Etude et programmation des microcontrôleurs

Nous allons au cours de ce chapitre définir le microcontrôleur ainsi que les éléments contenus dans ce dernier et qui n'ont pas été abordé au cours des chapitres précédents afin de comprendre l'architecture d'un système à microcontrôleurs, le pic 16f877 sera utilisé comme exemple dans la suite du chapitre. Nous allons aussi voir la manière de programmer un microcontrôleur avec un langage évolué.

6.1. Définition

Un microcontrôleur se présente sous la forme d'un circuit intégré réunissant tous les éléments d'une structure à base de microprocesseur. Voici généralement ce que l'on trouve à l'intérieur d'un tel composant :

- Un microprocesseur (C.P.U.),
- De la mémoire de donnée (RAM et EEPROM),
- De la mémoire programme (ROM, OTPROM, UVPROM ou EEPROM),
- Des interfaces parallèles pour la connexion des entrées / sorties,
- Des interfaces séries (synchrone ou asynchrone) pour le dialogue avec d'autres unités,
- Des timers pour générer ou mesurer des signaux avec une grande précision temporelle,
- Des convertisseurs analogique / numérique pour le traitement de signaux analogiques.

Le fonctionnement d'un microcontrôleur est cadencé par une horloge généralement à base de quartz ou de résonateur céramique.

Il peut aussi contenir les composants suivants :

- Un Watchdog : (surveillance du programme)
- Une sortie PWM (modulation d'impulsion)
- Une interface I²C.

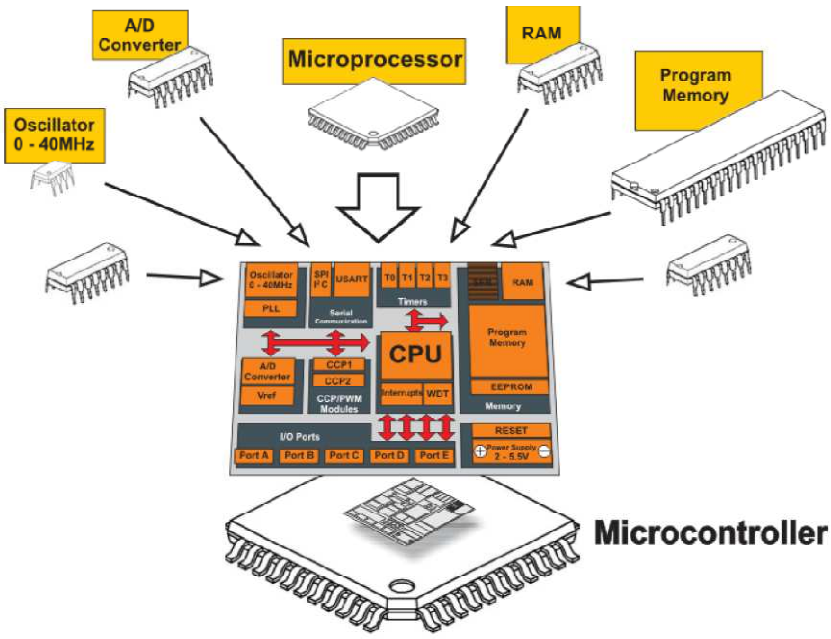

Figure 19: les composants d'un microcontrôleur

Les microcontrôleurs présentent donc les avantages suivants :

- Encombrement réduit,
- Faible consommation,
- Coût réduit
- Fiabilité
- Mise en œuvre plus simple

6.2. Les composants d'un microcontrôleur : (PIC 16F877)

6.2.1. Présentation du PIC 16F877

La famille des PICs est subdivisée en 3 grandes familles : La famille **Base-Line**, qui utilise des mots d'instructions de 12 bits, la famille **Mid-Range**, qui utilise des mots de 14 bits (et dont font partie le 16F877), et la famille **High-End**, qui utilise des mots de 16 bits.

Les éléments essentiels du PIC 16F877 sont :

- Consommation : moins de 2mA sous 5V à 4 MHz.
- Architecture RISC : 35 instructions de durée 1 ou 2 cycles.

