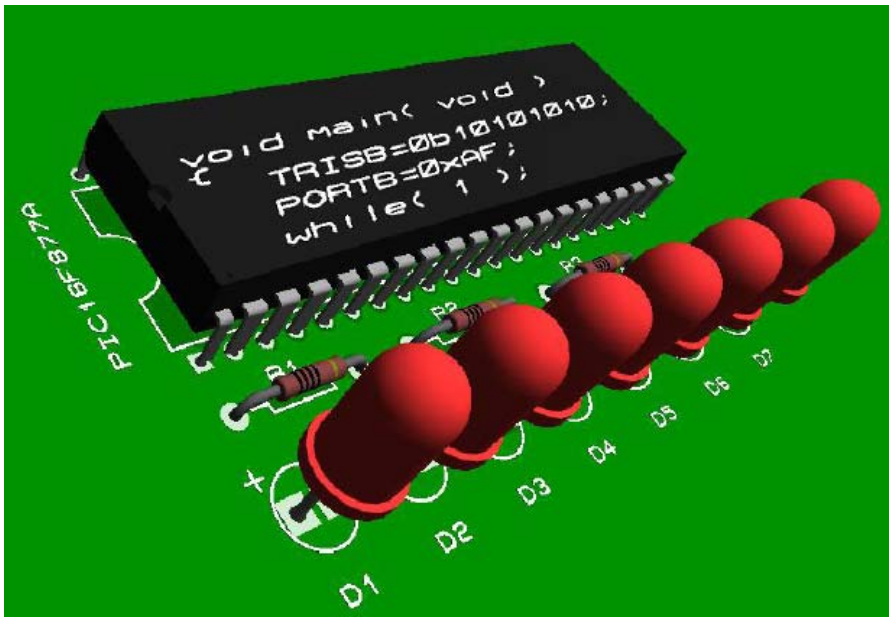
Support de cours et TP :
Pour classes BTS : 1ELT

Conception et simulation des systèmes à PIC



A.TAOUNI
Lycée Alkhaouarizmy
CASABLANCA

Programmation avec MikroC PRO
Simulation en Proteus ISIS

Table des matières

1. Le compilateur MikroC et les PICmicro

Un microcontrôleur est un circuit électronique encapsulé dans un circuit de haut niveau d'intégration. Les microcontrôleurs sont commercialisés par différents fabricants comme Freescale, Motorola, Intel, Philips, et Microchip.

Microchip en particulier, est un fabricant des circuits électroniques. Dans leurs lignes de production, on trouve les microcontrôleurs PICmicro, qui sont disponibles dans des différentes familles, certaines d'entre elles sont: 12F, 16F, 18F, 24F, 30F, 33F etc.…

Le promoteur du projet choisit la famille et la référence qui répondent à ses besoins, comme un PIC 12F675 ; microcontrôleur 8 broches avec des modules intégrés de base tels que le Timer et l'ADC. Un microcontrôleur comme le 16F877 dispose de 40 broches et des modules tels que: Timer, ADC, USART, I2C, PWM, entre autres. On peut facilement voir la différence entre ces deux PIC.

Figure 1-1

La documentation technique, la commercialisation massive et l'incroyable quantité d'informations publiées sur les microcontrôleurs PIC, les rendent idéals pour l'enseignement. L'entreprise Microchip a le portail web www.microchip.com où on peut télécharger des informations et des applications logiciels qui facilitent le développement avec ses microcontrôleurs.

Fondamentalement, le développement d'une application à base de microcontrôleur PIC consiste à identifier les problèmes de développement, éditer, déboguer le programme machine et programmer le microcontrôleur avec un programmateur spécifique pour PICmicro. Microchip fournit des programmateurs spécialisés dans différentes échelles, peut-être le plus populaire est le PICSTART plus, mais il ya d'autres comme PICkit2, PICkit3. Bien qu'il existe des programmateurs commerciaux un développeur peut construire ou acheter un programmateur éducatif à faible coût, pour cela, il y a amplement d'information sur Internet.

Figure 1-2

Un microcontrôleur a une architecture de base similaire à celle d'un PC ,dispose d'un bloc OTP ou mémoire Flash dans lequel sont stockées les instructions du programme, cette section est similaire au disque dur de l'ordinateur, le PICmicro a une RAM, qui joue le même rôle de la mémoire vive d'un ordinateur personnel, le microcontrôleur possède des ports d'entrée et de sortie qui sont similaires aux périphériques d'entrée et de sortie pour ordinateur tel que souris, une imprimante, un écran, un clavier, et autres.

Ces caractéristiques du microcontrôleur le rendent idéal pour la création de petites applications qui ont une interface utilisateur, comme des claviers, des boutons, des lecteurs de la mémoire de stockage de masse, et des capteurs de différentes grandeurs telles que la température, l'humidité, la pression, la lumière, la proximité, etc. De même, il est possible de créer des environnements de visualisation avec des afficheurs numériques, alphanumériques et graphiques. Les ports série comme USART et l'USB peuvent créer des communications série et des communications sans fil avec d'autres appareils. En bref, les possibilités sont sans fin.

1.1 Le compilateur MikroC PRO

La programmation des microcontrôleurs est basée sur le code machine, qui est connu comme code assembleur, ce code contient les instructions du programme, le code assembleur est bien détaillé et difficile à écrire. Le programme en code assembleur est très long et difficile à comprendre. La création des compilateurs de haut niveau a rendu facile la création et l'édition de programmes, bien entendu les microcontrôleurs ne font pas exception. Dans le commerce, il ya plusieurs variétés de compilateurs des différents fabricants et avec différents langages de haut niveau.

On peut acheter les compilateurs PICC, CCS, PIC Basic, entre autres. L'étude de ce document se concentre sur le compilateur MikroC PRO, qui est un compilateur de langage C pour les microcontrôleurs PICmicro famille 12F, 16F, 18F et.

MikroC PRO est un logiciel avec une grande variété des helps, supports et d'outils, qui facilite la création des projets et des applications pour les microcontrôleurs PICmicro.

L'étude de cet environnement de développement est possible, parce que l'étudiant peut télécharger une démo ou version étudiant, qui a les mêmes caractéristiques que la version complète, la seule limite est la taille du code machine ne peut pas excéder 2Koctets, une capacité suffisante pour un premier apprentissage. La version de démonstration peut être téléchargée à partir du site Web: www.mikroe.com.

La figure suivante montre l'aspect visuel de l'environnement de développement.

