

## CHAPITRE 3

### PROGRAMMATION C DES PIC

### AVEC LE COMPILATEUR CCS - C

#### 1. Outils de programmation d'un PIC

Comme toute solution programmable, le microcontrôleur nécessite un outillage informatique et éventuellement un programmeur. A chaque microcontrôleur correspond son outil de développement.

Pour développer une application fonctionnant à l'aide d'un microcontrôleur, il faut disposer de :

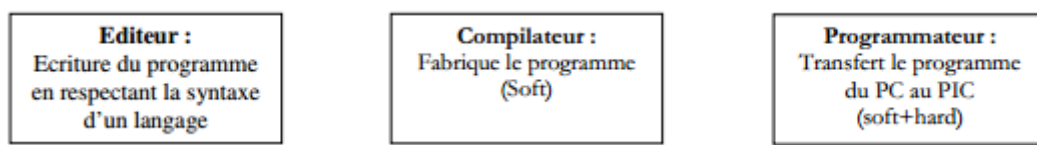


Fig. 3.1 : Outils de développement pour un PIC

- **Le compilateur** : Logiciel traduisant un programme écrit dans un langage donné (C, basic, assembleur) en langage machine.
- **Le programmeur** : Transfert le programme compilé (langage machine) dans la mémoire du microcontrôleur. Il est constitué d'un circuit branché sur le port série du PC, sur lequel on implante le PIC, et d'un logiciel permettant d'assurer le transfert. Il existe différents logiciels, nous utiliserons Icprog

Les microcontrôleurs PIC utilisent la plate-forme logiciel de développement MPLAB Integrated Development Environment IDE.

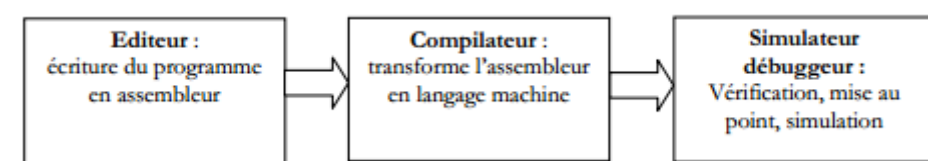


Fig. 3.2 : Environnement MPLAB IDE

Dans l'environnement MPLAB, Le programme doit être écrit en assembleur, langage proche de la machine et donc nécessitant un long apprentissage.

Le langage de programmation adopté pour programmer le PIC 16F877 est le langage évolué : Le code source écrit en langage c doit donc être compilé en assembleur à l'aide d'un compilateur C.

## 2. Le langage C

Le langage C dispose de beaucoup d'avantages. Il est :

- ✓ PORTABLE : Les modifications d'un programme pour passer d'un système à un autre sont minimales.
- ✓ COMPLET : Un texte C peut contenir des séquences de bas niveau (proches du matériel) en assembleur.
- ✓ SOUPLE : Tout est possible en C mais une grande rigueur s'impose.
- ✓ EFFICACE : On réfléchit (devant une feuille de papier) et on écrit (peu)

Le compilateur C de la société CCS (Custom Computer Services) est un compilateur C adapté aux microcontrôleurs PICs. Il ne respecte pas complètement la norme ANSI, mais il apporte des fonctionnalités très intéressantes.

## 3. Notion de filière de développement

On désigne par filière de développement l'ensemble des outils qui interviennent pour passer du fichier texte (source codé en C) au code objet (code machine) téléchargé dans le microcontrôleur.

Les étapes de génération d'un programme écrit en langage C sont :

- ✓ L'édition du fichier source `mon_programme.C` avec un éditeur de texte (simple sans mise en forme du texte).
- ✓ La compilation du fichier source pour obtenir un fichier objet : `mon_programme.ASM`. La compilation est la transformation des instructions C en instructions assembleur pour microcontrôleur PIC.
- ✓ L'édition de liens permet d'intégrer des fonctions prédéfinies. Le programme auxiliaire Éditeur de liens (linker ou binder) génère à partir du fichier `mon_programme.ASM` un fichier exécutable `mon_programme.HEX` compatible avec le PIC.