- Durée du cycle : Période de l'oscillateur quartz divisée par 4 soit 200 ns pour un quartz de 20 MHz.
- Deux bus distincts pour le code programme et les data.
- Code instruction : mot de 14 bits et compteur programme (PC) sur 13 bits, ce
- qui permet d'adresser 8 K mots (de h'0000' à h'1FFF')
- Bus DATA sur 8 bits.
- 33 Ports Entrée-Sortie bidirectionnels pouvant produire 25 mA par sortie.
- PORTA = 6 bits et PORTB PORTC et PORTD = 8bits PORTE = 3 bits pour le
- 16F877 et 22 I/O seulement pour le 16F876.
- 4 sources d'interruption :

    - ✓ Externe par la broche partagée avec le Port B : PB0
    - ✓ Par changement d'état des bits du Port B: PB4 PB5 PB6 ou PB7
    - ✓ Par un périphérique intégré dans le chip: écriture de Data en EEPROM terminée, conversion analogique terminée, réception USART ou I2C.
    - ✓ Par débordement du Timer.

- 2 Compteurs 8 bits et 1 compteur 16 bits avec pré diviseur programmable.
- Convertisseur analogique 10 bits à 8 entrées.
- UART pour transmission série synchrone ou asynchrone.
- Interface I2C.
- 2 modules pour PWM avec une résolution de 10 bits.
- Interface avec un autre micro: 8 bits + 3 bits de contrôle pour R/W et CS.
- 368 Octets de RAM
- 256 Octets d'EEPROM Data.
- 8K mots de 14 bits en EEPROM Flash pour le programme (h'000' à h'1FFF').
- 1 registre de travail : W et un registre fichier : F permettant d'accéder à la RAM ou aux registres internes du PIC. Tous les deux sont des registres 8 bits.

- **PORTA :** 6 entrées -sorties. 5 entrées du CAN. Entrée CLK du Timer 0.

- **PORTB :** 8 entrées-sorties. 1 entrée interruption ext. Clk et Data pour prog.

- **PORTC :** 8 entrées-sorties. Clk Timer1 et PWM1. USART. I2C.

- **PORTD :** 8 entrées-sorties. Port interface micro processeur (8 bits data).

- **PORTE :** 3 entrées-sorties. 3 bits de contrôle. 3 entrées du CAN.

6.2.2. Organisation de la mémoire du 16F877

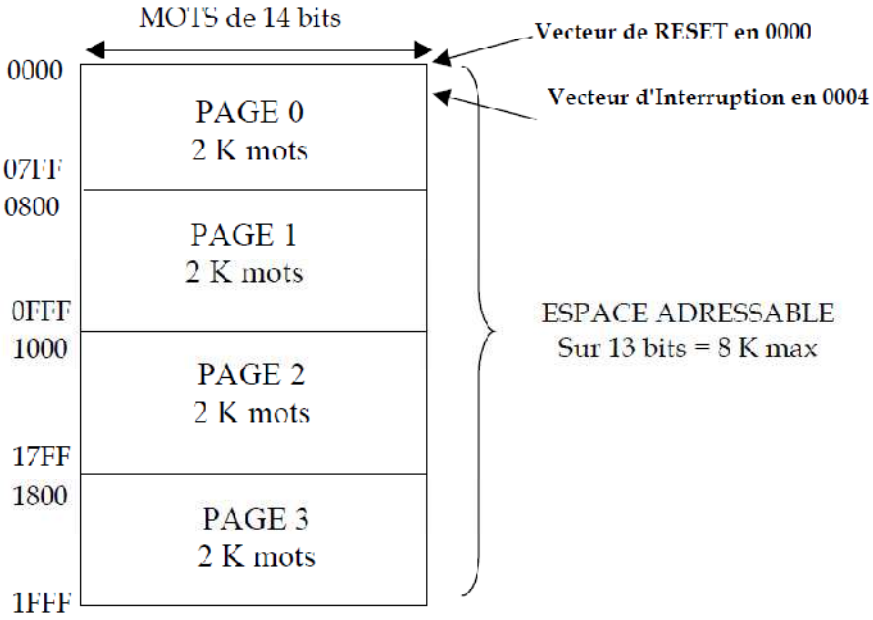

Figure 20: organisation de l'espace mémoire du 16F877

Les 2 bits MSB des 13 bits d'adresse (bits 11 et 12), viennent du registre PCLATH (bits 3 et 4) qui est à l'adresse : h'0A'. Il faut impérativement les positionner pour la bonne page, avant d'utiliser les instructions: CALL et GOTO.