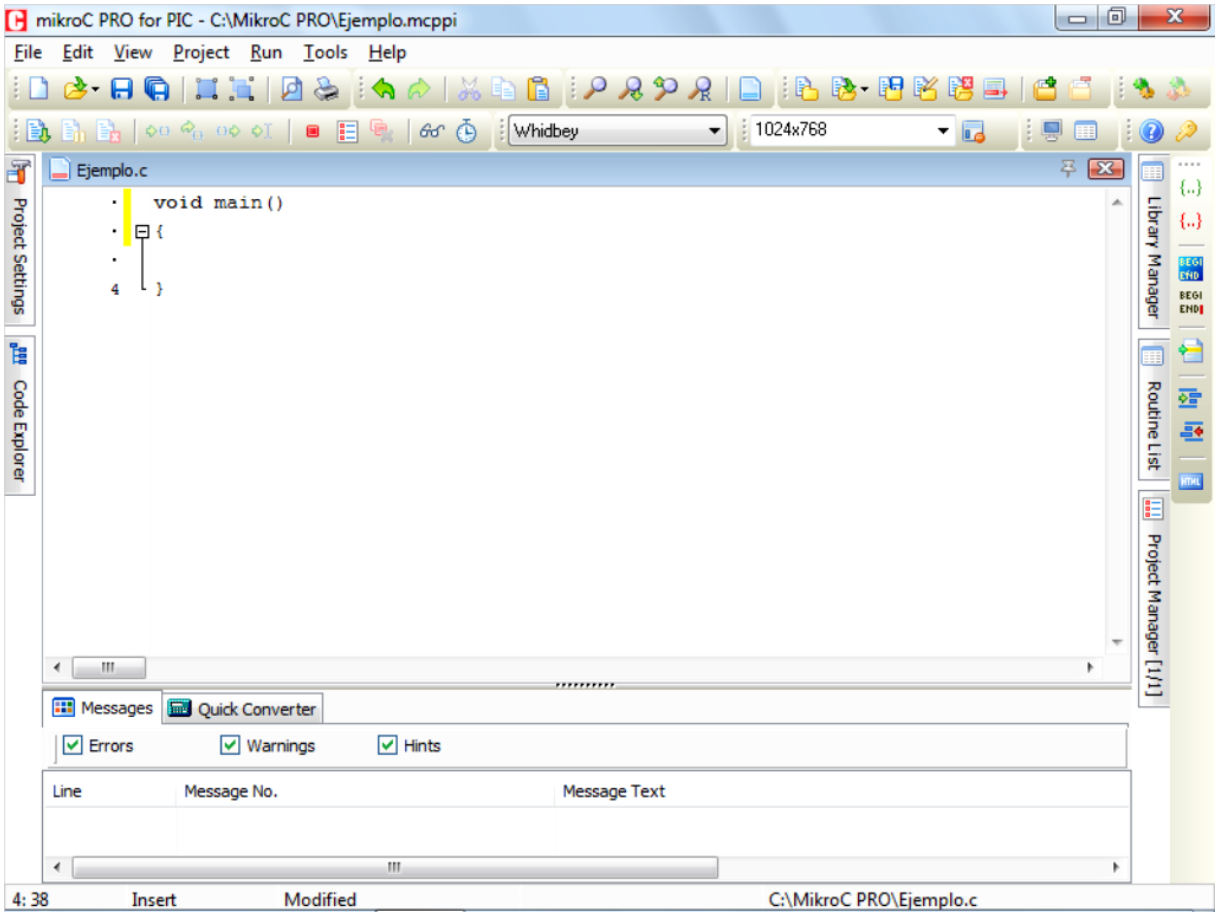


Figure 1-3

Le compilateur de haut niveau en langage C utilise des structures, facilitant la programmation, optimisant les opérations mathématiques, et les processus grâce à l'utilisation de fonctions prédéfinies et des fonctions que le développeur peut créer et l'utilisation d'un ensemble de variables, caractère, nombre entier, et réel. Le compilateur crée automatiquement le code assembleur et un code enregistré dans un fichier avec l'extension *. Hex, ce fichier est le résultat principal de la compilation, avec lequel le microcontrôleur sera programmé électriquement ou pour être utilisé pour une simulation sur ordinateur.

2. Principes de langage C

Le langage C datant de 1972, a été créé par les laboratoires Bell en raison de la nécessité de réécrire les systèmes d'exploitation UNIX afin d'optimiser le code assembleur. De la même manière, le langage C a été l'évolution des langages précédents appelé B et BCPL. Le nouveau langage C, a rapidement pris la main pour ses fonctionnalités et sa facilité à la mise en œuvre dans des différents systèmes de calcul.

La forme du langage C, est basée sur une structure complexe, nécessitant une étude approfondie. Pour la programmation des microcontrôleurs, l'étudiant n'a besoin que des notions fondamentales qui le permettent de se lancer et de créer les premiers projets en MikroC PRO. La fin ce chapitre est axé sur la connaissance des notions de base pour commencer à étudier les microcontrôleurs avec ce document.

2.1 Déclaration des variables en C

Les variables de base dans ce compilateur spécifique sont:

- *bit*
- *char*
- *short*
- *int*
- *long*
- *float*
- *double*

Les variables *bit* peuvent stocker un booléen qui peut être vrai ou faux, 0 ou 1.

Les variables *char* sont utilisées pour stocker des caractères codés avec le code ASCII. Il est utile pour stocker des lettres ou des textes.

Une variable *short* contient un entier 8 bits qui peut prendre des valeurs : -127 à 127.

Les variables de type *int* sont des entiers 16 bits, cette variable peut stocker des nombres : -32767 à 32767.

La variable type *long* stocke un entier sur 32 bits, sa plage de valeurs est:

-2147483647 à 2147483647.

Les variables type *double* et *float* permettent de stocker des nombres avec virgule.

Les variables ci-dessus peuvent être déclarées, y compris le signe positif et négatif, ou peuvent être déclarées au moyen de la directive *unsigned*.

Dans le tableau ci-dessous, on peut voir les caractéristiques des variables.

Type de variable	*Taille en octets*	*Valeurs*
Bit	*1*	*0 ou 1*
Char	*1*	*-127 à 127*
Short	*1*	*-127 à 127*
Int	*2*	*- 32767 à 32767*
Long	*4*	*- 2147483647 à 2147483647*
Float	*4*	*-1.5x10^45 à 3.4x10^38*
Double	*4*	*-1.5x10^45 à 3.4x10^38*
unsigned char	*1*	*0 à 255*
unsigned short	*1*	*0 à 255*
unsigned int	*2*	*0 à 65535*
unsigned long	*4*	*0 à 4294967295*

Tableau 2.1

La déclaration des variables s'effectue en indiquant le type de la variable suivi d'un nom que le développeur attribue arbitrairement à la variable. Au moment de la déclaration d'une variable, il est possible de lui donner une valeur initiale, mais ceci n'est pas strictement nécessaire. Enfin, la déclaration doit se terminer par un point-virgule (;).

Dans les exemples suivants, on peut voir comment les déclarations sont faites:

```
bit VARIABLE1_BIT;                  //Déclaration d'une variable type bit.
char CARACTERE;                     // Déclaration d'une variable type char.
char CARACTERE2='J';                //Déclaration d'une variable de type char initialisée
                                    //avec la valeur ASCII le caractère J.
int ENTIER=1234;                    // Déclaration d'une variable entière initialisée avec
                                    //la valeur 1234.
float DECIMAL=-12.45;               // Déclaration d'une variable avec un point virgule
                                    //initialisée avec la valeur -12,45.
double DECIMAL2=56.68;              // Déclaration d'une variable avec un point virgule
                                    //initialisée avec la valeur 56,68.
long ENTIER2=-954261;               // Déclaration d'une variable entière de type long
                                    // initialisée avec la valeur -954261.
```

Les exemples suivants montrent comment déclarer des variables non signées:

```
unsigned char CARACTERE;            //Déclaration d'une variable type char non signé.
unsigned int ENTIER;                //Déclaration d'une variable type entier non signé.
unsigned long ENTIER2;              // Déclaration d'une variable type entier long non
                                    //signé.
```

Les variables peuvent également être déclarées dans un format qui associe plusieurs variables dans un même nombre, ce format est connu comme une chaine de variables ou un vecteur et peut-être même un tableau de variables, en conclusion ce genre de déclaration peut avoir une ou plusieurs dimensions.

L'exemple suivant montre un vecteur de caractère, ou également connu sous le nom chaîne de caractères:

```
char Text[20];                      //Chaîne avec 20 emplacements de mémoire.
```

De même des chaînes de variables peuvent être déclarées avec des valeurs initiales, ce genre de déclaration peut être vu dans l'exemple suivant:

```
char Text[20] = "Nouveau Texte";        //Déclarer une chaîne
                                        //initialisée avec le texte: Nouveau texte.
int Eniers[5]={5,4,3,2,1};               // Déclarer une chaîne d'entiers avec des
                                        //valeurs initiales.
float Decimals[3]={0.8,1.5,5.8};        // Déclaration d'une chaîne de nombres
                                        //décimaux initialisée....
```

La déclaration des variables doit respecter quelques règles de base pour éviter les erreurs et les incohérences dans la compilation de code, à cet effet il faut tenir compte des conseils suivants:

- Les noms des variables ne doivent pas être répétés.
- Les variables ne doivent pas commencer par un chiffre.
- Une variable ne peut pas utiliser des caractères spéciaux comme: / * ‘;} {- \! . %.

