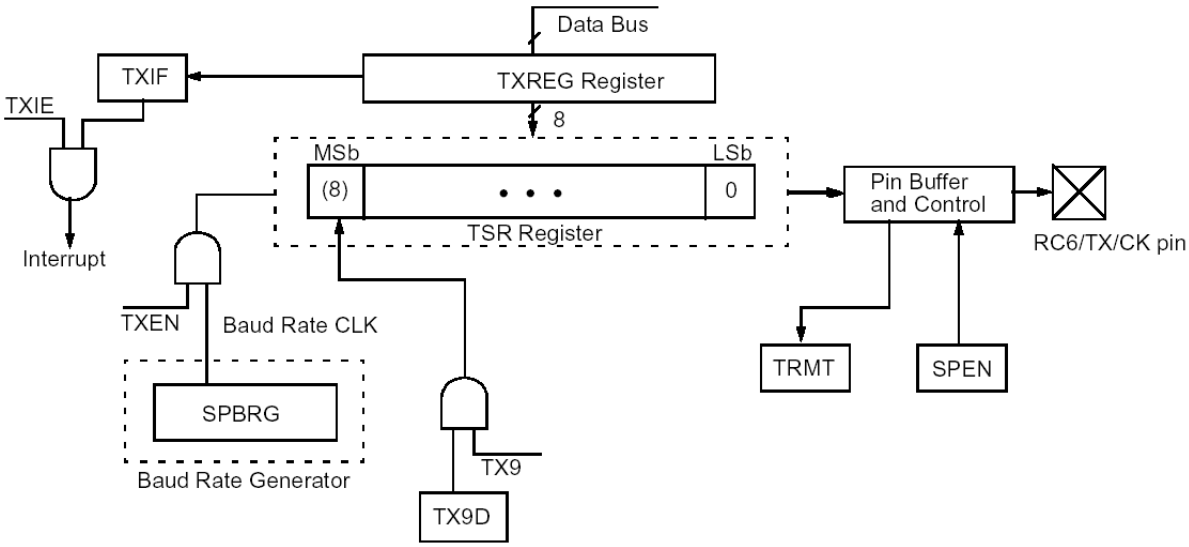
Dans ce type de transmission l'horloge de transmission est comprise dans le signal, le bit de start est utilisé par le récepteur pour se synchroniser avec l'émetteur. Cependant les deux horloges de transmission et de réception sont au départ très proche

L'horloge de réception possède une fréquence multiple de celle de transmission (en général x16 ou x64)
Dans le cas d'une division par 16 :
Lors de la réception du front descendant du bit de start, l'USART attend 8 tops d'horloge, le circuit reçoit alors théoriquement le milieu du bit de start, l'USART attend ensuite 16 tops d'horloge, le circuit reçoit alors le milieu de D7 et lit ce bit, l'USART attend ensuite 16 tops etc. L'horloge du récepteur est donc resynchronisée lors de la réception de chaque caractère.



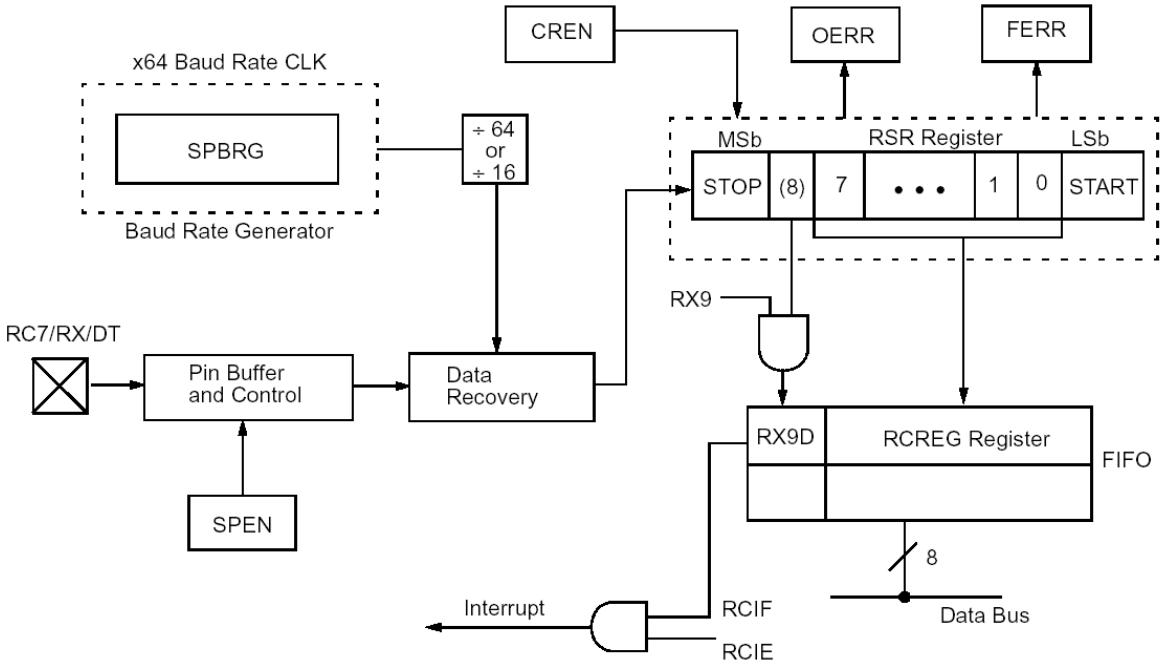
L'horloge RX (réception) doit donc toujours être supérieure à celle de TX (transmission). En réalité les deux horloges sont identiques et TX est divisé dans l'USART pour produire la vitesse de transmission souhaité.

USART : Transmission



TXREG : registre de transmission (tampon)
SPBRG : définit la vitesse de transmission (BAUD)
TXEN : valide l'horloge
TX9 : valide le 9ième bit
TX9D :9ième bit (donnée, adresse ou parité)
TXIE : autorise l'interruption
TXIF : drapeau d'interruption, indique que TXREG est vide
SPEN : configure TX/RX pin pour USART
TMRT : indique si TSR est vide

USART : Réception



CREN : active le récepteur asynchrone
SPBRG : définit la vitesse de transmission (BAUD)
SPEN : configure TX/RX pin pour USART
RCIE : autorise l'interruption en réception
RCIF : drapeau d'interruption de réception d'une donnée
RX9 : valide la prise en compte de D8 (adresse, donnée ou parité, traitement par logiciel)
OERR, FERR : indicateurs d'erreurs de réception

Registres généraux

TXSTA : Registre d'état et de contrôle des émissions

7	6	5	4	3	2	1	0
CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D

CSRC: Clock Source Select bit
Non utilisé en mode asynchrone
1 = mode maitre (horloge générée en interne depuis BRG)
0 = mode esclave (horloge externe)
TX9:
1 = transmission sur 9 bits
0 = transmission sur 8 bits
TXEN:
1 = transmissions activées
0 = transmissions désactivées
SYNC: USART Mode Select bit
1 = mode synchrone
0 = mode asynchrone
SENDB : Send Break Character bit
Mode asynchrone :
1 = Emet le caractère Break lors de la prochaine transmission (RAZ automatique)
0 = Fin de transmission du caractère Break
Inutilisé en mode synchrone
BRGH: High Baud Rate Select bit
Mode asynchrone :
1 = grande vitesse
0 = petite vitesse
Inutilisé en mode synchrone
TRMT: Transmit Shift Register Status bit
1 = registre de transmission TSR vide
0 = registre de transmission TSR plein
TX9D: 9th bit of Transmit Data
Peut être une adresse, une donnée ou un bit de parité