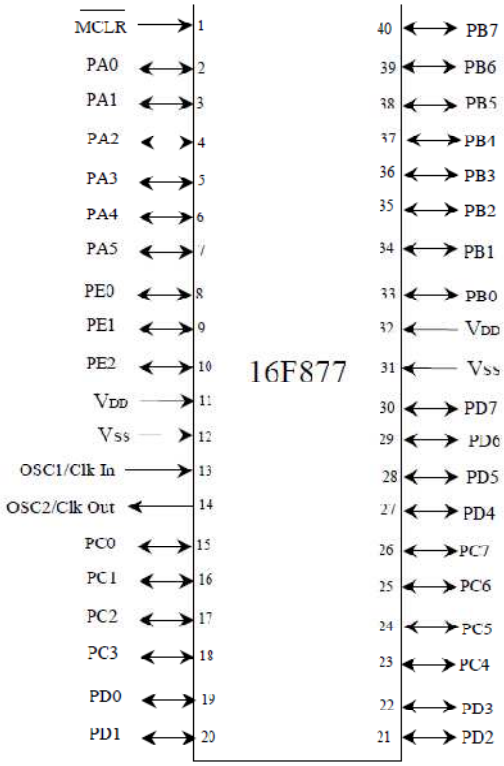

Figure 21: Brochage du 16F877

Dans la suite nous allons présenter quelques éléments de ce microcontrôleur, néanmoins pour plus de détail il faut se référer au datasheet.

6.2.3. Présentation de quelques registres internes

Un microcontrôleur présente plusieurs registres internes permettant la configuration de ce dernier, nous allons au cours de ce paragraphe présenter quelques un de ces registre afin d'avoir une idée sur leur utilité.

6.2.3.1. Le registre OPTION : (h'81' ou h'181').

Ce registre en lecture écriture permet de configurer les prédiviseurs du Timer et du Watchdog, la source du Timer, le front des interruptions et le choix du Pull up sur le Port B

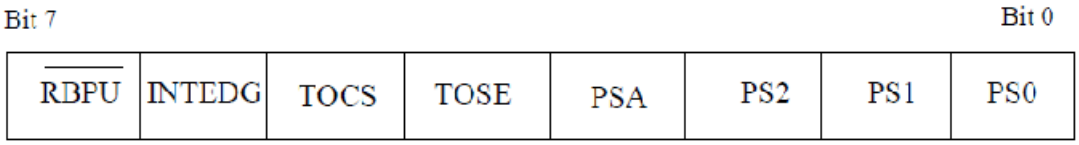

Figure 22: le registre OPTION

Au reset : OPTION = 11111111

Bit 7 : $\overline{\boldsymbol{RBPU}}$ = Pull up Enable bit on Port B.
1 = Pull up désactivé sur le Port B.
0 = Pull up activé.

Bit 6 : **INTEDG** = Interrupt Edge select bit.
1 = Interruption si front montant sur la broche PB0/IRQ (pin 6).
0 = Interruption si front descendant sur PB0/IRQ.

Bit 7 Bit 0
RBPU INTEDG TOCS TOSE PSA PS2 PS1 PS0
DOC PIC 16F876 et 16F877 D. 8 *MENESPLIER ENAC/ELE 2001*

Bit 5 : **TOCS** = Timer TMR0 Clock Source select bit.
1 = L'horloge du Timer est l'entrée PA4/Clk (pin 3).
0 = Le Timer utilise l'horloge interne du PIC.

Bit 4 : **TOSE** = Timer TMR0 Source Edge select bit.
1 = Le Timer s'incrémente à chaque front montant de la broche PA4/Clk.
0 = Le Timer s'incrémente à chaque front descendant de la broche PA4/Clk.