Des exemples de déclarations de variables qu'on ne peut pas faire, c'est:

```
bit 1_VARIABLE-;
char -CARÁCTERE!;
int 3ENTIER*;
```

De même, on peut créer des structures d'informations comme une nouvelle variable créée par le développeur. Ces structures sont réalisées avec les variables prédéfinies et peuvent être des déclarations des variables ou des variables tableaux. De telles structures sont déclarées par l'intermédiaire de la directive: *typedef struct*, et la façon de déclarer une variable créée par le développeur est la suivante:

```
typedef struct
{
char Nom[10];
int age;
}Utilisateur;
```

Voici comment utiliser une variable personnalisée par le développeur:

```
Utilisateur U;
U.age = 25;
U.Nom[0]='A';
U.Nom[1]='h';
U.Nom[2]='m';
U.Nom[3]='e';
U.Nom[4]='d';
```

2.2 Les formats numériques dans le langage C :

Les applications en langage C utilisent des nombres de différentes bases de numération, alors pour le fonctionnement des microcontrôleurs en bas niveau, tous les nombres sont convertis en base *2* (deux nombres binaires *0* et *1*). L'être humain n'est pas habitué à penser et à traiter des opérations en cette base numérique.

Dès les premières étapes de la formation académique les écoles et les collèges apprennent aux étudiants à penser et à traiter tous les calculs en base *10*, c'est à dire avec des nombres décimaux. C'est pourquoi les compilateurs du langage C travaillent avec des nombres décimaux facilitant les conceptions pour le développeur. Outre le système décimal, le compilateur du langage C peut

travailler avec les autres bases telles que binaire, hexadécimal et rend plus facile à effectuer des calculs et des tâches qui seraient plus complexe en décimal.
Les systèmes de numération en base *2*, *10*, et *16*, sont implémentés dans le compilateur du langage C. L'écriture des nombres décimaux, est la forme la plus simple, ils sont écrits en langage C de la même manière conventionnelle qu'on a appris dès les premiers cours des mathématiques. Les nombres binaires sont écrits avec le préfixe *0b* suivi par le nombre binaire. Un exemple d'écriture est: *0b10100001* qui est équivalent au nombre décimal *161*.

Les nombres en base *16* ou hexadécimal sont désignées avec *0x* en entête suivi d'un nombre en hexadécimal, l'expression *0x2A*, est un exemple d'un nombre hexadécimal en langage C, qui est équivalent à *42* en décimal. Les nombres binaires ne peuvent avoir que deux chiffres sont *0 et 1*. Les nombres hexadécimaux peuvent comprendre des chiffres parmi les suivants: *0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E et F*.

2.3 Les opérateurs en langage C

Le langage C permet de faire des opérations arithmétiques de base entre des variables ou des constantes. Les opérations arithmétiques disponibles sont:

- *Somme*
- *soustraction*
- *Multiplication*
- *Division*
- *Modulo*

L'exemple suivant montre la somme arithmétique de deux nombres ou plus:

```
int A;
int B;
int C;
C = A+B;                 //cet expression stocke la somme de A et B dans la variable C.
C = A+B+C;               //cet expression stocke la somme de A, B et C dans
                         //la variable C.
```

L'exemple suivant montre la soustraction arithmétique entre deux nombres ou plus:

```
int A;
int B;
int C;
C = A-B;                 //cet expression stocke la différence de A et B dans
                         //la variable C.
C = A-B-C;               //cet expression stocke la différence de  A, B et C dans la
                                 //variable C.
```

L'opération mathématique de multiplication peut être effectuée entre deux ou plusieurs variables, cet opération est réalisée avec l'astérisque (*), cette expression peut être vue plus clairement dans les exemples suivants:

```
int A;
int B;
int C;
C = A*B;                 //cet expression stocke la multiplication entre A et B dans la
                         //variable C.
C = A*B*C;               //cet expression stocke la multiplication entre A, B et C dans
                         la //variable C.
```

La division arithmétique en langage C est spécifiée par la barre oblique (*/*), dans l'exemple suivant, on peut voir sa mise en œuvre:

```
int A;
int B;
int C;
C = A/B;                    //cet expression stocke la division entre A et B dans
                            //la variable C.
```

L'opération modulo calcule le reste d'une division arithmétique, le calcul du modulo est indiqué par le caractère pour cent (*%*), l'application de cette opération peut être vue dans l'exemple suivant:

```
C = A%B;                    //cet expression stocke le reste de la division entre A et B dans
                            //la //variable C.
```

Les opérations arithmétiques peuvent être utilisées en combinaison, c'est à dire que on peut mélanger plusieurs opérations en une seule expression, pour voir plus clairement on va observer les exemples suivants:

```
int A;
int B;
int C;
C = (A+B)/C;                //cet expression est équivalente à C = (A+B) ÷C.
C = (A/B)*C;                //cet expression est équivalente à C = (A÷B) X C.
```

D'autres opérateurs mathématiques raccourcis peuvent être utilisés lorsqu'on veut une modification constante des variables, par exemple, il est possible d'incrémenter ou de décrémenter une variable d'un certain nombre, tout comme il est possible de faire cette opération avec la multiplication et la division. Pour comprendre ce concept, on observe plus clairement les exemples suivants:

```
A++;                        //Cet opérateur incrémente de un la valeur de A.
A--;                        //Cet opérateur décrémente de un la valeur de A.
A+=4;                       //Cet opérateur incrémente de 4 la valeur de A.
A-=5;                       //Cet opérateur décrémente de 5 la valeur de A.
A/=4;                       //Cet opérateur divise la valeur de A par 4.
A*=3;                       //Cet opérateur multiplie la valeur de A par 3.
A+=B;                       //Cet opérateur incrémente de B la valeur de A.
A*=B;                       //Cet opérateur multiplie la valeur de A par B.
```

Autres opérations mathématiques plus complexes peuvent être effectuées en langage C, à l'aide des fonctions prédéfinies dans la bibliothèque mathématiques. Ce sujet sera traité plus tard, quand les déclarations des fonctions seront étudiées.

Les opérateurs logiques permettent d'effectuer des actions ou des opérations qui respectent la logique numérique soulevée par le mathématicien anglais *George Boole* au *XIXe* siècle. *Boole* a posé les opérations booléennes *OU, ET, NON, XOR, NOR, NAND, XNOR,* qui sont largement étudiées dans les cours de la logique combinatoire et systèmes numériques séquentielles.

Les opérations logiques sont effectuées en langage C entre deux variables ou des constantes, bit à bit de même poids dans une variable ou un nombre. Pour voir et comprendre les exemples, on rappelle d'abord les tables de vérité des opérations logiques comme indiqué ci-dessous:

Inverseur NOT	
Entrée	**Sortie**
0	1
1	0

OR inclusive		
Entrée 1	**Entrée 2**	**Sortie**
0	0	0
0	1	1
1	0	1
1	1	1

NOR inclusive		
Entrée 1	**Entrée 2**	**Sortie**
0	0	1
0	1	0
1	0	0
1	1	0

OR exclusive ou XOR		
Entrée 1	**Entrée 2**	**Sortie**
0	0	0
0	1	1
1	0	1
1	1	0

NOR exclusive ou XNOR		
Entrée 1	**Entrée 2**	**Sortie**
0	0	1
0	1	0
1	0	0
1	1	1

AND		
Entrée 1	**Entrée 2**	**Sortie**
0	0	0
0	1	0
1	0	0
1	1	1

NAND		
Entrée 1	**Entrée 2**	**Sortie**
0	0	1
0	1	1
1	0	1
1	1	0

Les opérations logiques en C, sont réalisées avec des caractères spéciaux qui désignent chacun. Dans l'exemple, on peut voir des applications de ces opérateurs logiques:

- Opération NON logique, ou complément, désigné par le caractère tilde (~);

```
unsigned short VALEUR1 = 0b01010000      // variable initialisée au  nombre binaire 80.
unsigned short  RESULTAT;
RESULTAT = ~ VALEUR1                     // La variable RESULTAT contient le complément de
                                         //VALEUR1 , 175.
```

- Opération logique OU, OU inclusive, est désignée par le caractère barre verticale (|);

```
unsigned short VALEUR1 = 0b01010000      // variable initialisée en binaire avec le nombre 80.
unsigned short VALEUR2 = 0b01011111      // variable initialisée en binaire avec le nombre 95.
unsigned short RESULTAT;
RESULTAT = VALEUR1 | VALEUR2             // La valeur de l'opération logique ou est stockée dans
                                         //la variable RESULTAT, 95.
```