RCSTA: RECEIVE Registre d'état et de contrôle des réceptions

7	6	5	4	3	2	1	0
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D

SPEN: Serial Port Enable bit
1 = Active le port série (configure RX/DT et TX/CK comme des broches de port série)
0 = Désactive le port série
RX9: 9-bit Receive Enable bit
1 = Réception sur 9 bits, 0 = Réception sur 9 bits
SREN: Single Receive Enable bit
Inutilisé en modes asynchrones et en mode synchrone esclave
En mde synchrone et maitre
1 = Autorise une réception unique (effacé après la réception)
0 = interdit la réception
CREN: Continuous Receive Enable bit
En mode asnchrone
1 = Active le récepteur
0 = Désactive le récepteur
En mode synchrone
1 = Active la réception (CREN est prioritaire sur SREN)
0 = Désactive la réception
ADDEN: Address Detect Enable bit
En mode asynchrone sur 9bits (RX9 = 1):
1 = Active la détection d'adresse, autorise l'interruption et ne charge pas la donnée dans le buffer de réception quand RSR<8> =1
0 = Désactive la détection d'adresse, tous les bits sont envoyés dans le buffer de réception et le 9ième bit peut être utilisé comme bit de parité
FERR: Framing Error bit (erreur de trame, généralement le bit de start ou stop n'a pas été détecté correctement)
1 = Framing error (mis à jour par une lecture de de RCREG et la réception du prochain octet)
0 = No framing error
OERR: Overrun Error bit - Indique qu'un caractère a été perdu
1 = Overrun error (effacé en effacant CREN), 0 = No overrun error
RX9D: 9th bit of Received Data
Peut être un bit d'adresse ou de donnée ou de parité et doit être géré par logiciel

Exemple en C18 :
Emetteur activé, transmission sur 8bits mode asynchrone, pas de Break.
Active le port série, récepteur activé sur 8 bits,

```
TXSTA = 0b00100000;
RCSTA = 0b10010000;
PIE1bits.TXIE=0;    // IT en emission désactivée
PIE1bits.RCIE=1;    // IT en reception activée (si nécessaire)
PIR1bits.TXIF=0;    // efface drapeau transmission
PIR1bits.RCIF=0;    // efface drapeau reception
```

Générateur de BAUD.

Le générateur de BAUD (BRG) repose sur un comptage de l'horloge Fosc (à ne pas confondre avec Fcycle=Fosc/4)
BRG peut être un compteur 8 bits ou 16 bits

```
BAUDCONbits.BRG16=1 ; // compteur 16 bits
```

Le bit BRGH (TXSTA) permet d'activer ou non le pre-divideur sur BRG

```
TXSTAbits.BRGH=0; // BRG lent
```

La vitesse de transmission dépend de l'oscillateur de BRG16 de BRGH et de la valeur dans SPBRGH:SPBRG. Un choix judicieux de BRGH et BRG16 permettra de réduire l'erreur sur la vitesse de transmission en fonction de la fréquence de l'oscillateur et la vitesse en BAUD souhaitée

BRG16	BRGH	Mode BRG/EUSART	Formule de calcul de la vitesse en Baud (n = ([SPBRGH:SPBRG])
0	0	8-bit/Asynchronous	Fosc/[64 (n + 1)]
0	1	8-bit/Asynchronous	Fosc/[16 (n + 1)]
1	0	16- bit/Asynchronous	
1	1	16-bit/Asynchronous	Fosc/[4 (n + 1)]

Exemple : Pour un PIC avec FOSC=32Mhz et une vitesse de transmission souhaitée de 9600 Bauds, mode 8 bits (BRG16=0) , avec prediviseur (BRGH=0)

Vitesse recherchée en BAUD= FOSC/(64 ([SPBRGH:SPBRG] + 1))
On recherche X= SPBRGH:SPBRG:
X = ((FOSC/Desired Baud Rate)/64) – 1= ((32000000/9600)/64) – 1= [51.08] = 51 (arrondi)
La vitesse réelle en BAUD sera = 32000000/(64 (51 + 1)) = 9615.38
L'erreur est donc : (BAUDcalculé – BAUDdésiré)/ BAUDdésiré
= (9615.38 – 9600)/9600 = 0.16%

La configuration du générateur de BAUD (BRG) s'écrira :

```
BAUDCONbits.BRG16=0;
TXSTAbits.BRGH=0;
SPBRGH= 0x00;     // ligne inutile ici puisque BRG est sur 8 bits
SPBRG = 51;
```

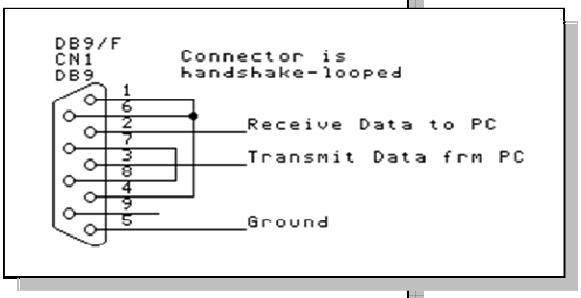
BAUDCON: BAUD RATE CONTROL REGISTER

7	6	5	4	3	2	1	0
ABDOVF	RCIDL	RXDTP	TXCKP	BRG16	—	WUE	ABDEN

ABDOVF: Auto-Baud Acquisition Rollover Status bit
1 = BRG a été mis à jour durant le mode Auto-Baud Rate Detect (RAZ par logiciel)
0 = BRG n'a pas été mis à jour
RCIDL: Receive Operation Idle Status bit
1 = Une opération de réception est en attente
0 = Une opération de réception est active
RXDTP: Received Data Polarity Select bit (Asyncnronous mode only)
Asynchronous mode:
1 = la donnée en RX est inversée (0-1)
0 = la donnée en RX n'est pas inversée (1-0)
TXCKP: Clock and Data Polarity Select bit
Asynchronous mode:
1 = L'état de repos pour une transmission est 0
0 = L'état de repos pour une transmission est 1
Synchronous mode:
1 = L'état de repos pour l'horloge est 1
0 = L'état de repos pour l'horloge est 0
BRG16: 16-Bit Baud Rate Register Enable bit
1 = 16-bit Baud Rate Generator – SPBRGH et SPBRG forment SPBRG sur 16 BITS
0 = 8-bit Baud Rate Generator – SPBRG seulement (Compatible avec les anciens PIC), SPBRGH est ignoré
bit 2 **Unimplemented:** Read as '0'
WUE: Wake-up Enable bit
Asynchronous mode:
1 = EUSART echantillonne la broche RX en continu, une interruption est générée sur front descendnat, ce bit est effacé automatiquement lors du prochain front montant
0 = RX n'est pas surveillé
Synchronous mode:
Unused in this mode.
ABDEN: Auto-Baud Detect Enable bit
Asynchronous mode:
1 = Active la mesure automatique de la vitesse de réception au prochain caractère. (Nécessite la réception du caractère 55h) . Effacé automatiquement après la mesure
0 = La mesure de la vitesse de reception est desactivé.