Bit 3 : **PSA** = Prescaler Assignement bit.
1 = Le prédiviseur est affecté au watchdog..
0 = Le prédiviseur est affecté au Timer TMR0.

Bits 2 à 0 : **PS2 PS1 PS0** = Prescaler Rate Select bits.

Quand le prédiviseur est affecté au Watchdog (PSA=1), TMR0 est prédivisé par 1.

6.2.3.2. Le registre INTCON : (h'0B' ou h'8B' ou h'10B' ou h'18B').

Ce registre en lecture écriture permet de configurer les différentes sources d'interruption.

Au reset : INTCON = 0000000X

Bit 7 : **GIE** = Global Interrup Enable bit
1 = Autorise toutes les interruptions non masquées.

0 = Désactive toutes les interruptions.

Bit 6 : **PEIE** = Peripheral Interrupt Enable bit.

1 = Autorise les interruptions causées par les périphériques.

0 = Désactive les interruptions causées par le périphériques.

Bit 5 : **TOIE** = Timer TMR0 Overflow Interrup Enable bit.

1 = Autorise les interruptions du Timer TMR0.

0 = Désactive les interruptions du Timer TMR0.

Bit 4 : **INTE** = RB0/Int Interrup Enable bit.

1 = Autorise les interruptions sur la broche : PB0/IRQ (pin6).

0 = Désactive les interruptions sur la broche : PB0/IRQ (pin6).

Bit 3 : **RBIE** = RB Port Change Interrup Enable bit.

1 = Autorise les interruptions par changement d'état du Port B (PB4 à PB7).

0 = Désactive les interruptions par changement d'état du Port B (PB4 à PB7).

Bit 2 : **TOIF** = Timer TMR0 Overflow Interrup Flag bit.

1 = Le Timer à débordé. Ce flag doit être remis à zéro par programme.

0 = Le Timer n'a pas débordé.

Bit 1 : **INTF** = RB0/Int Interrup Flag bit.

1 = Une interruption sur la broche PB0/IRQ (pin 6) est survenue.

0 = Pas d' interruption sur la broche PB0/IRQ (pin 6).

Bit 0 : **RBIF** = RB Port Change Interrup Flag bit. Ce flag doit être remis à zéro par programme.

1 = Quand au moins une entrée du port B (de PB4 à PB7) a changé d'état.

0 = Aucune entrée de PB4 à PB7 n'a changé d'état.

6.2.4. LES PORTS ENTREE / SORTIE

Les ports d'entrée / sortie numériques peuvent être considérés comme les périphériques les plus simples du microcontrôleur. Pour le PIC, on contrôle leur fonctionnement à l'aide de registres spéciaux (deux registres par port). Par exemple, pour le port A, on a le registre PORTA et le registre TRISA.

Les registres « TRISx » contrôlent le mode de fonctionnement des entrées / sorties : selon la valeur de chaque bit, 0 ou 1, la pin correspondante du port fonctionnera soit en sortie, soit en entrée. Les registres « PORTx » contiennent les données, lues, ou à écrire sur le port. Certains ports possèdent en plus des fonctionnalités spécifiques : le PORTB est équipé de résistances de tirage (pull-up) internes qui peuvent être activées ou non. La pin RB0 du PORTB sert d'entrée d'interruption externe, et les pins RB4 à RB7 peuvent générer une interruption lorsqu'un changement d'état est détecté à leurs bornes. Toutes ces particularités sont bien entendu décrites dans les documentations Microchip spécifiques à chaque composant.