- Opération XOR, ou exclusif, est notée avec l'accent circonflexe (^);

```
unsigned short VALEUR1 = 0b01010000      // variable initialisée en binaire avec le nombre 80.
unsigned short VALEUR2 = 0b01011111      // variable initialisée en binaire avec le nombre 95.
unsigned short RESULTAT;
RESULTAT = VALEUR1^ VALEUR2              // La valeur de l'opération logique ET
                                         //est stockée dans RESULTAT, 15.
```

- ET logique est notée avec l'esperluette (&);

```
unsigned short VALEUR1 = 0b01010000      // variable initialisée en binaire avec le nombre 80.
unsigned short VALEUR2 = 0b01011111      // variable initialisée en binaire avec le nombre 95.
unsigned short  RESULTAT;
RÉSULTAT = VALEUR1 &VALEUR2;             // La valeur de l'opération logique ET
                                         // est stockée dans le RESULTAT, 80.
```

- La mise en œuvre de NAND, NOR est XNOR, sont semblables aux opérations logiques ET, OU, XOR, avec l'ajout du caractère tilde (~) de la complémentation, on observe les exemples:

```
unsigned short VALEUR1 = 0b01010000   // variable initialisée en binaire avec le nombre 80.
unsigned short VALEUR2 = 0b01011111   // variable initialisée en binaire avec le nombre 95.
unsigned short RESULTAT;
RESULTAT = ~(VALEUR1|VALEUR2)         // La valeur de l'opération logique NOR est stockée
                                      //dans RESULTAT.
RESULTAT =~(VALEUR1 & VALEUR2)        // La valeur de l'opération logique  NAND est stockée
                                      //dans RESULTAT.
RESULTAT =~(VALEUR1^ VALEUR2 )        // La valeur de l'opération logique XNOR est stockée
                                      //dans RESULTAT.
```

- Le décalage des bits d'une variable binaire est utile pour des procédés et des tâches impliquant la manipulation de données au niveau des bits. Le décalage à droite d'une variable ou d'une constante, en langage C, est réalisé par double caractère supérieur à (>>), de la même manière pour le décalage à gauche qui est réalisé par (< <). L'opération de décalage fait perdre des bits et insère des zéros dans les nouveaux bits. Dans les exemples suivants, on voit la mise en œuvre de ces opérateurs:

```
short Donnee=0xFF                     // Déclaration des variables.
short RESULTAT;
RESULTAT = Donnee>> 4                 //cet opération stocke dans la variable RESULTAT le
                                      //résultat de décalage à droite de 4 bits la variable
                                      //Donnee, la valeur du résultat final est 0x0F.
RESULTAT= Donnee << 4,                //cet opération stocke dans la variable RESULTAT le
                                      //résultat de décalage à gauche de 4 bits de la
                                      variable //Donnee, la valeur du résultat final est 0xF0.
```

Les opérations logiques et arithmétiques mentionnées ci-dessus sont résumées dans le tableau suivant:

Description de l'opération	*Opérateur*
Somme	+
Reste	-
Division	/
Modulo ou reste de la division entière	%
Incrémenter	++
Décrémenter	--
Opération OU	|
Opération ET	&
Opération Ou exclusive	^
Complément à 1	~
Décalage à droite	>>
Décalage à gauche	<<

Tableau 2-2

2.4 Les fonctions en langage C :

Une fonction est une partie du code qui effectue une tâche spécifique chaque fois qu'elle est invoquée par le flux du programme principal. Les fonctions ont deux caractéristiques essentielles, une ou plusieurs paramètres d'entrée et un paramètre de sortie. Ces paramètres peuvent être des variables ou des tableaux, tout comme ces paramètres d'entrée et de sortie peuvent être vides.
La structure des fonctions peut être vue dans les exemples suivants:

```
void nom_fonction (void)          // Fonction avec des paramètres d'entrée et de sortie
                                  //vides.
{                                 // Début de la fonction avec accolade.
                                  // Code où la routine de fonction à exécuter.
}                                 // Fin de la fonction avec accolade.

void nom_fonction(int A)          // Fonction avec un paramètre d'entrée A entier et de
                                  //sortie vide.
{                                 // Début de la fonction
                                  // Code où la routine de fonction à exécuter.
}                                 // Fin de la fonction.

int nom_fonction(void)            // Fonction avec paramètre d'entrée vide et de sortie
                                  //entier.
{                                 // Début de la fonction.
                                  // Code où la routine de fonction à exécuter.
}                                 // Fermeture de la fonction.
int nom_fonction(int A)           // Fonction avec un paramètre d'entrée et de
                                  sortie //tout entiers.
{                                 // Début de la fonction.
                                  //Code où la routine de fonction à exécuter.
}                                 // Fermeture de la fonction.
```

Les noms désignant les fonctions répondent aux mêmes règles de dénomination de variables.
L'exemple suivant montre une fonction qui est capable de calculer le produit de deux nombres avec virgule:

```
float produit(float A, float B)   // Fonction pour calculer le produit de A et B.
{                                 //Début de la fonction.
float RESULTAT;
RESULTAT=A*B                      //Produit de A et B.
return RESULTAT;                  //La fonction retourne le résultat stocké dans
                                  //RESULTAT.
}                                 //Fermeture de la fonction.
```

Une fonction peut utiliser une autre fonction pour exécuter des fonctions plus complexes. Pour voir cette situation, on voit l'exemple suivant, qui calcule l'aire d'un cercle en fonction de son rayon:

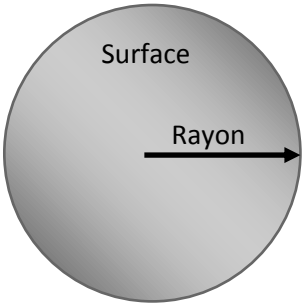


Figure 2-1

L'aire du cercle est calculé par l'équation A = πr ², où la valeur de π est d'environ 3,1416.

```
float Valeur_PI ( void )          //Fonction qui retourne la valeur de  π.
{                                 //Début de la fonction
float PI=3.1416;
return PI;
}                                 //Fin de la fonction
```

Dans le cas où une fonction utilise des variables locales, la déclaration doit être faite au début de celui-ci, avant toute autre action ou instruction.

```
float Aire_Cercle(float Rayon)    //Fonction pour calculer l'aire d'un cercle.
{                                  //Début de la fonction
float Aire;
Aire= Valeur_PI()*Rayon*Rayon;    //Calcul de l'aire d'un cercle.
return Aire;
}                                 //fin de la fonction
```

Les fonctions sont créés par le développeur, mais il y en a certaines qui sont prédéfinies dans le compilateur et peuvent être consultées et mises en œuvre en respectant les paramètres d'entrée et de sortie, la plus importante d'entre elles est la fonction *main()* ou principale. L'importance de cette fonction réside dans le fait que c'est sur elle que court tout le programme du microcontrôleur. Elle est automatiquement exécutée quand le microcontrôleur est alimenté. La fonction *main()* dans le cas particulier des microcontrôleurs ne contient ni paramètres d'entrée ou ni de sortie, sa déclaration peut être vue dans l'exemple suivant:

```
void main( void )                 //Déclaration de la fonction main.
{                                 // Début de la fonction main.
                                  // Code pour le programme principal du
                                  //microcontrôleur.
}                                 // fin de la fonction main.
```

Les déclarations des fonctions, écrites par le développeur, doivent être déclarées avant la fonction *main()*.

Dans le cas d'une fonction qui appelle une autre fonction, la fonction appelée doit être déclarée avant la fonction qui fait l'appel.

Les fonctions trigonométriques et d'autres fonctions mathématiques telles que logarithmes, et exponentielle, peuvent être utilisées avec la bibliothèque prédéfinie par le compilateur. Cette bibliothèque est appelée *C_Math.h.*

2.5 Création d'un programme en langage C

La structure d'un programme en langage C est relativement simple, il est d'abord essentiel de déclarer des variables globales qui on juge nécessaires pour le fonctionnement du programme. Les variables globales sont reconnues par toutes les parties du programme, y compris les fonctions propres au développeur et la fonction *main()*.