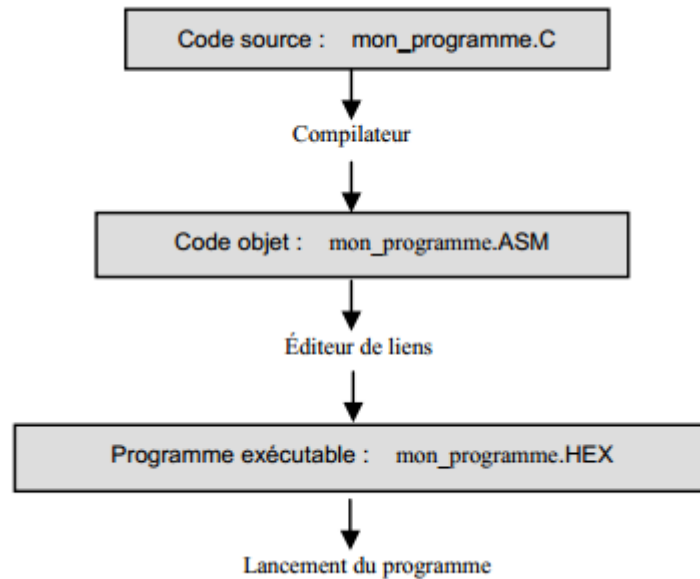


Fig. 3.3 : Etapes de génération d'un programme écrit en langage C

Fichiers (Extensions)	Description du contenu du fichier	Fichiers (Extensions)	Description du contenu du fichier
<b>.C</b>	Fichier source en langage C.	<b>.HEX</b>	Code objet (exécutable) téléchargé dans le µC
<b>.H</b>	Entête de définition des broches, Registres, Bits de Registres, Fonctions, et directives de pré-compilation.	<b>.TRE</b>	Montre l'organisation du programme sous forme d'arbre (découpages en fonction) et l'utilisation de la mémoire pour chaque fonction.
<b>.PJT</b>	Fichier de projet (pas obligatoire).	<b>.COF</b>	Code machine + Informations de débogage
<b>.LST</b>	Fichier qui montre chaque ligne du code C et son code assembleur associé généré.	<b>.ERR</b>	Erreurs éventuelles rencontrées durant la compilation.
<b>.SYM</b>	Indique la correspondance entre le nom des symboles (variables, bits, registres) et leur adresses hexadécimale en mémoire.		
<b>.STA</b>	Fichier statistique sur l'espace mémoire occupé, etc.		

Fig. 3.4: Fichiers générés

## 4. Règles de bases

Toutes instructions ou actions se terminent par un point virgule ;

Une ligne de commentaires doit commencer par /\* et se terminer par \*/ ou commencer par //.

Un bloc d'instructions commence par { et se termine par }.

Un programme en C utilise deux zones mémoires principales :

- ✓ La zone des variables est un bloc de RAM où sont stockées des données manipulées par le programme.

- ✓ La zone des fonctions est un bloc de ROM qui reçoit le code exécutable du programme et les constantes.

## 5. Les variables et les constantes

### 5.1. Les constantes

Les constantes n'existent pas, c'est-à-dire qu'il n'y a pas d'allocation mémoire, mais on peut affecter à un identificateur (Nom : Il ne doit pas dépasser 32 caractères, sans accent)

Une valeur constante par l'instruction **#define**.

**Syntaxe :** <#define> <identificateur> <valeur> ;

**Exemple:** #define PI 3.14

### 5.2. Déclarations spécifiques au compilateur CCS

**#bit id = x,y**

- ✓ Id : identifiant (Nom d'un bit)
- ✓ X : Nom du variable ou d'une constante
- ✓ Y : position du bit

**Exemple :** #bit RW = PORTA,2

#bit LED\_R=PortB.4

**#byte id = X**

- ✓ Id: identifiant
- ✓ X: valeur 8 bits

**Exemple :**

```
#byte PORTA = 5           // adresse du port A
#byte PORTB = 6           // adresse du port B
#byte PORTC = 7           // adresse du port C
#byte PORTD = 8           // adresse du port D
#byte PORTE = 9           // adresse du port E
```

### 5.3. Les Variables

Les variables sont définies par signé ou non signé, le type et l'identificateur.