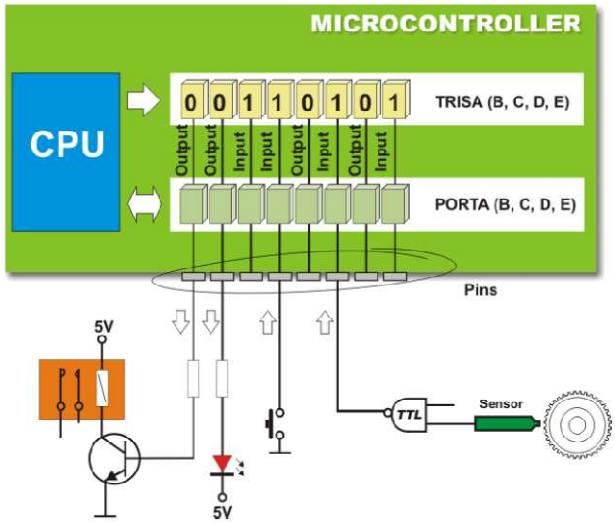

Figure 23: configuration des ports d'E/S

6.2.5. CONVERTISSEUR A/D

Il est constitué d'un module convertisseur à 8 entrées. Les 5 premières entrées sont sur le Port A en PA0, PA1, PA2, PA3 et PA5, les 3 autres sont en PE0, PE1 et PE2.

Le résultat de la conversion est codé sur 10 bits. C'est une valeur comprise entre h'000' et h'3FF'.

Les 4 registres utilisés par le module convertisseur A/D sont :
- ADRESH en h'1E' page 0 : MSB des 10 bits du résultat.
- ADRESL en h'9E' page 1 : LSB des 10 bits du résultat.
- ADCON0 en h'1F' page 0 : registre de contrôle n°0 du convertisseur.
- ADCON1 en h'9F' page 1 : registre de contrôle n°1 du convertisseur.

ADCON0 : (h'1F' : page 0)

Bit 7							Bit 0
ADSC1	ADSC0	CHS2	CHS1	CHS0	GO/$\overline{\text{Done}}$		ADON

Au reset : ADCON0 = 00000000
Bit 7 et bit 6 : **ADSC1 et ADSC0** = Clock Select bits.

Ces 2 bits permettent de choisir la vitesse de conversion :
00= Fosc/2.
01= Fosc/8.
10= Fosc/32.
11= Oscillateur RC interne.

Le temps de conversion d'un bit est TAD. Pour une conversion totale des 10 bits il faut : 12.TAD.

Bit 5 bit4 et bit 3 : **CHS2 CHS1 et CHS0** = Channel Select bits.
Ces 3 bits permettent de choisir l'entrée qui va être convertie.

Canal	CHS2	CHS1	CHS0	PORT
0	0	0	0	PA0
1	0	0	1	PA1
2	0	1	0	PA2
3	0	1	1	PA3
4	1	0	0	PA5
5	1	0	1	PE0
6	1	1	0	PE1
7	1	1	1	PE2

Bit 2 : **GO/DONE :** Status bit si ADON=1.
1 = Démarre la conversion A/D. Ce bit est remis à "0" par hard.
0 = La conversion A/D est terminée.
Bit 1 : **Bit non implanté.**
Bit 0 : **ADON :** A/D on bit.
1= Convertisseur A/D en service.
0 = Convertisseur A/D à l'arrêt.

ADCON1 : (h'9F' : page 1).
Au reset : ADCON1 = 00000000
Bit 7 : **ADFM** = A/D Result format.
1 = Justifié à droite. ADRESH ne contient que les 2 MSB du résultat. Les 6 MSB de ce registre sont lus comme des "0".
0 = Justifié à gauche. ADRESL ne contient que les 2 LSB du résultat. Les 6 LSB de ce registre sont lus comme des "0".

Bit 6 bit 5 et bit 4 : **Bits non implémentés.**
Bit 3 bit 2 bit 1 et bit 0 : **PCFG3 PCFG2 PCFG1 et PCFG0**
Bits de contrôle de la configuration des Ports.
Ces bits permettent de choisir le partage entre entrées analogiques et digitales sur les PORTS A et E.

A = Entrée Analogique.
D = I/O Digitale.