La prochaine étape est de faire des déclarations de fonctions conçues par le développeur pour des tâches spécifiques dans le programme. Par la suite, la fonction *main()* est déclarée, au début on doit déclarer des variables qui sont nécessaires au sein de la même fonction *main()*. Le code ci-dessous doit configurer et initialiser les ports du microcontrôleur et les modules qui sont essentiels à l'application. Enfin, on écrit le code qui contient le programme de l'application utilisateur. Dans l'exemple ci-dessous, on peut voir la structure d'un programme:

```
int Variable1;                          //Déclaration des variables globales.
char Variable2;
float Variable3;
                                        //Déclaration de fonctions propres au développeur.
float Valeur_PI ( void )
{
float PI=3.1416;
return PI;
}

float Calcul_Aire_Cercle( float Rayon )
{
float Aire;
Aire = Valeur_PI()*Rayon*Rayon;         //Calcul de l'aire.
return Aire;
}
void main( void )                       //Déclaration de la fonction main ou principale
{
float Valeur_Rayon;                     //Déclaration des variables de la fonction main.
float Valeur_Aire;
TRISB=0;                                //Configuration de ports et des modules.
TRISC=0;
PORTB=0;
PORTC=0;
while( 1 )                              //Code de l'application.
{
Valeur_Aire=Calcul_Aire_Cercle( Valeur_Rayon );
}
}
```

2.6 Formes conditionnelles et itération en langage C

Les formes conditionnelles et itératives font partie intégrante de la structure de n'importe quel code en langage C. Les instructions conditionnelles *if, if else,* et *switch case* permettent de réaliser des menus, prendre des décisions, et de contrôler le flux du programme ou de l'application. Les structures itératives *while, do while* et *for,* permettent de créer des boucles itératives, qui régissent des tâches et des processus développés dans le programme.

2.6.1 Création des conditions logiques en C

Les conditions logiques permettent de prendre des décisions en fonction d'un test. Les conditions logiques retournent seulement deux valeurs possibles: vrai ou faux. Lorsque la condition logique se fait directement sur une variable ou un nombre, la condition est fausse lorsque sa valeur est 0 et renvoie vrai pour une valeur différente 0.

Les formes conditionnelles utilisées dans le langage C sont : NOT, ou la négation, OR ou ou inclusif , AND , ou et . On peut également utiliser des comparaisons entre deux valeurs telles que supérieur à, inférieur à ou supérieur ou égal à, inférieur ou égal à, différents de, et égal à. L'utilisation de la négation se fait avec le caractère d'exclamation(!), la condition OU est effectuée avec les caractères double barres verticales (||) , La condition AND ou et, est utilisée avec double esperluette (&&), la condition supérieur à est utilisée avec le caractère (>) , la condition inférieur à est utilisée avec le caractère (<) , supérieur ou égal à est faite avec les caractères (>=) , l'état inférieur ou égal à , se fait avec (<=) , l'égalité est réalisée avec les caractères double égal (==) et l'état différent est utilise les caractères de l'égalité et d'exclamation (!=) . Pour bien comprendre ces opérateurs, on observe et on analyse les exemples suivants :

```
short A;
short B;
(A==10)&&(B==20)          //La condition est vraie si A vaut 10 et B vaut 20.
(A!=5)&&(B>2)             // La condition est vraie si A es différent de 5 et B est
                          //supérieur à 2.
(A<=4)&&(B>=6)            // La condition est vrai si A est inférieur ou égal  4 et
                          si //B est supérieur ou égal  6.
!(A>3)&&(B!=4)            // La condition est vrai si A n'est pas supérieur 3 et si
                          B //est différente de 4.
```

2.6.2 L'instruction conditionnelle if et if else

L'instruction *if* évalue une condition logique et exécute un partie de code si et seulement si le résultat de la condition est vrai. On observe l'instruction *if* dans l'exemple suivant:

```
short Valeur;
if( Valeur>100 )          //La condition est vraie  si la variable Valeur est
                          //supérieure à 100.
{
                          // Partie de code qui est exécutée si la condition est
                          //vraie.
}
```

La structure *if else* est la même que *if* , la seule différence est que dans le cas où la condition est fausse, c'est la partie de code associée à *else* qui est exécutée. Pour comprendre cette situation, on prend l'exemple suivant:

```
short Valeur;
if( Valeur>100 )          //La condition est vraie si la variable Valeur est
supérieure à 100.
{
                          // Partie de code qui est exécutée si la condition est
                          //vraie.
}
else
{
```

```
                                        // Partie de code qui est exécutée si la condition est
                                        //fausse.
}
```

Le *if* et *if else* peuvent être imbriqués pour créer des structures plus complètes et complexes, ce cas peut être vu clairement dans l'exemple suivant:

```
short Valeur1;                          //Déclaration des variables.
short Valeur2;
if( Valeur1>30 )                        //test de if.
{
                                        // Partie de code qui est exécutée si la condition est
                                        //vraie.
if( (Valeur2==4)&&(Valeur1<50) )        //Seconde if  imbriquée.
{
                                        // Code qui s'exécute lorsque la seconde if a une
                                        //condition vraie.
}
}
```

2.6.3 L'instruction switch case

L'instruction *switch case*, fonctionne d'une manière similaire à l'instruction *if*, mais la différence c'est qu'on n'évalue pas une condition mais la valeur d'une variable, et qu'on exécute une partie de code pour chaque valeur possible de la variable.
Les valeurs écrites dans les *case* sont les cas possibles de la variable, le développeur les choisit en fonction du besoin. Pour les cas des valeurs qui n'ont pas d'intérêt pour le développeur, une partie de code s'exécute par défaut, en utilisant la directive *default*.

Chacun des fragments du code édités doit se terminer par l'instruction *break* pour rompre le flux de la structure *switch case*, cet action est nécessaire parce que si deux cas se suivent les fragments de codes correspondants vont s'exécuter jusqu'à la directive *break* ou jusqu'à la fin la structure *switch case*.
Pour bien comprendre le fonctionnement de l'instruction *switch* prenons l'exemple suivant:

```
short Valeur;
switch( Valeur ) //Evaluation de la variable Valeur.
{
case 0:          // Fragment de code correspondant à la variable Valeur égale 0 .
                 break;          //Rupture du cas 0.
case 1:          // Fragment de code correspondant à la variable Valeur égale 1.
                 break;          // Rupture du cas 1.
case 10:         // Fragment de code correspondant à la variable Valeur égale 10.
                 break;          // Rupture du cas 10.
case 20:         // Fragment de code correspondant à la variable Valeur égale 20.
                 break;          // Rupture du cas 20.
case 50:         // Fragment de code correspondant à la variable Valeur égale 50.
                 break;          // Rupture du cas 50.
default:         // Fragment de code correspondant à la variable Valeur égale 0.
                 break;          //Rupture de default.
}
```

2.6.4 Les formes itératives while et do while

Le cycle itératif répète et exécute un fragment de programme à condition que la condition énoncée dans *while* soit vraie. Pour savoir comment déclarer ce type de forme itérative, on prend l'exemple suivant:

```
short CONT=0;                    //Déclaration de la variable entière qui compte jusqu'à 100.
while( CONT<100 )                //Déclaration de la boucle while.
{
CONT++;                          //Incrémentation de la variable CONT.
}
```

Mettre en œuvre une boucle *do while* est similaire à la boucle *while*, à la différence qu'on entre dans la boucle, le fragment de code est exécuté, puis il évalue la condition contenue dans *while.* En conclusion, le code est exécuté au moins une fois avant l'évaluation de la condition. Dans l'exemple ci-dessous, on peut voir comment utiliser ce type de boucle:

```
short CONT=0;                    //Déclaration de la variable entière qui compte jusqu'à 100.
do                               //Déclaration de la boucle do while.
{
CONT++;                          //Incrémentation de la variable CONT.
} while( CONT<100 );
```

2.6.5 La forme itérative for

La forme itérative *for* est une structure similaire à la boucle *while*, la différence est que *for* a une définition de la condition, entre les parenthèses, plus complexe. La condition a trois paramètres, le premier est un champ pour donner les valeurs initiales à un ou plusieurs variables, le second champ permet de créer une condition logique qui fait répéter le fragment du code *for* tant qu'elle est vraie, le dernier champ réalise la modification d'une ou plusieurs variables, telles que: incrémentation, décrémentation, etc.