Exemple : programme echo (PIC↔ PC) : Fosc=32MHz , comm : 9600,n,8,1

```
#include <p18f4620.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)
{
        TXSTA = 0b00100000;
        RCSTA = 0b10010000;
        PIE1bits.TXIE=0;      // IT en emission désactivée
        PIE1bits.RCIE=0;      // IT en reception déactivée
        BAUDCONbits.BRG16=0;
        TXSTAbits.BRGH=0;
        SPBRG = 51;    // SPBRG=25 avec FOSC=4Mhz
        putchaine(mess);      // intro
        while(1)              // echo +1 ex: si 'a' est transmis 'b' est envoyé en echo
            { if (data_recue()) putch(RCREG+1); }
}
```

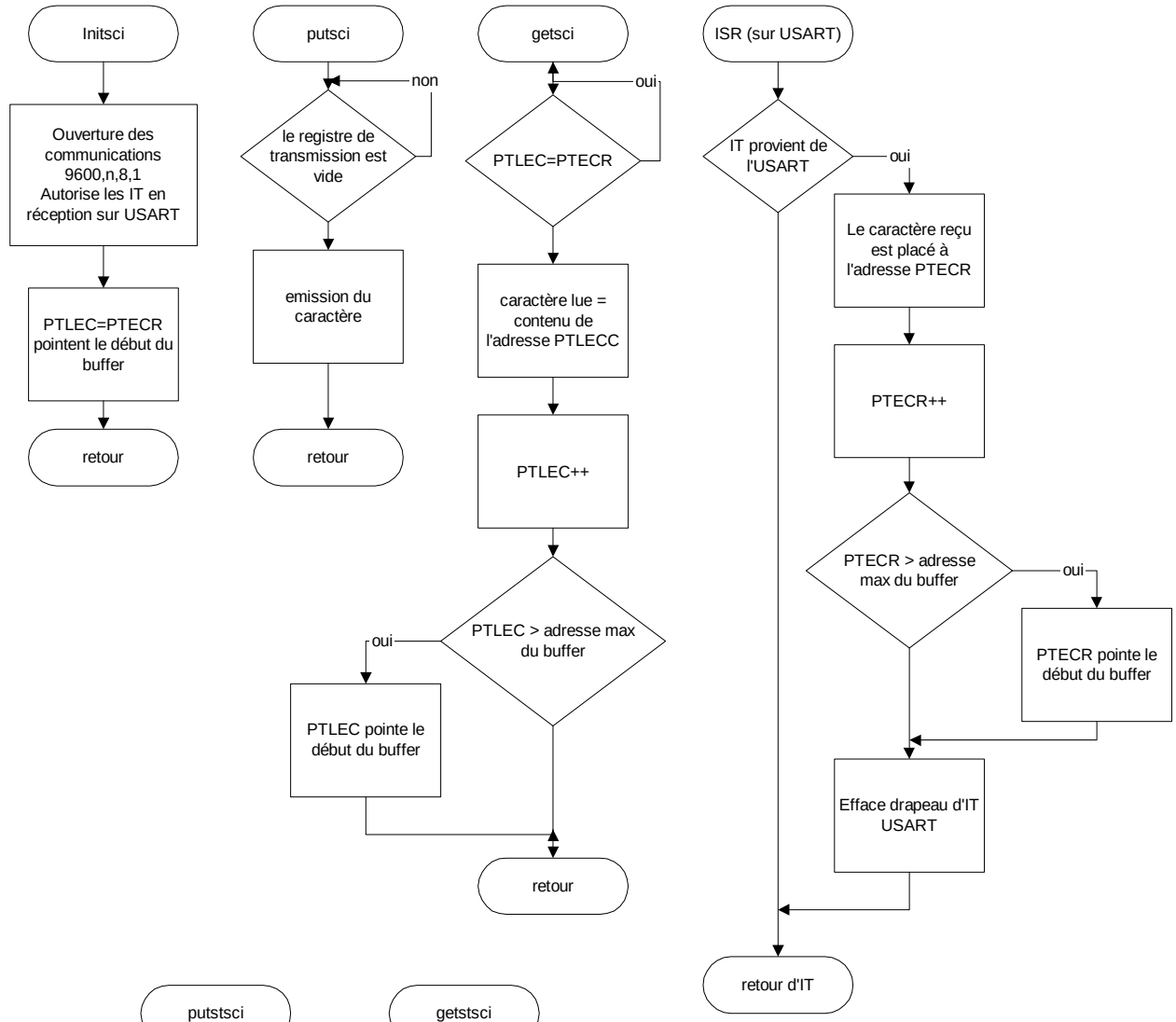


Ce programme **ne traite pas la perte de données en réception par écrasement**

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**



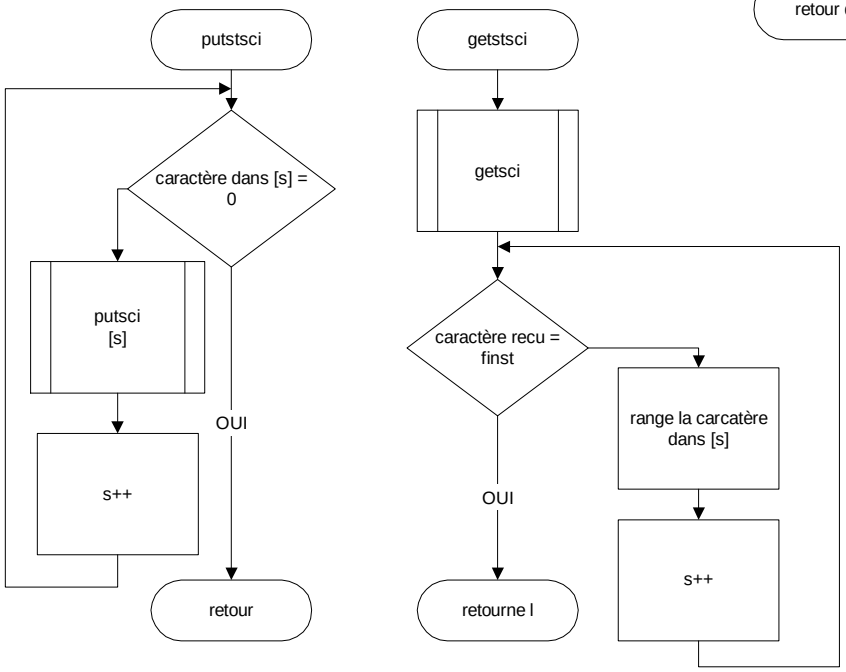
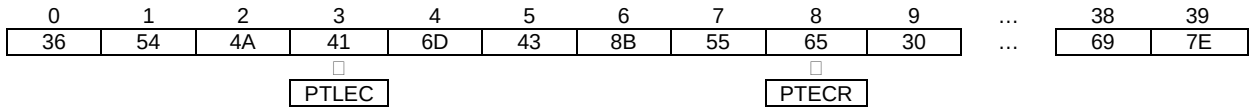


Tableau de réception :
Les caractères reçus sont rangés dans un tableau (buffer) à l'adresse d'un pointeur PTECR qui est ensuite incrémenté. getsci retourne le caractère ponté par PTRECR puis PTRECR est incrémenté.
Lorsque tous les caractères reçus ont été lus, PTERC=PTLEC. Si l'un des pointeurs dépasse 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.