4 Bits PCFG	N°7 PE_2	N°6 PE_1	N°5 PE_0	N°4 PA_5	N°3 PA_3	N°2 PA_2	N°1 PA_1	N°0 PA_0	V_{REF^+}	V_{REF^-}
0 0 0 0	A	A	A	A	A	A	A	A	V_{DD}	V_{SS}
0 0 0 1	A	A	A	A	V_{REF^+}	A	A	A	PA_3	V_{SS}
0 0 1 0	D	D	D	A	A	A	A	A	V_{DD}	V_{SS}
0 0 1 1	D	D	D	A	V_{REF^+}	A	A	A	PA_3	V_{SS}
0 1 0 0	D	D	D	D	A	D	A	A	V_{DD}	V_{SS}
0 1 0 1	D	D	D	D	V_{REF^+}	D	A	A	PA_3	V_{SS}
0 1 1 x	D	D	D	D	D	D	D	D	V_{DD}	V_{SS}
1 0 0 0	A	A	A	A	V_{REF^+}	V_{REF^-}	A	A	PA_3	PA_2
1 0 0 1	D	D	A	A	A	A	A	A	V_{DD}	V_{SS}
1 0 1 0	D	D	A	A	V_{REF^+}	A	A	A	PA_3	V_{SS}
1 0 1 1	D	D	A	A	V_{REF^+}	V_{REF^-}	A	A	PA_3	PA_2
1 1 0 0	D	D	D	A	V_{REF^+}	V_{REF^-}	A	A	PA_3	PA_2
1 1 0 1	D	D	D	D	V_{REF^+}	V_{REF^-}	A	A	PA_3	PA_2
1 1 1 0	D	D	D	D	D	D	D	A	V_{DD}	V_{SS}
1 1 1 1	D	D	D	D	V_{REF^+}	V_{REF^-}	D	A	PA_3	PA_2

Figure 24: configuration du registre ADCON1

Au reset le registre ADCON1 est initialisé à h'00'. Cela signifie que les 5 bits du Port A et les 3 bits du Port E sont configurés en entrées analogiques.

Pour récupérer le 5 bits du Port A et les 3 bits de Port E en tant que I/O digitales il faut écrire la valeur h'06' dans ADCON1.

6.2.6. Les timers

Un timer est le nom courant de compteur / temporisateur. Il sert à :

- Mesurer du temps (compter le nombre de coup d'horloge) > **Mode temporisateur**
- Compter le nombre d'évènement sur une broche (exemple : Nombre d'appuis sur un bouton poussoir > **Mode compteur**

Il arrive souvent qu'on désire introduire des temporisations pendant l'exécution d'un programme. Le PIC 16F877 dispose de 3 timers permettant de gérer le temps avec précision. L'idée est d'initialiser une variable à une valeur donnée et ensuite la décrémenter en boucle jusqu'a ce qu'elle atteint 0. Connaissant le temps d'exécution de chaque instruction, on peut calculer le temps que mettra le processeur à terminer la boucle de décrémentation.

En pratique, on visualise la valeur de départ, puis la valeur d'arrivée. La valeur de comptage est la différence des deux valeurs.

6.2.7. LE CHIEN DE GARDE (Le Watchdog Timer WDT)

Ce dispositif est un système anti-plantage du microcontrôleur. Il s'assure qu'il n'y ait pas d'exécution prolongé d'une même suite d'instruction.

Pour le 16F877 c'est un compteur 8 bits incrémenté en permanence (même si le μC est en mode sleep) par une horloge RC intégrée indépendante de l'horloge système. Lorsqu'il déborde, (WDT TimeOut), deux situations sont possibles :

- Si le μC est en fonctionnement normal, le WDT time-out provoque un RESET. Ceci permet d'éviter de rester plante en cas de blocage du microcontrôleur par un processus indésirable non contrôle.
- Si le μC est en mode SLEEP, le WDT time-out provoque un WAKE-UP, l'exécution du programme continue normalement la ou elle s'est arrêtée avant de rentrer en mode SLEEP.

Cette situation est souvent exploitée pour réaliser des temporisations

6.3. Les étapes de réalisation d'une application à base de pic :

6.3.1. Les outils nécessaires

Pour développer une application fonctionnant à l'aide d'un microcontrôleur, il faut disposer d'un compilateur et d'un programmateur et éventuellement d'un logiciel de simulation.

Le compilateur est un logiciel traduisant un programme écrit dans un langage donné (C, basic, assembleur, etc.) en langage machine. Ce logiciel peut aussi comporter un « debugger » permettant la mise au point du programme, et un simulateur permettant de vérifier son fonctionnement, et le programmateur sert à transférer le programme du pc au pic. La simulation peut être effectuée par un logiciel indépendant du compilateur.