Les variables qui sont modifiées dans le dernier champ sont généralement les mêmes qui ont été initialisées dans le premier champ, mais ce n'est pas une règle. On observe la structure *for* sur l'exemple suivant:

```
for( _INITIALISATION_ ; _CONDITION_; _MODIFICATION_ )
{
}
```

Pour connaitre le fonctionnement de la boucle *for* on observe l'exemple suivant:

```
short A;                                    //Déclaration des variables.
short B;
for( A=0, B=100; (A<100)||(B>20); A++, B-- ) //structure de la boucle for.
{
                // Code qui s'exécute lorsque la condition de for est vraie.
}
```

2.6.6 Utilisation des formes itératives imbriquées

La plupart des applications utilisent des boucles imbriquées pour accroître la fonctionnalité et optimiser la taille du code machine final. L'imbrication des boucles consiste à exécuter une boucle

dans une autre. Une boucle peut contenir une ou plusieurs boucles imbriquées. Pour mieux comprendre ce concept, on considère l'exemple suivant:

```
short COL;                          //Déclaration des variables.
short LIN;
short MATRICE[10][10];              //Déclaration d'un vecteur bidimensionnel.
for( COL=0; COL<10; COL++ )         //Déclaration de la boucle for.
{
                                    //Fragment du code de la première boucle.
for( LIN=0; LIN<10; LIN++ )         //Déclaration de la boucle  for imbriquée.
{
                                    //Fragment du code de la boucle imbriquée.
MATRICE[COL][LIN] = COL*LIN;
}
}
```

Dans l'exemple ci-dessus, les valeurs stockées sont dans une matrice de 10 sur 10, recouvrant toutes les positions de la matrice.

3. Le simulateur ISIS de PROTEUS :

Le simulateur ISIS de Proteus est un logiciel puissant, développé par la compagnie électronique Labcenter, qui s'est positionné pour plus de 10 ans comme l'un des outils les plus utiles pour simuler les microcontrôleurs PICmicro.

L'ISIS permet la simulation des familles des PICmicro les plus populaires tels que: 12F, 16F, 18F. En plus des PIC, ISIS peut simuler une variété de dispositifs numériques et analogiques, y compris les afficheurs sept segments, les LCD des caractères et graphiques. ISIS permet de simuler des capteurs de température, l'humidité, la pression et la lumière, entre autres.

Le simulateur peut simuler les actionneurs tels que des moteurs CC, les servomoteurs, les lampes à incandescence, entre autres. Il est possible de simuler des périphériques d'entrée et de sortie comme les claviers, ordinateur et les ports physiques tels que RS232 et USB. Ce simulateur a une grande variété d'instruments de mesure tels que voltmètres, ampèremètres, oscilloscopes et analyseurs de signaux.

En conclusion ces caractéristiques et d'autres font d'ISIS de Proteus, un outil idéal pour la conception et l'étude des PICmicro. Une version de démonstration du logiciel peut être téléchargée à partir du site Web: www.labcenter.com. Sur la photo suivante, on peut voir l'aspect visuel de l'environnement de développement d'ISIS:

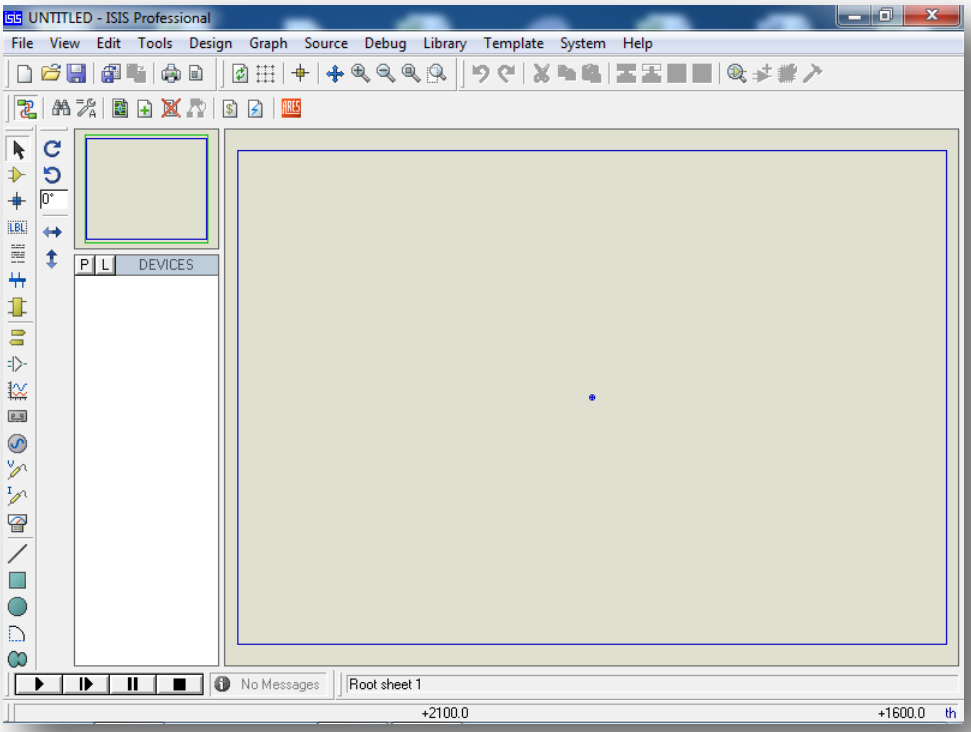


Figure 3-1

L'utilisation et le travail avec ISIS exigent un large éventail d'outils et d'options qu'on doit savoir. On va les découvrir progressivement au cours des exemples.

3.1 Les bases du simulateur ISIS

La section suivante montre les caractéristiques de base pour commencer la première simulation dans ISIS. Dans une première étape, on identifie la palette *sélecteur d'objets*, c'est-à-gauche de l'écran, le développeur doit identifier la touche P dans la palette comme le montre l'image ci-dessous:

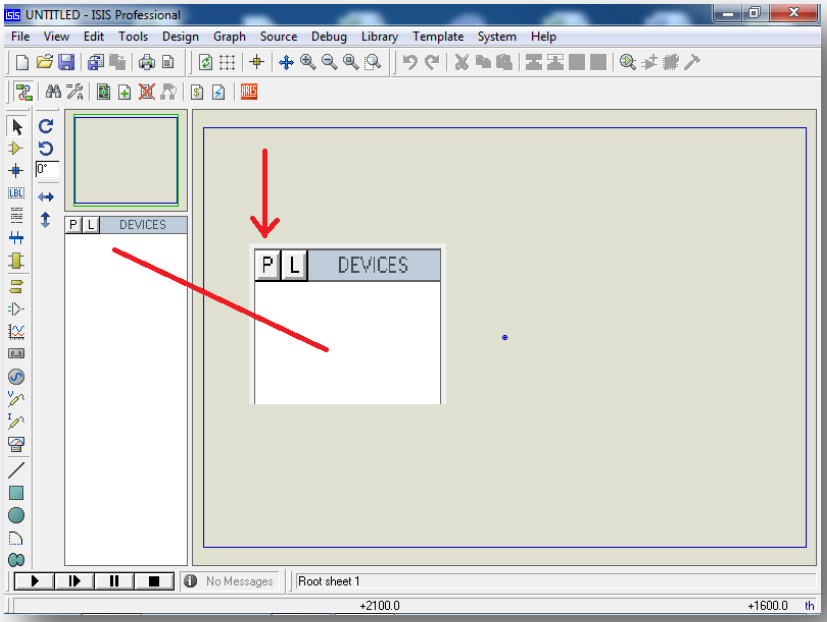


Figure 3-2

Lorsqu'on appuie sur la touche P, le programme ouvre une nouvelle fenêtre qui permet de trouver les dispositifs électroniques à travers la référence commerciale, ou sous la classification adopté par ISIS. Pour rechercher un dispositif, on entre la référence dans la zone: Keywords, le programme génère une liste, sur le côté droit de la fenêtre, des dispositifs en fonction de la recherche de l'utilisateur.