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

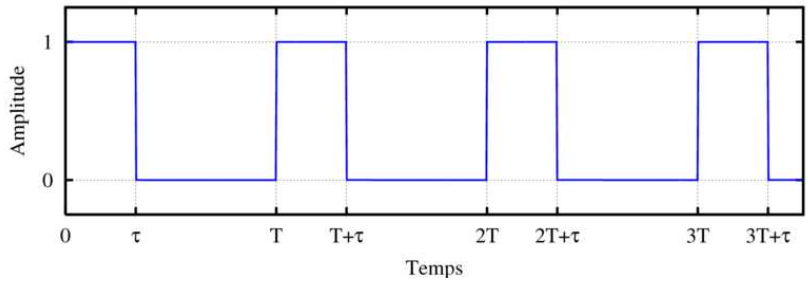
```
// gestion SCI en IT
#include <p18f4620.h>
#include "libusart.c"

void main(void)
{
unsigned char chaine[]="Bonjour\n\r";
      initsci(); // !!! cette fonction doit etre configuree en fonction de Fosc
      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
      }
}
```

8.5. PWM

8.5.1. Principes

Pulse Width Modulation : modulation de largeur d'impulsion
Le principe est de générer un signal logique ayant une fréquence mais dont le rapport cyclique est contrôlé numériquement. La valeur moyenne du signal de sortie est proportionnelle au rapport cyclique. Un filtre passe bas permettra d'obtenir cette valeur moyenne sous forme analogique.
Ce procédé est utilisé dans la commande des moteurs et en amplification audio de classe D. Les microcontrôleurs l'utilisent également pour la synthèse vocale sans convertisseur numérique analogique.
L'interface de puissance est composé de transistor MOS-FET, ces derniers fonctionnant en régime triode/bloqué ont un excellent rendement en commutation.

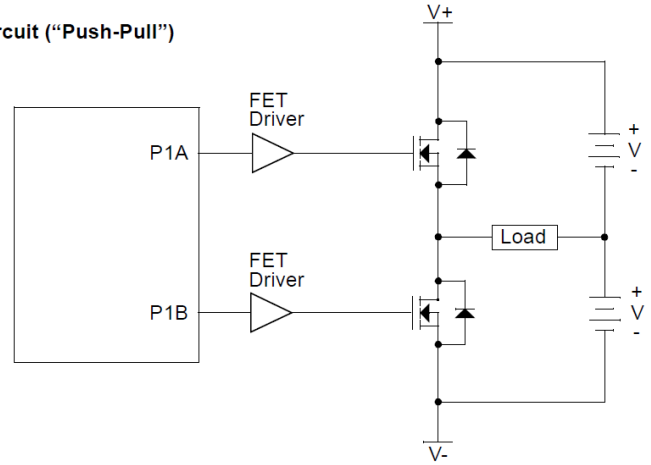
Rapport cyclique :



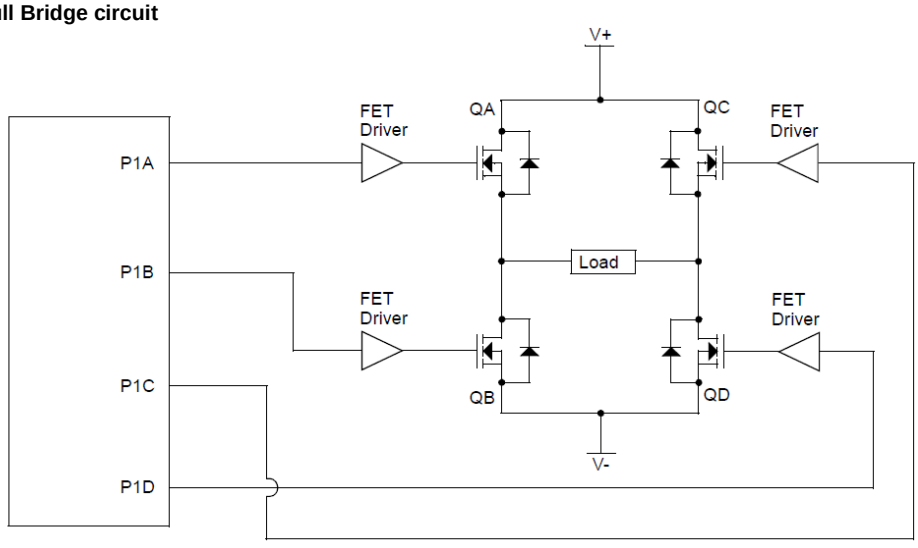
n=th/T n : rapport cyclique, th : durée de l'état haut, T : période
La valeur moyenne du signal étant : Vmoy=Vmax . n

Les PIC sont équipés d'une fonction PWM.
Sur les PIC18Fxx20, il existe deux sorties PWM d'usage général, les PIC18FxxK20 dispose d'un mode étendu destiné à la commande d'un demi-pont ou d'un pont entier de transistors MOS, la charge étant généralement un moteur à courant continu.