Donc afin de pouvoir réaliser une application à base de pic, nous devons tout d'abord commencer par écrire le programme dans un éditeur de texte indépendant ou dans l'éditeur du compilateur choisi et ce avec le langage évolué choisi (C, Pascal, basic, etc.), ensuite il faut utiliser le compilateur afin de générer le code machine, et c'est donc un fichier en .hex qui est généré.

Plusieurs compilateur existent, nous pouvons citer : mikroC, mikroBasic, mikropascal de mikroelektronika, MPLAB IDE de microchip ; *Pic* Basic *Compiler* de Selectronic etc.

Une fois nous avons notre code machine, nous avons le choix de le simuler sur un logiciel de simulation tel que ISIS ou de l'envoyer à l'aide du programmateur et de son logiciel au pic. Le programmateur permet de transférer le programme compilé (langage machine) dans la mémoire du microcontrôleur. Il est constitué d'un circuit branché sur le port COM du PC ou USB, sur lequel on implante le PIC, et d'un logiciel permettant d'assurer le transfert. Il existe différents logiciels, nous citons : Icprog, ICSP, picflash 2 usb etc.

Dans la suite nous allons utiliser le compilateur mikroC de mikroelektronika, le logiciel de simulation ISIS et le programmateur ICprog ;

6.3.2. Architecture d'un programme C pour mikroC

La saisie d'un programme en "C" répond pratiquement toujours à la même architecture. On peut noter que le symbole "#" est suivi d'une directive de compilation, le symbole "//" est suivi d'un commentaire.

```
#include<sdtio.h>          // Directive de compilation indiquant d'inclure la bibliothèque E/S standard //
#include <reg_uc.h>        //Directive de compilation indiquant d'inclure la bibliothèque spécifique au 0

#define clear=OxOO         // Directive de compilation indiquant des équivalences

char val1=0xA5;            // Déclaration d'une variable "caractère" avec valeur initiale
int val2;                  // Déclaration d'une variable "nombre entier"
.
.
.
void tempo(char temps) {   // Fonctions et procédures appelées plusieurs fois dans le programme
                           principal

}
int bintobcd(char bin) {
…

return ...;
}

void main (void)           // Programme principal
{
   DDRBA=0xFF              // initialisation et configuration

   while (1)               // Boucle principale
   {
    …
    tempo(100);
    …
   val2=bintobcd(val1);
  }
}

void nmi(void)interrupt 0
{                              // Sous programme d'interruption
```

Chaque ligne d'instruction se termine par un ";". Le début d'une séquence est précédé du symbole "{". La fin d'une séquence est suivie du symbole "}".

La notation des nombres peut se faire en décimal de façon normale ou en hexadécimal avec le préfixe "0x".

6.3.3. miKroC et exemple

Nous allons au cours de ce paragraphe présenter le compilateur mikroC, et réaliser un exemple afin de comprendre le fonctionnement.

Le compilateur *mikroC* pour *PIC* sauvegarde vos applications au sein de projets qui s'apparentent à un fichier 'projet' unique (avec l'extension .ppc) ainsi qu'à un ou plusieurs fichiers sources (avec l'extension .c). L'environnement *IDE* du compilateur *mikroC* pour PIC *ne* permet la gestion que d'un seul projet à la fois. Les fichiers sources ne peuvent être compilés que s'ils font partis d'un projet.

Un fichier projet renferme les informations suivantes:

- Nom du projet et description optionnelle;
- Composant cible;
- Options du composant (configuration word);
- Fréquence d'horloge du composant;
- Liste des fichiers sources du projet source;
- Fichiers binaires (*.mcl);
- Autres fichiers.

Dans ce paragraphe, nous allons apprendre à créer un nouveau projet, écrire un code, le compiler avec *mikroC* pour *PIC* et tester le résultat. Notre exemple permettra de faire clignoter des LEDs afin qu'il soit facile de le tester sur un microcontrôleur PIC.

Il faut donc installer le compilateur et créer un nouveau projet, le donner un nom ainsi que le chemin du dossier et choisir le pic à utiliser dans le champ device. Pour les autres réglages nous allons garder leurs valeurs par défaut. Une fenêtre vierge s'ouvre et c'est là ou nous allons écrire notre programme en C.