**Syntaxe:** <signed> <type> <identificateur1>, ..., <identificateurn>

L'identificateur : C'est le nom (Il ne doit pas dépasser 32 caractères, sans accent) affecté à la variable.

Le type : Il détermine la taille de la variable et les opérations pouvant être effectuées. On peut rajouter le mot `signed` devant le type du variable, alors la variable devient signée.

**Exemples :** `Int A,B,C,D ;`

`Char MESSAGE[10] ;`

`Signed int A ; // Entier de type signé, de -128 à +127`

Les types du compilateur CCS figurent dans le tableau ci-dessous :

**Tab 3. 1: Types du compilateur CCS**

Type	Taille	Valeur
<code>int1</code>	1 bit	0 ou 1
<code>Int8</code>	8 bits	De 0 à 255
<code>int16</code>	16 bits	De 0 à 65535
<code>int32</code>	32 bits	De 0 à 4 294 967 295
<code>Char</code>	8bits	De 0 à 255
<code>Float</code>	32 bits	
<code>Short</code>	1 bits	0 ou 1
<code>Int</code>	8 bits	De 0 à 255
<code>Long</code>	16 bits	De 0 à 65535

#### 5.4. Représentation des différentes bases du compilateur CCS et du code ASCII

`int a = 4 ; // Un nombre seul représente un nombre décimal.`

`int b = 0b1010 ; // Un nombre précédé de 0b est un nombre binaire.`

`int p = 0x00FF ; // Un nombre précédé de 0x est un nombre hexadécimal.`

`char c = 'A' ; char c = '0x41' ; ou char c = '65' ; // Un caractère entre '' représente son code ASCII.`

### 6. Les fonctions

➤ Avec des paramètres d'entrée et un paramètre de sortie

**Syntaxe :**

Type de la variable de retour    nom de fonction (types nom des paramètres)

{

    Instruction 1 ;

.

    Instruction n ;

**Return (valeur) ; // Valeur à renvoyer**

}

Avec :

//Nom de la fonction :

//Description du rôle de la fonction :

//Paramètres d'entrée : Noms et types des paramètres d'entrée

//Paramètre de sortie : Nom et type du paramètre de sortie

➤ **Une fonction sans paramètres d'entrée et de sortie**

Void nom de fonction (Void)

{

Instruction 1 ;

.

. Instruction n ;

}

**Remarque :** Il peut y avoir aussi des fonctions avec paramètres d'entrées et sans paramètre de sortie et vice versa.

## 7. Les Opérateurs

**Tab 3. 2: Les opérateurs d'affectation et arithmétiques**

Type	Symbole	Exemple
Affectation	=	x = 3; y = a - b;
Addition	+	a = a + b; b = a + 5;
Soustraction	-	a = a - b; b = a - 5;
Moins unitaire	-	a = -b;
Multiplication	*	a = b * 6;
Division	/	x = a / b;
Reste de la division	%	r = a % b;

**Tab 3. 3: Les opérateurs de comparaison**

Type	Symbole	Exemple
Egalité	==	a == b;
Différent	!=	a != b;
Supérieur	>	a > b;
Supérieur ou égal	>=	a >= b;
Inférieur	<	a < b;

Inférieur ou égal	$\leq$	$a \leq b;$
Reste de la division	$\%$	$r = a \% b;$

Le résultat de la comparaison peut prendre deux valeurs VRAI ou FAUX.

-FAUX correspond à 0.

-VRAI correspond à toute valeur différente de 0

**Tab 3. 4: Les opérateurs logique de comparaison**

Type	Symbole	Exemple
Et logique	$\&\&$	$c = a \&\& b;$ //c est vrai si a et b sont vrais, sinon c est faux
Ou logique	$\ $	$c = a \  b;$ //c est vrai si a et b sont vrais, sinon c est faux
Non logique	$!$	$c = !c;$ //c prend le complément de ce qu'il vaut

Ces opérateurs permettent la comparaison de conditions composées telles que  $(x > y) \&\& (u > v)$ . Le résultat fourni est de type logique (0 ou 1).