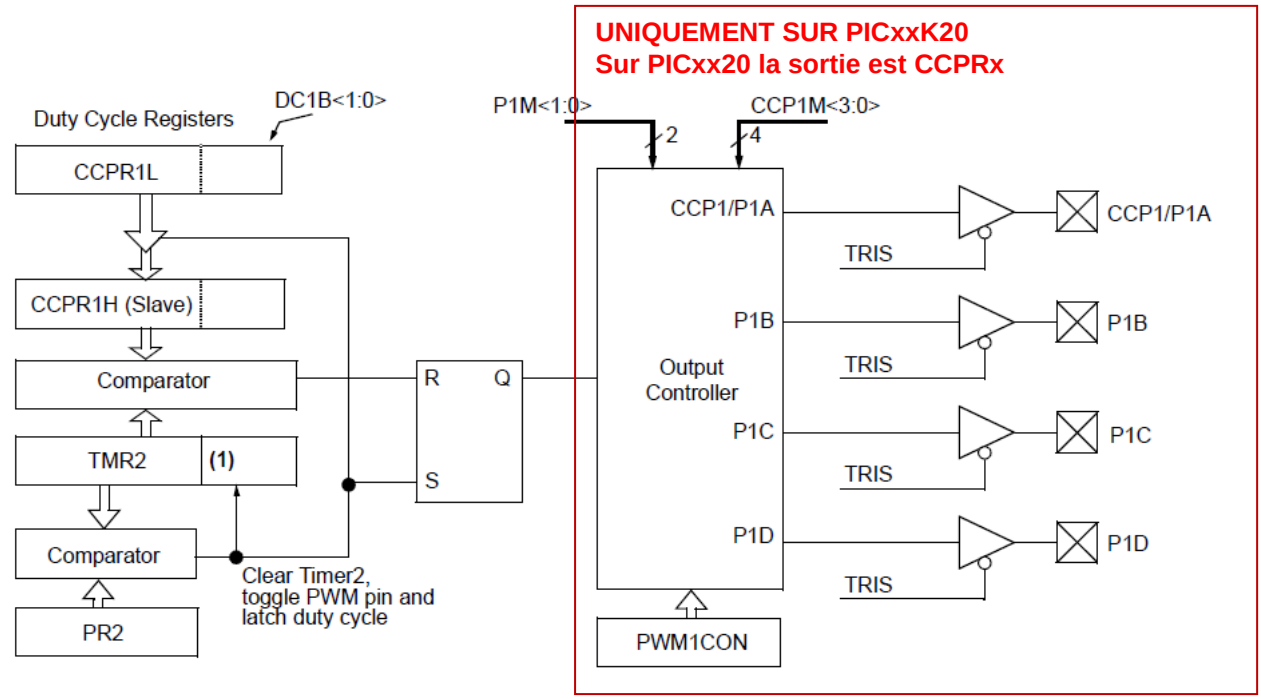
Standard Half-Bridge Circuit ("Push-Pull")

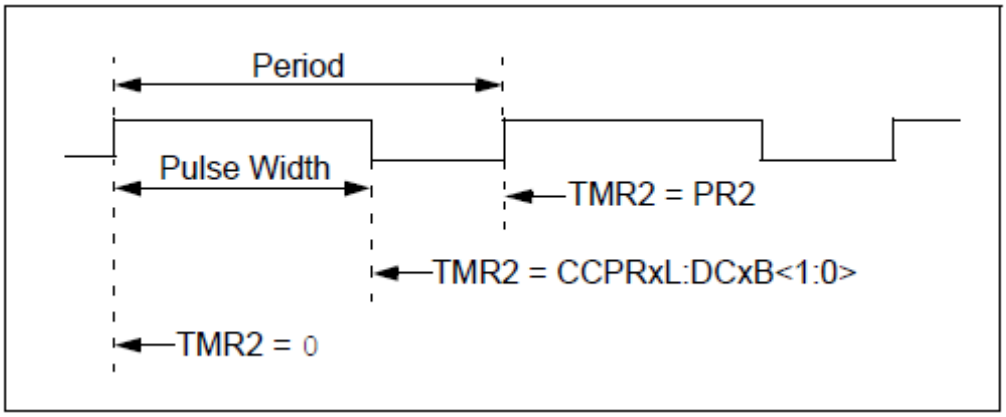


Full Bridge circuit



8.5.2. Structure de la fonction PWM sur PIC





A t=0 le signal passe à l'état haut, TMR2=0 et commence à compter (la période de comptage a été définie)
Lorsque TMR2<<2+ 2bits = CCPRxL<<2+DCB1:0, le signal est mis à 0
(les 2 bits associés à TMR2 dépendent de la configuration, voir ci-dessous, DCB1 :0 sont deux bits de CCPxCON)
Lorsque TMR2=PR2, TMR2 est mis à 0, le signal est mis à 1, CCPR1H est chargé avec CCPR1L

$$PWM\ Period = [(PR2) + 1] \bullet 4 \bullet Tosc \bullet (TMR2\ Prescale\ Value)$$

Note: Tosc = 1/Fosc.

$$Duty\ Cycle\ Ratio = \frac{(CCPRxL:DCxB<1:0>)}{4(PR2 + 1)}$$

La période PWM est définie par la configuration de l'horloge de TMR2 et la valeur de PR2.
Le rapport cyclique dépend de CCPR1xL qui est associé soit aux deux DCB1 :0 de CCPxCON soit aux deux bits d'horloge interne (Fosc), on peut ainsi améliorer la résolution de la PWM

8.5.3. Exemple de mise en œuvre du PWM sur PIC45K20

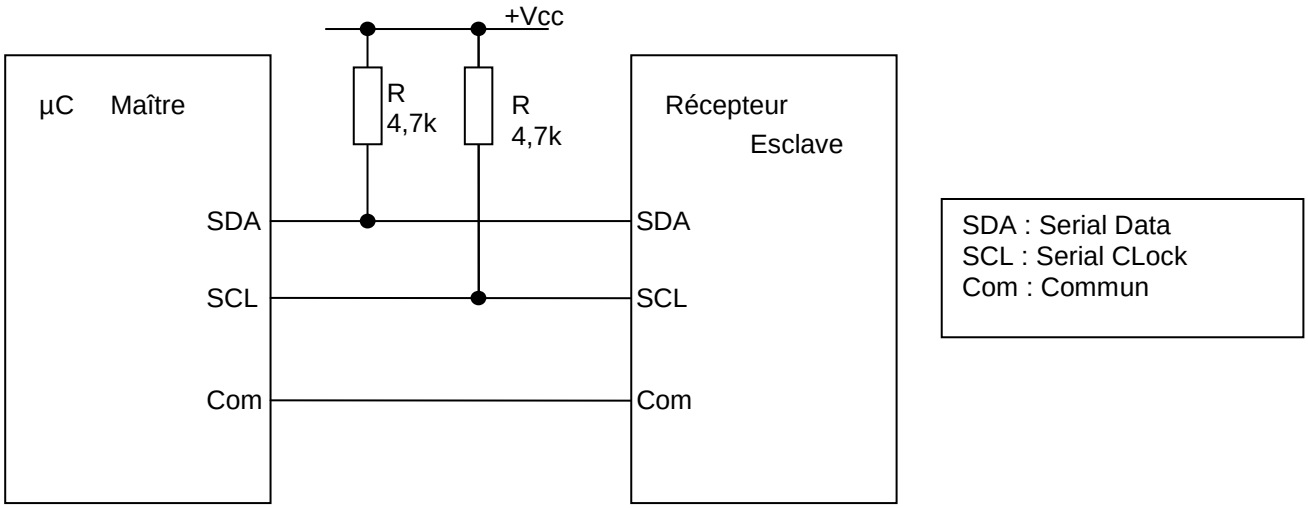
```
// Programme de demo PWM sur PCKIT3
#pragma config FOSC = INTIO67, FCMEN = OFF, IESO = OFF
#pragma config PWRT = OFF, BOREN = OFF, BORV = 30
#pragma config WDTEN = OFF, WDTPS = 32768
#pragma config MCLRE = ON, LPT1OSC = OFF, PBADEN = OFF, CCP2MX = PORTC

#include "p18f45k20.h"

void main (void)
{
    // RD7en sortie PWM pour commander la LED7
    // sortie P1D du PWM en mode etendu (uniquement sur P18FxxKxx)
    TRISDbits.TRISD7 = 0;

    // Timer2 définie la periode PWM
    // FPWM = Fosc/(4*prediv*PR2)
    // 1MHz clock / 4 = 250kHz (FCY).
    // (250kHz / 16 prescale) / 250) = 62.5Hz, soit une période de 16ms.

    T2CON = 0b00000111;// Prescale = 1:16, timer on
    PR2 = 249;         // Timer 2 Period Register = 250
    // CCPR1L définit le rapport cyclique en PPM
    // inialisation à 500 soit 0x1F4 (codé sur 10 bits)
    // les 8 bits de poid fort font 0x7D, les deux bits de poids faible (00) sont
   // rangés  dans DC1B1 and DC1B0 bits de CCP1CON
   CCPR1L = 0x7D;   // valeur d'initalisation PWM=50%
   CCP1CON = 0b01001100; // Le mode PWM etendu n'existe que sur les PIC18xxKxx
   // Sur le PICKIT seuls P1B et P1D sont conncetes à une LED et peuvent
   // être configurés comme sortie de modulation
   // on active le mode Full-Bridge Output forward ou P1D est la sortie de PWM
   // P1D actif à l'état haut

    while(1)
    {
        do
        {
// Le rapport cyclique est augmenté de 8 PPM (2 bits DC1Bx de poids faible ajoute
CCP1CON)
            CCPR1L += 4;
            PIR1bits.TMR2IF = 0;  // efface drapeau, mis à un lorsque TMR2 = PR2
            while (PIR1bits.TMR2IF == 0); // attend fin de periode
        } while (CCPR1L < 250);

        do
        {
            CCPR1L -= 4;    // // Le rapport cyclique est diminué de 8 PPM
            PIR1bits.TMR2IF = 0;
            while (PIR1bits.TMR2IF == 0);
        } while (CCPR1L > 1);
    };
}
```