Le programme est le suivant :

```
void main() {
  PORTC = 0x00;              // initialisation du PORTC
  TRISC = 0;              // Configurer tout le PORTC comme sortie

  while(1) {
    PORTC =  ~PORTC;// changer la valeur du PORTC de 0 à 1 et inversement
    Delay_ms(1000);         // attendre pendant une seconde
  }
}
```

Une fois le programme écrit nous allons le compiler. Dans le dossier du projet nous allons retrouver notre code source en C dans un fichier dont l'extension est .ppc et notre fichier en code machine dont l'extension est .hex et c'est ce dernier que nous allons utiliser pour la simulation ou pour être transférer au pic via le programmateur.

Nous allons donc utiliser le logiciel PROTEUS ISIS pour simuler le programme.

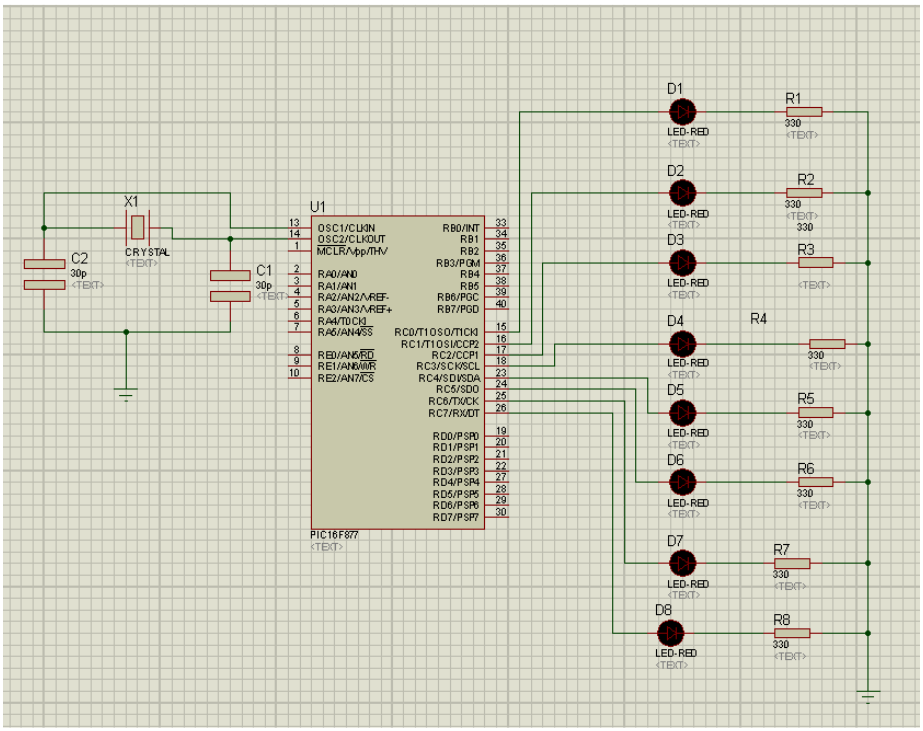

Figure 25: Schéma sous ISIS

Nous réalisons donc le schéma sous ISIS et nous éditons les propriétés du pic afin de l'indiquer le fichier .hex généré par le compilateur mikroC ainsi que la fréquence de l'horloge. Et en lançant la simulation, nous pourrons voir les LEDs s'allumer et s'éteindre toutes les secondes.

Nous pourrons par la suite transférer notre programme sur le pic via le programmateur qu'on peut aussi fabriquer nous même.

6.3.4. Quelques applications classiques :

Les microcontrôleurs peuvent être utilisés pour réaliser des applications diverses, nous pourrons trouver des applications basique dans le fichier d'aide de mikroC. Nous allons citer quelques uns dont la conception est assez simple :

- Afficher un texte sur un afficheur lcd
- Convertir un signal analogique en numérique et afficher le résultat sur des leds
- Convertir un signal analogique et afficher le résultat sur un afficheur lcd utilisant les commandes lcd.
- Envoyer un signal en utilisant la liaison série et les commandes USART