**Tab 3. 5: Les opérateurs binaires bit à bit**

Type	Symbole	Exemple
Et logique	$\&$	$c = a \& b;$
Ou logique	$ $	$c = a   b;$
Et exclusif	$\wedge$	$c = a \wedge b;$
Complément à 1	$\sim$	$a = \sim a;$
Décalage de n bits à droite	$\gg$	$a = b \gg n;$
Décalage de n bits à gauche	$\ll$	$a = b \ll n;$

**Exemples :** a et b deux entiers tel que :  $a = 1100\ 0111\ 0101\ 0011$  (0xC753)

$b = 0001\ 1001\ 1010\ 1110$  (0x19AE)

$a \& b = 0000\ 0001\ 0000\ 0010$  (0x0102)

$a | b = 1101\ 1111\ 1111\ 1111$  (0xDFFF)

$a \wedge b = 1101\ 1110\ 1111\ 1101$  (0xDEFD)

$\sim a = 0011\ 1000\ 1010\ 1100$  (0x38AC)

$\sim b = 1110\ 0110\ 0101\ 0001\ (0xE651)$

$a \ll 2 = 0001\ 1101\ 0100\ 1100\ (0x1D4C)$

## 8. Les structures répétitives

Tab 3. 6: Les structures itératives

Structure “while”	Structure “do ... while” : faire ... tant que...	Structure “for” :	Structure “if ... Else” :	Structure “switch ... case”.
tant que ... faire ...		Pour <variable> allant de <valeur initiale> à <valeur finale> faire...	Si <condition> faire ... sinon faire ...	
<pre>void main() {     while     (condition vraie)     {         Action 1;     } }</pre>	<pre>void main() {     do     {         Action ;     }     while     (condition vraie) }</pre>	<pre>void main() {     For (i=m ; i&lt;=n ;     i=i+p)     {         Action ;     } }</pre>	<pre>void main() {     if (condition vraie)     {         Action 1;     }     else     {         Action 2;     } }</pre>	<pre>switch (choix) case c1 : &lt; sequence 1&gt; case c2 : &lt; sequence 2&gt; case c3: &lt; sequence 3&gt; .... case cn: &lt; sequence n&gt; default: &lt; sequence_par_defaut&gt; /</pre>

## 9. Les fonctions adaptées aux microcontrôleurs PIC

### 9.1. Les directives

➤ **#use delay**

**Syntaxe : #use delay(clock=fréquence)**

Renseigne le compilateur sur la fréquence du quartz utilisé.

**Exemple : #use delay(clock=4000000)**

➤ **#fuses**

**Syntaxe : #fuses options**

Permet de définir le mot de configuration. Les options sont :

- LP, XT, HS, RC : Choisir le type d'oscillateur
- WDT, NOWDT : Activer/désactiver le chien de garde



- PUT, NOPUT : Activer/désactiver le **Power Up Timer** PUT( Timer spécial qui retarde le démarrage de l'exécution du programme après que le PIC a été réinitialisé. Ce délai donne le temps de PIC oscillateur pour démarrer et se stabiliser).
- PROTECT, NOPROTECT : Activer/Désactiver la protection du code

**Exemple :** #fuses XT,NOWDT,NOPUT,NOPROTECT

➤ **#int\_xxxx**

Spécifie la source de l'interruption.

**Syntaxe :**

#int\_ext : interruption externe.

## 9.2. La gestion des entrées et des sorties

### 9.2.1 Configurer un port en entrée ou en sortie

**Syntaxe :** set\_tris\_X(valeur)

Avec :

- X : Nom du port : A .. E
- valeur : définit la configuration du port (1 entrée et 0 sortie)

**Exemple :**

```
set_tris_A(0b00001111) ; PA0 à PA3 en entrée et P A4 à PA7 en sortie
set_tris_A ( 0x0F ); // B7,B6,B5,B4 en sortie (0)
                    // B3,B2,B1,B0 en entrée (1)
```