8.6. Interface I2C

8.6.1. Description

Le bus I²C (Inter Integrated Circuit) a été développé par la société Philips dans au cours des années 80. Il permet d'établir entre des composants une communication série synchrone à l'aide de trois conducteurs.

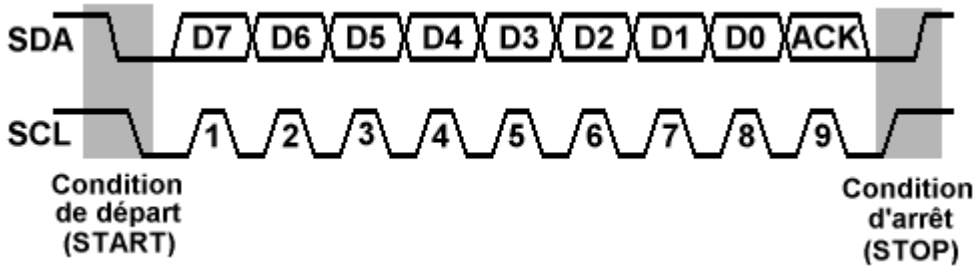
Structure du bus I²C :



Sur un même bus I²C, il est possible de connecter plusieurs récepteurs.

Protocole I²C :

a) Pour la transmission d'un octet on observe le chronogramme suivant :

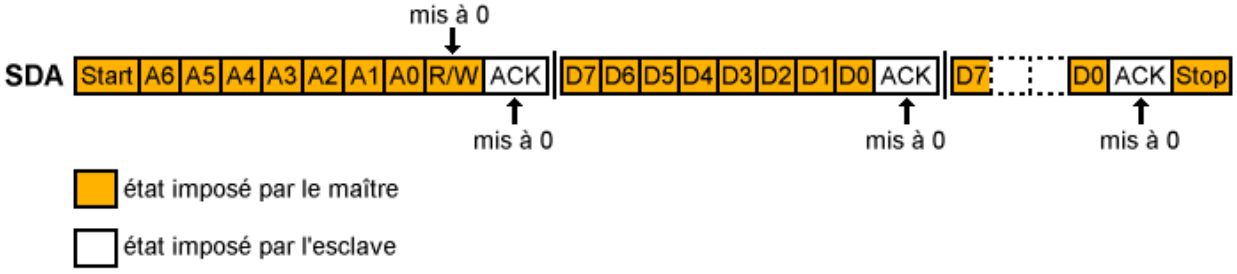


1. Les communications, sur le bus se déroule toujours à l'invite du maître (**master**). Emission de la condition de départ (START).
2. Emission par le maitre de la donnée sur un octet.
3. Emission par l'esclave d'un bit d'acquittement (ACK) pour indiquer au maître la bonne réception.
4. Emission par le maître d'un bit d'arrêt (STOP) pour mettre fin à la communication.

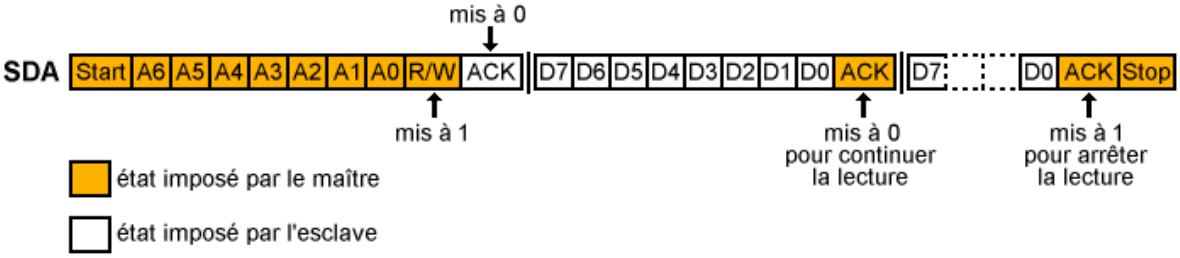
Remarque : La fréquence de l'horloge est de 100 kHz

b) La transmission d'une séquence d'échange entre un maître et un esclave :

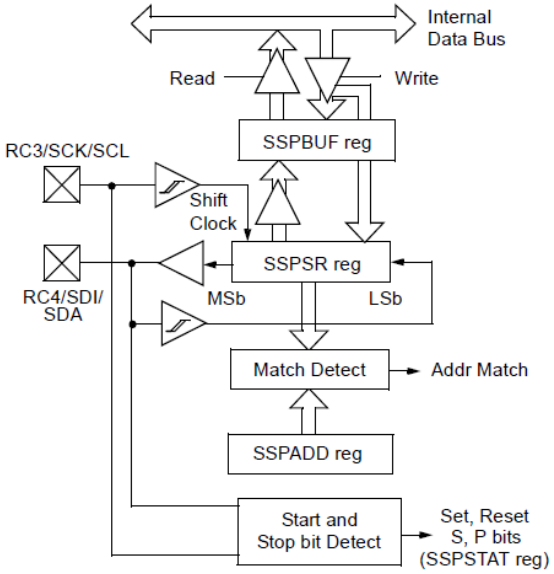
- Ecriture de données



- Lecture de données



8.6.2. Structure interface I²C (MSSP : Master Synchronous Serial Port)



REGISTRES UTILISES

SSPSTAT : Status Register (I²C)