Pour sélectionner un dispositif, on double-clique sur sa référence dans la liste donnée. Pour commencer, on repère les dispositifs suivants: BUTTON, LED-RED, RES, correspondants au bouton poussoir, LED rouge, et une résistance. Après ce processus, dans la palette on devrait avoir les appareils suivants:

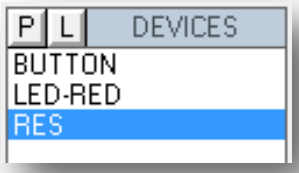


Figure 3-3

La prochaine étape consiste à trouver les bornes de la masse et l'alimentation, à cet effet, on clique sur l'icône suivant:  dans la palette d'outils, qui se trouve sur le côté gauche de la fenêtre du programme.

Lorsqu'on clique sur cet outil, une liste de terminaux qui apparait, on trouve GROUND et POWER, qui correspondent respectivement à la référence électrique ou la masse et à l'alimentation ou Vcc. La borne d'alimentation a une différence de potentiel de 5 volts par défaut. Pour placer ces terminaux dans la zone de travail, on clique sur les éléments dans la palette des terminaux, puis dans la zone de travail. Après cette action, la zone de travail devrait être comme la figure suivante:

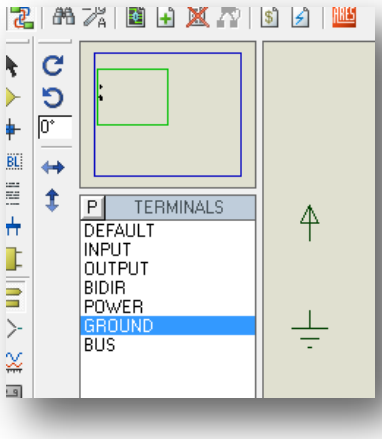


Figure 3-4

L'étape suivante consiste à fixer les dispositifs dans la zone de travail, on choisit le bouton: dans la palette d'outils sur la gauche. Pour coller les dispositifs dans l'espace de travail, on procède de la même manière que les terminaux, enfin on aura l'espace de travail suivant:

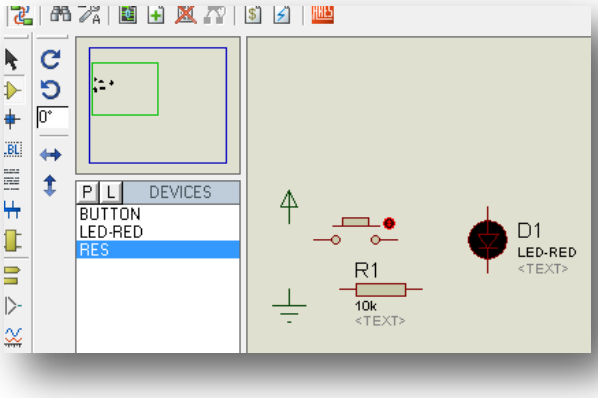


Figure 3-5

La prochaine étape est d'effectuer les connexions électriques, pour cet exemple, tous les éléments sont connectés en série, à cet effet le curseur de la souris prend la forme d'un crayon, pour lier entre deux bornes on clique sur la première puis la prochaine et le programme termine la connexion. À la fin des connexions, l'espace de travail devient:

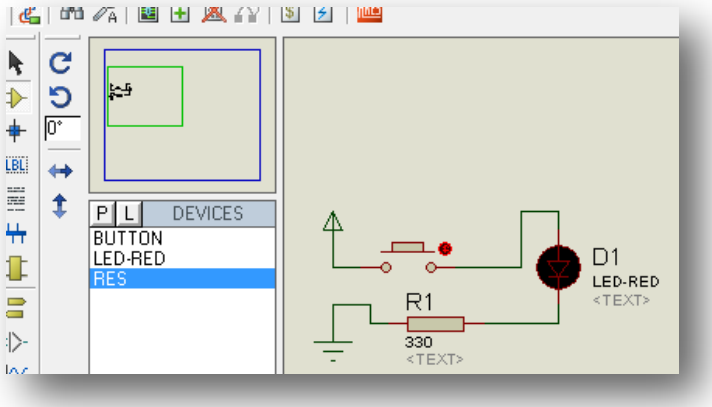


Figure 3-6

Enfin, on peut modifier la valeur de la résistance par un double clic de la souris sur la résistance, avec cette action on affiche une fenêtre pour modifier la valeur de la résistance, qui est par défaut 10k, ce qu'il faut faire est de la changer à 330.

L'étape suivante consiste à lancer la simulation, la palette située dans le coin inférieur gauche de l'écran permet de contrôler les simulations. Cette palette a l'apparence suivante:

Figure 3-7

Enfin, on appuie sur le bouton *play* et la simulation fonctionne, lorsque la simulation est en cours, on peut appuyer sur le bouton poussoir, et voir l'effet sur la simulation. Cet exemple simple montre le processus de création d'un schéma sur ISIS.

La création des circuits en ISIS, implique une autre série d'actions qui seront abordés au cours des exemples.

4. Création du premier programme en MikroC PRO

Le processus suivant doit être appris par cœur, pour mettre en œuvre à chaque fois des nouveaux projets ou programmes pour les PICmicro. En lançant MikroC PRO, on identifie dans le menu supérieur l'outil *Project*, et on choisit *New Project…* avec cette action, le programme affiche un assistant facile à utiliser pour créer un nouveau projet. L'aspect visuel de cet assistant est:

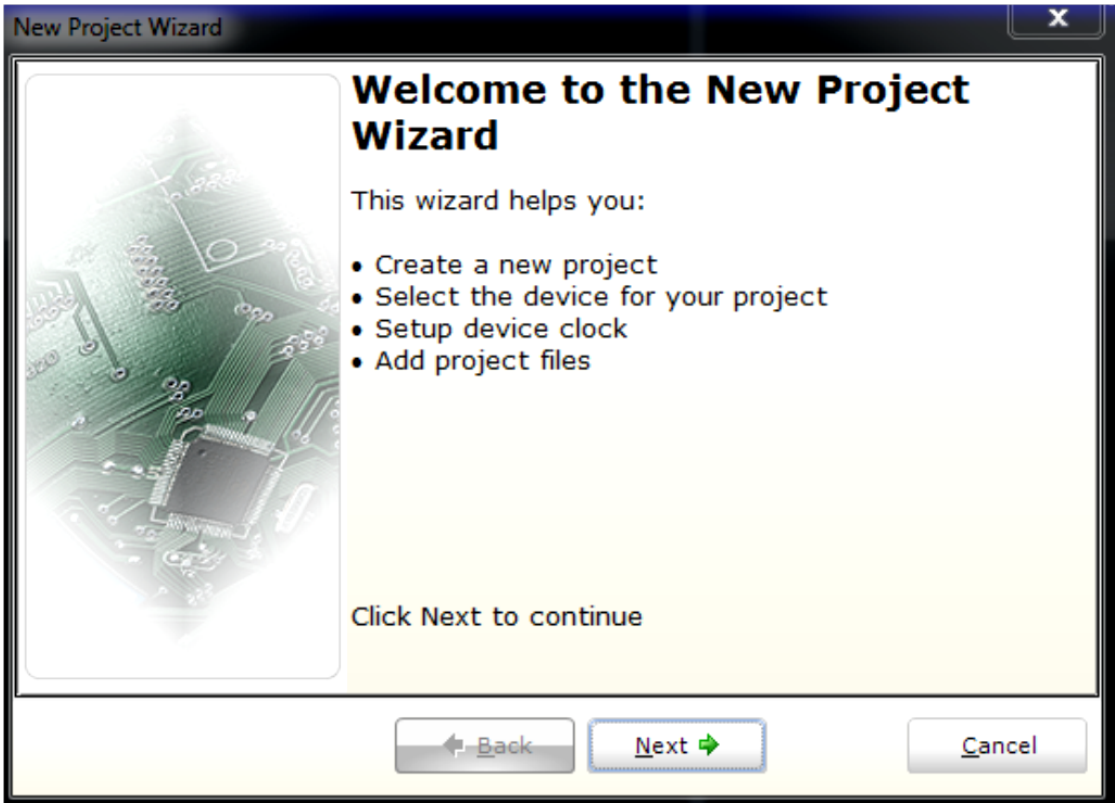


Figure 4-1

La prochaine action est de cliquer sur le bouton *Next*, à cette étape de l'assistant affiche une case pour sélectionner la référence de PICmicro, qu'on souhaite utiliser. Dans ce champ, on sélectionne le PIC P16F84A. L'étape suivante est de définir la fréquence d'oscillation avec laquelle travaillera le PIC ; dans cet exemple on sélectionne 4.000000 MHz. L'option suivante permet de définir le répertoire où le développeur enregistrera le projet, dans ce répertoire le programme enregistrera tous les fichiers nécessaires, parmi lesquels le code source qui sera archivé avec l'extension *.c*, et l'exécutable du PIC de avec l'extension. *hex*.