### 9.2.2. Gestion des ports d'entrée ou en sortie

➤ **Mise à l'état bas d'une pine d'un port**

**Syntaxe :** output\_low(pin)

- pin est une constante définie dans le fichier entête PIC16FXX.h

**Exemple :** output\_low(PIN\_A0); // mise à 0 de la broche A0

➤ **Mise à l'état haut d'une pine d'un port**

**Syntaxe :** output\_high(pin)

**Exemple :** output\_high(PIN\_B2); // mise à 0 de la broche B2

➤ **Modifier l'état d'une pine d'un port**

**Syntaxe :** output\_bit(pin,valeur)

**Exemple :** output\_bit( PIN\_B0, 0); // même rôle que output\_low(pin\_B0);

➤ **Modifier l'état d'un port**

**Syntaxe : output\_X(valeur)**

**Exemple :** `output_b(0xf0); ); // mise à 1 de B7,B6,B5,B4`  
`// mise à 0 de B3,B2,B1,B0`

➤ **Lecture de l'état d'une pine****Syntaxe : valeur=input(pin)**

**Exemple :** `if( input(PIN_A0) ) // Si A0 est à l'état haut, écrire « A0 est active »`  
`printf("A0 est active \r\n");`

➤ **Lecture de l'état d'un port****Syntaxe : valeur=input\_X()**

**Exemple :** `Data = input_c(); //data reçoit l'état du port c`

➤ **Mettre à 0 un bit d'une variable****Syntaxe : BIT\_CLEAR(var, bit)****Exemple :**

`a=0x1F`  
`BIT_CLEAR(a, 3) // a devient 17 hexa.`

➤ **Mettre à 1 un bit d'une variable****Syntaxe : BIT\_SET(var, bit)****Exemple :**

`a=0x1F`  
`BIT_SET(a, 6) // a devient 3F hexa.`

➤ **Test de l'état d'un bit d'une variable****Syntaxe : BIT\_TEST(var, bit)****Exemple :**

`a=0x1F`  
`BIT_TEST(a, 2) // Le résultat est 1 car le bit 2 de la variable « a » est à 1.`

**9.3. Gestion des temporisations**

Le compilateur intègre des fonctions très pratiques pour gérer les délais, à savoir :

**delay\_us(valeur) ;** // temporisation en  $\mu$ S

**delay\_ms(valeur) ;** // temporisation en mS

Pour pouvoir utiliser ces fonctions, il faut indiquer par la ligne ci-dessous la fréquence du Quartz de votre application. Cette ligne doit être définie au début du programme.

**#use delay (clock=fréquence\_du\_quartz)**

**Exemples :**

```
#use delay (clock=4000000) // Quartz de 4Mhz
#use delay (clock=20000000) // Quartz de 20Mhz
```

**9.4. Gestion de la liaison série**

Toutes les fonctions d'entrée et de sortie peuvent être redirigées sur le port série dumicrocontrôleur, Il suffit d'ajouter la ligne ci-dessous pour configurer le port série :

```
#use rs232 (BAUD=9600, xmit=PIN_C6, rcv=PIN_C7)
```

// Cette ligne configure la liaison série du PIC avec une vitesse de 9600 bauds.

**Remarque :** Cette ligne doit être définie au début du programme, après la ligne qui définit la Fréquence du quartz.

Alors les fonctions suivantes utiliseront le port série comme moyen de communication.

➤ **La fonction printf**

Cette fonction permet d'envoyer une chaîne de caractère formatée sur la liaison RS232 ou bien vers une fonction bien déterminée.

**Syntaxe : printf (chaîne) ou printf (Cchaîne, valeurs...)**

- Chaîne peut être une constante de type chaîne ou un tableau de char terminé par le caractère null.
- Cchaîne est une chaîne composée (voir ci-dessous).
- valeurs est une liste de variables séparées par des virgules.

La « Cchaîne » doit être composée de chaînes de caractères constante et d'arguments de mise en forme des valeurs représenté par le symbole %wt (formatage) avec :

- w est optionnel et peut être compris entre 1 et 9. il indique sur combien de caractère va être le résultat ou 01-09 ou 1.1 to 9.9 pour les nombres à virgule.