7	6	5	4	3	2	1	0
SMP	CKE	$D/\overline{A}$	P	S	$R/\overline{W}$	UA	BF

SMP : Contrôle de SLew Rate
1 : Pas de contrôle de slew rate pour les fréquences standards (100kHz, 1 MHz)
0 : Contrôle de slew rate activé (f= 400 kHz)

CKE : Bit de sélection SMBus
1 : Fonctionnement en mode SMBus
0 : Fonctionnement en mode standard I²C

$D/\overline{A}$: Data /Adresse
Master Mode :
Non utilisé
Slave Mode :
1 : Le dernier octet reçu est une donnée

0 : Le dernier octet reçu est une adresse

P : Stop Bit
1 : Un bit de stop a été détecté
0 : Pas de bit stop détecté

$R/\overline{W}$: Bit Lecture / Ecriture
Master Mode :
1 : Transmission en cours
0 : Pas de transmission
Slave Mode :
1 : Lecture
0 : Ecriture

UA : Update Adress Bit
Slave Mode Only :
1 : L'utilisateur doit mettre à jour l'adresse
0 : Pas de besoin de mise à jour

BF : Buffer Full
1 : Registre SSPBUF occupé
0 : Registre SSPBUF vide

SSPCON1 : MSSP CONTROL REGISTER 1 (I²C)

7	6	5	4	3	2	1	0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0

WCOL
Master Mode :
1 : Une écriture dans le registre SSPBUF s'est produite pendant que le bus I²C était occupé (remis à zéro par logiciel).
0 : Pas de collision
Slave Mode :
1 : Ecriture d'une donnée dans le registre SSPBUF alors qu'il n'est pas vide (remis à zéro par logiciel)
0 : Pas de collision

SSPOV : Receive Overflow
Mode réception :
1 : Un octet a été reçu alors que le SSPBUF n'était pas vide (remis à zéro par logiciel)
0 : Pas de débordement
Mode transmission :
Non utilisé

SSPEN : Synchronous Serial Port ENable
1 : Mise en marche de l'interface I²C
0 : Arrêt de l'interface I²C

CKP : SCK Release Control bit
Master Mode :
Non utilisé
Slave Mode :
1 : horloge
0 : Maintien de l'horloge au niveau bas

SSPM3 :SSPM0 : Synchronous Serial Port Mode Select bits
1111 : I²C Mode esclave, 10 bit d'adresse, Bit de START et Stop en interruption
1110 : I²C Mode esclave, 7 bit d'adresse, Bit de START et Stop en interruption
1011 : I²C Firmware
1000 : I²C en mode Maître, fréquence d'horloge régler par le registre SSPAD
0111 : I²C Mode esclave, 10 bit d'adresse
0110 : I²C Mode esclave, 7 bit d'adresse

SSPCON2 : MSSP CONTROL REGISTER 2 (I²C)

7	6	5	4	3	2	1	0
GCEN	ACKSTAT	ACDT	ACKEN	RCEN	PEN	RSEN	SEN

GCEN : General Call Enable Bit (Mode esclave seulement)
1 : Validation de l'interruption si un appel général
0 : L'appel général est désactivé

ACKSTAT : Acknowledge Status bit (Mode Maître uniquement)
1 : Acquittement non reçu de l'esclave
0 : Acquittement reçu de l'esclave

ACKDT : Acknowledge Data bit (Mode maître en reception seulement)
1 : Pas d'acquittement
0 : émission d'un dit d'acquittement après réception d'un octet

ACKEN : Acknowledge Sequence Enable bit (Mode maître en reception seulement)
1 : initialise une séquence d'acquittement sur les broches SDA et SCL, puis transmet le bit ACKDT
0 : Séquence d'acquittement en attente

RCEN : Receive Enable bit (Mode Maître seulement)
1 : Validation du mode réception
0 : Récepteur en attente

PEN : Stop Condition Enable bit (Mode Maître seulement)
1 : Emission d'un bit Stop ; Remis à zéro par automatiquement
0 : Bit de Stop en attente

RSEN : Repetead Start Condition Enable bit (Mode Maître seulement)
1 : Réémission d'un bit de Start, Remis à zéro par automatiquement
0 : Réémission en attente

SEN : Start condition Enable
Master Mode :
1 : émission d'un bit d Start, Remis à zéro par automatiquement
0 : émission en attente
Slave Mode :
1 : L'horloge est validée pour la transmission et la reception
0 : Horloge n'est pas validée

SSPADD : MSSP ADDRESS AND BAUD RATE REGISTER 2 (I²C)

7	6	5	4	3	2	1	0
ADD7	ADD6	ADD5	ADD4	ADD3	ADD2	ADD16	ADD0

Master Mode :

[7 ::0] : Rapport de division du générateur de Baud : $F_{SCL} = \frac{F_{OSC}}{4*(ADD[7::0]+1)}$

Slave Mode Adresse sur 10 bits : Octet de poids fort

15-11 : Non utilisé
10-9 : bits de poids fort de l'adresse
8 : Non utilisé

Slave Mode Adresse sur 10 bits : Octet de poids faible
[7::0] : Octet de poids faible de l'adresse

Slave Mode Adresse sur 7 bits :
[7::1] : Adresse sur 7 bits
0 : Non utilisé

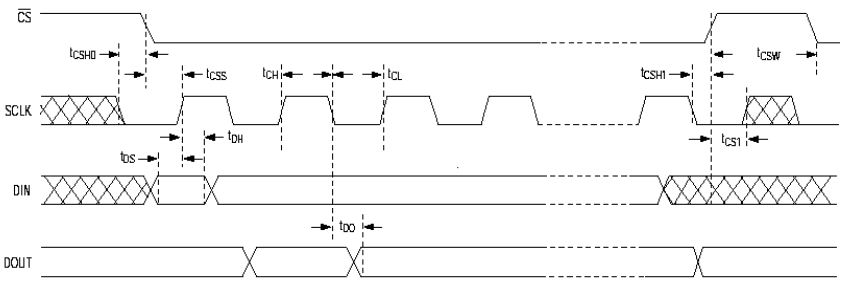
Exemple de programme communication I²C avec un PCF8574
/* Test I2C */

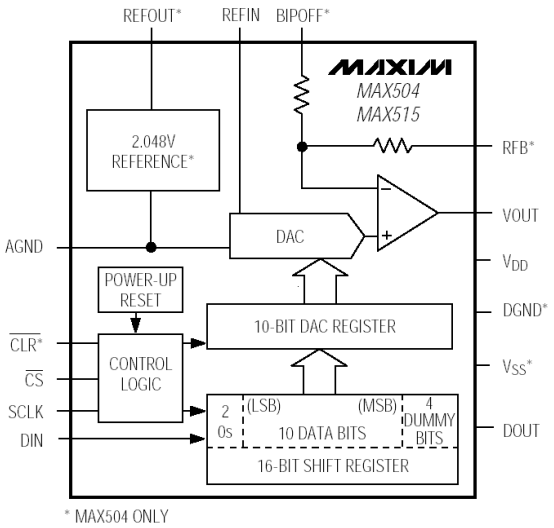
```
#include<p18f2520.h>
#include<delays.h>

/*Definition ds constante*/

/* Directives de configuration*/
#pragma config OSC = INTIO67, FCMEN = OFF, IESO = OFF            // CONFIG1H
#pragma config PWRT = OFF, BOREN = OFF, BORV = 3                 // CONFIG2L
#pragma config WDT = OFF, WDTPS = 1                              // CONFIG2H
#pragma config MCLRE = ON, LPT1OSC = OFF, PBADEN = ON, CCP2MX = PORTC  //
CONFIG3H
#pragma config STVREN = ON, LVP = OFF, XINST = OFF               // CONFIG4L
#pragma config CP0 = OFF, CP1 = OFF, CP2 = OFF, CP3 = OFF        // CONFIG5L
#pragma config CPB = OFF, CPD = OFF                              // CONFIG5H
#pragma config WRT0 = OFF, WRT1 = OFF, WRT2 = OFF, WRT3 = OFF    // CONFIG6L
#pragma config WRTB = OFF, WRTC = OFF, WRTD = OFF                // CONFIG6H
#pragma config EBTR0 = OFF, EBTR1 = OFF, EBTR2 = OFF, EBTR3 = OFF      //
CONFIG7L
#pragma config EBTRB = OFF                                       // CONFIG7H

/* Sous programme */
/* Initialisation I²C*/
void init_i2c(void)
{
      TRISCbits.TRISC3=1;
      TRISCbits.TRISC4=1;
      SSPADD=19;                    // Fréquence de l'horloge 100khz à 8Mhz
      SSPCON1=0b00101000;           // I2C valider  SSPCON1.SPE=1 et SSPPM = 1000
=> Master
      SSPSTATbits.SMP =1;           // pas de SRate
      SSPCON2bits.ACKSTAT=0;


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

/* Ecriture  */

void ecriture(char adresse ,char donnee )
{
      /* Constitution adresse recepteur */
      adresse = adresse << 1;
      adresse = adresse &0x0E;
      adresse = adresse |0x40;

       SSPCON2bits.SEN=1;             // Emission Start Bit
       while (SSPCON2bits.SEN);       // Fin Start condition ?
       SSPBUF=adresse;                // Envoi adresse + Ordre Ecriture
       ack();                         // Adresse reçue?
       SSPBUF = donnee;               // Emission de la donnée
       ack();                         // Donnée reçue
       SSPCON2bits.PEN=1;             // Envoi fin transmmission
       while (SSPCON2bits.PEN);       // Test envoi

}
```

```
/* Programme Principal*/

void main (void)
{
      char data;

/* initialisation Horloge*/
      OSCCONbits.IRCF0=0;
      OSCCONbits.IRCF1=1;
      OSCCONbits.IRCF2=1;
/* Initialisation I²C */
      init_i2c();

      while (1)
      {
            ecriture(5,data);
            data++;
            Delay1KTCYx(255);
      }

}
```

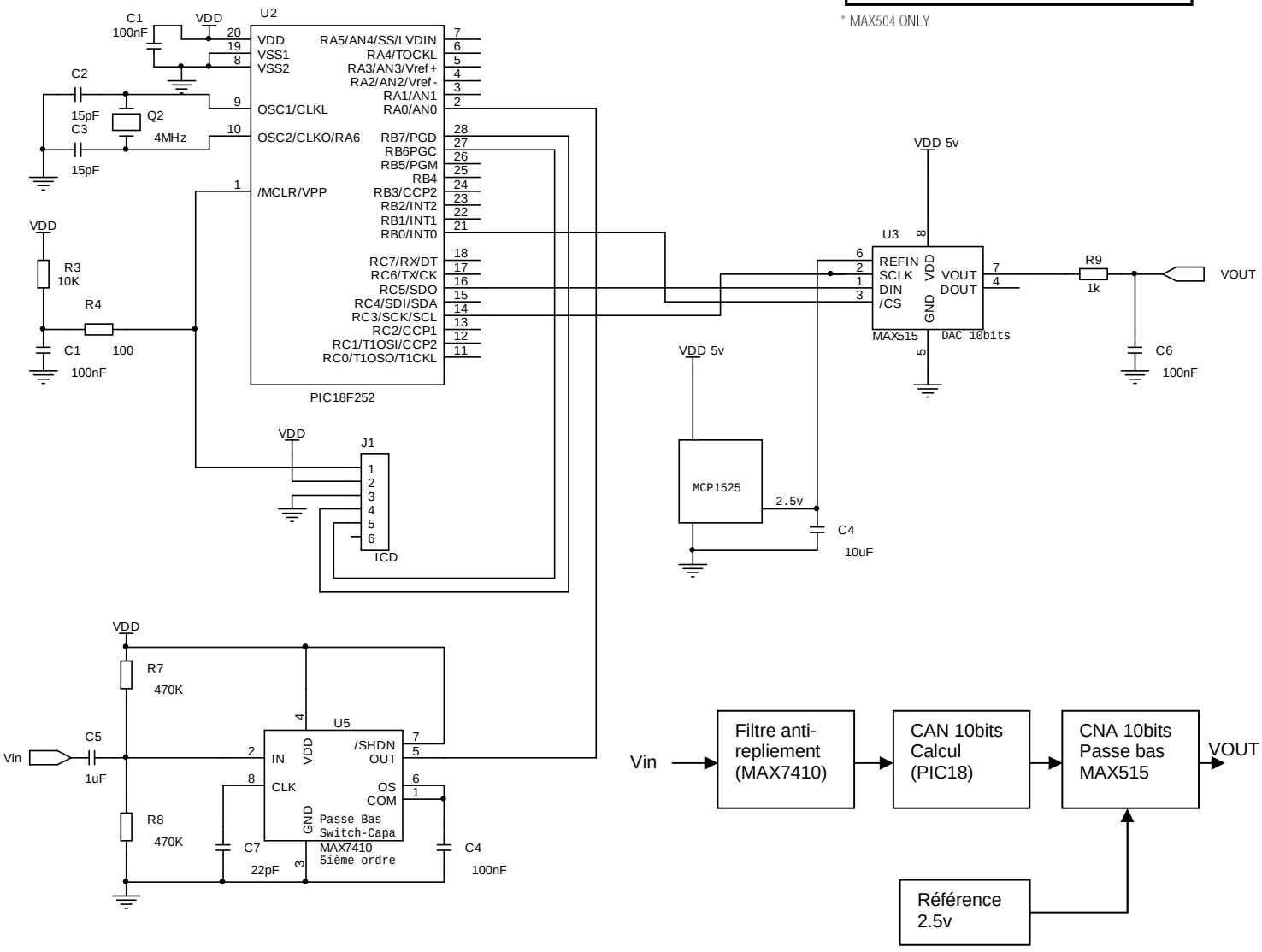
8.7. Bus SPI

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





Exemple de connexion (filtre numérique):



```
// 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
            }
}
```