- t est le type et peut être :

- ✓ c Caractère ascii
- ✓ c caractère ou chaîne de caractère
- ✓ u entier (int8 ou int 16) non signé affiché en base décimale
- ✓ x entier (int8 ou int 16) affiché en base hexadécimale en minuscules
- ✓ X entier (int8 ou int 16) affiché en base hexadécimale en majuscules
- ✓ d entier (int8 ou int 16) signé affiché en base décimale
- ✓ f Float
- ✓ Lx entier long (int32) affiché en base hexadécimale en minuscules

- ✓ LX long (int32 affiché en base hexadécimale en majuscules)
- ✓ lu entier long (int32) non signé affiché en base décimale
- ✓ ld entier long (int32) signé affiché en base décimale
- ✓ : entier (int8 ou int 16) signé affiché en base octale

(Pour les autres arguments voir la documentation de CCS-Compiler)

**Exemple :**

```
printf("Bonjour !");  
printf("\r\nMin: %2X Max: %2X\n\r",min,max);  
// \n=LF (saut de ligne), \r=CR (retour à la première colonne)  
printf("A/D value = %2x\n\r", value);
```

**➤ La fonction putc**

Cette fonction permet d'envoyer un caractère formatée sur la liaison RS232.

**Syntaxe : putc (char)**

- Char est un caractère 8 bits

**➤ La fonction puts**

Cette fonction permet d'envoyer une chaîne de caractères sur la liaison RS232, terminée par le passage à la ligne (LF)

**Syntaxe : variable=puts(chaine)**

- chaîne est une chaîne de caractère constante terminée par le caractère null

**Exemple :** puts( " | Salut ! | " );**➤ La fonction getc**

Cette fonction permet de recevoir un caractère formatée sur la liaison RS232.

**Syntaxe : variable=getc()**

- Variable est un caractère 8 bits

**Exemple :**

```
void main () {  
printf("Continuer (O,N)?");  
do  
{ reponse=getc(); }  
while (reponse!='O' &&reponse!='N'); }
```

**➤ La fonction gets**

Cette fonction permet de recevoir une chaîne de caractère sur la liaison RS232

**Syntaxe : variable=gets()**

-Variable est une chaîne de caractère constante terminée par le caractère null

**10. Structure d'un programme en C**

```
#include <16F84A.H>    //fichier de déclaration des registres internes du 16F84A.H
//Déclaration des adresses des ports E/S
#byte PORTA = 5        //adresse du port A
#byte PORTB = 6        // adresse du port B
//Déclaration des constantes
#define NB_MAX 100
//Affectation des entrées et sorties de type bit
#bit BUZZER = PORTD7    //par exemple : Command d'un buzzer
//Fréquence du quartz
#use delay (clock=20000000)
//Configuration de la liaison série de PIC avec une vitesse de 9600 bauds
#use rs232 (BAUD=9600, xmit=PIN_C6, rcv=PIN_C7)
//Déclaration du prototype de toutes les fonctions logicielles
//Nom de la fonction suivi d'un point virgule par exemple :
Void INIT_UC(Void) ;
//Déclaration des variables globales Par exemple :
Long DEBIT ;
Long VOULUME ;
Main() //Programme principale
{
    Instruction 1 ;
    Instruction n ;
}
//Déclaration des fonctions logicielles Par exemple :
//Nom de la fonction : DECALAGE_DROITE
//Description du rôle de la fonction : décalage à droite de NB bits
//Paramètres d'entrée : entier VAL, entier NB
//Paramètre de sortie : entier RESULTAT
int DECALAGE_DROITE (int VAL ,int NB)
{
```

```
Int RESULTAT;
```

```
RESULTAT = VAL >> NB;
```

```
Return (RESULTAT);}
```

```
//Appel de la fonction: Nom de la fonction (nom des paramètres) ;
```

```
//Exemple :
```

```
A = 16 ;
```

```
B = DECALAGE_DROITE (A, 2) ; //la valeur de A sera affectée à VAL
```

```
                //la valeur 2 sera affectée à NB
```

```
                //le résultat de la fonction sera affectée à B
```