L'étape suivante, l'assistant demande d'ajouter des fichiers à joindre au projet. Lorsqu'on crée un nouveau projet, cette étape peut être ignorée, et on appuie sur *Next*, le dernier élément de l'assistant demande si le développeur veut sélectionner les bibliothèques qu'il va utilisées dans ce travail, par défaut, ce paramètre sélectionne toutes les bibliothèques utilisables pour ce PIC, le meilleur est de laisser toutes les bibliothèques actives. Enfin, la configuration est terminée et le projet est créé, à la fin la fenêtre doit apparaître comme suit:

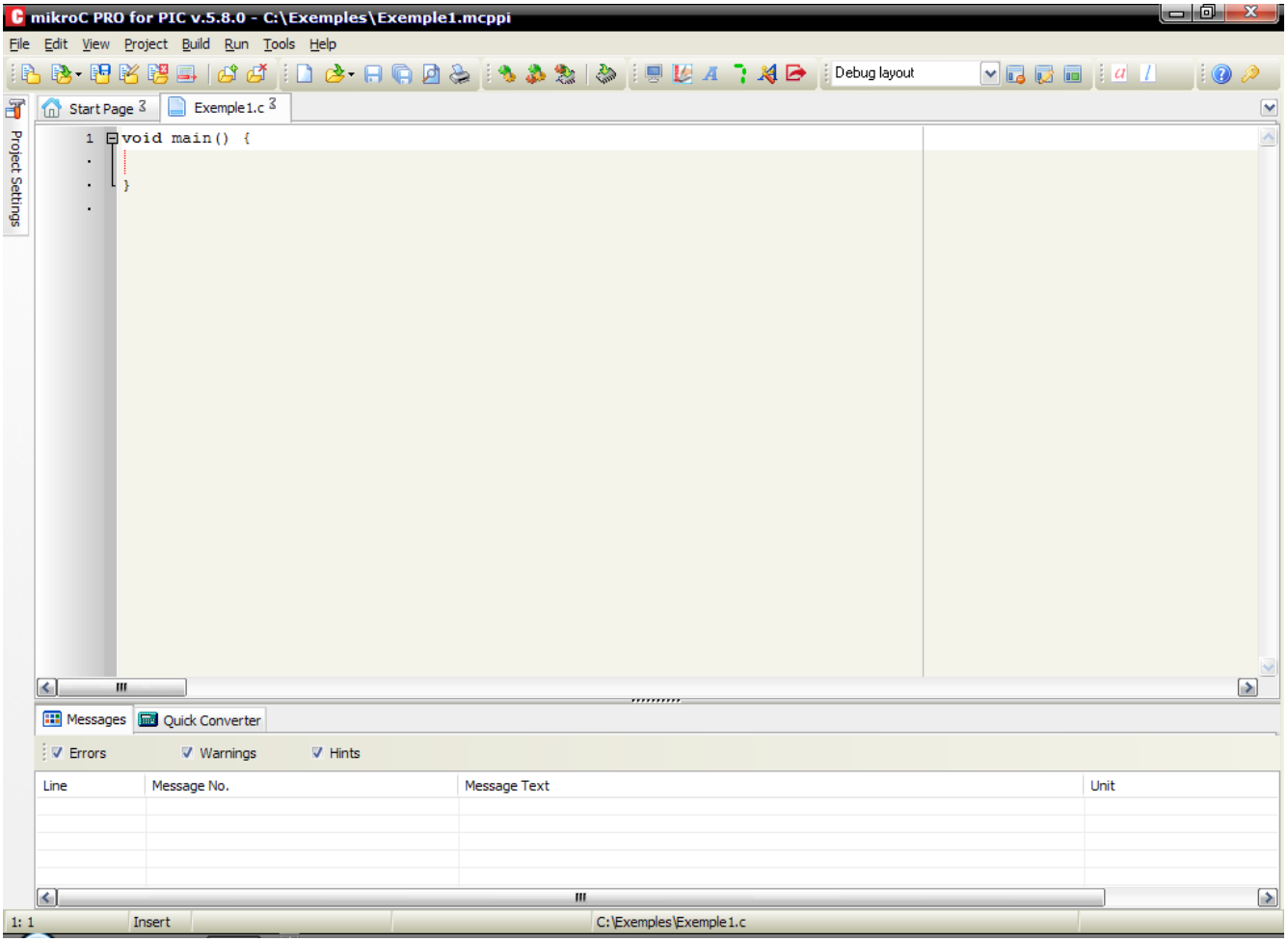


Figure 4-2

A chaque modification effectuée sur le code du programme, on doit compiler le code en appuyant sur le bouton suivant  : il est situé en haut du programme dans la barre d'outils. Cette action produit des résultats de la compilation qu'on trouve dans la partie inférieure de la fenêtre de programme. Les messages doivent se terminer par un text ''*Finished successfully*''.

Pour commencer l'édition d'un projet, on configure les ports du PIC, puis on insère le programme dans une boucle infinie. Le PIC 16F84A dispose de deux ports A et B, on manipule le registre TRIS pour configurer les ports en sortie ou entrée. Chaque port a son registre TRIS respectif, qu'on configure avec trois états possibles, états haut, bas, et haute impédance.

Les registres TRIS ont le même nombre de bits que les ports, par exemple le port B ou PORTB de ce PIC est de 8 bits, donc le TRISB comporte également 8 bits. Les bits des registres TRIS correspondent à ceux des ports, et définissent bit à bit l'état du port. Si un bit du TRIS est 0 le même bit du port est en sortie, et Si un bit du TRIS est 1 le même bit du port est en entrée ou en haute impédance. Pour voir ce concept plus clairement, on observe et on analyse l'exemple suivant:

```
TRISB = 0b11110000;          // Configure le quartet des bits de poids faible en
                             //sortie, et le quartet des bits de poids fort en entrée.
PORTB=0b00000000;            //Les bits de sortie ont un 0 logique
```

Cet exemple utilise un bouton et deux LED pour afficher le comportement du programme. On observe et on analyse le programme ci-dessous:

```
void main ( void )

{

        unsigned int COMPTEUR=0;

        TRISB = 0b11110000;             // Configure le quartet des bits de poids faible en
                                        sortie, //et le quartet des bits de poids fort en entrée.
        PORTB=0b00000000;               // Les bits de sortie ont un 0 logique
        while( 1 ) //Boucle infinie
        {
                if( PORTB.F7==0 )       //test si bit RB7 est à 0
                {
                        if( PORTB.F0==1 )   //test la valeur du bit RB0 et commute sa valeur.
                        PORTB.F0=0;
                        else
                        PORTB.F0=1;
                        while( PORTB.F7==0 ); //attend que  RB7 passe à 1.
                }
                COMPTEUR++;             //Incrémente la valeur du COMPTEUR.
                                        //La condition if qui suit change automatiquement
                                        l'état //du bit RB1tout les 256 incrémentations
                if(COMPTEUR&0x0100 )    //test si le bit 8 du COMPTEUR est 1
                PORTB.F1=1;
                else
                PORTB.F1=0;
        }
}
```

L étape suivante est de faire la simulation sur ISIS, les résistances des LED doivent être changées à 330Ω, l'entrée Master CLeaR , MCLR doit être connecté à Vcc pour que le PIC ne redémarre pas ,à la fin on devrait voir la forme suivante:

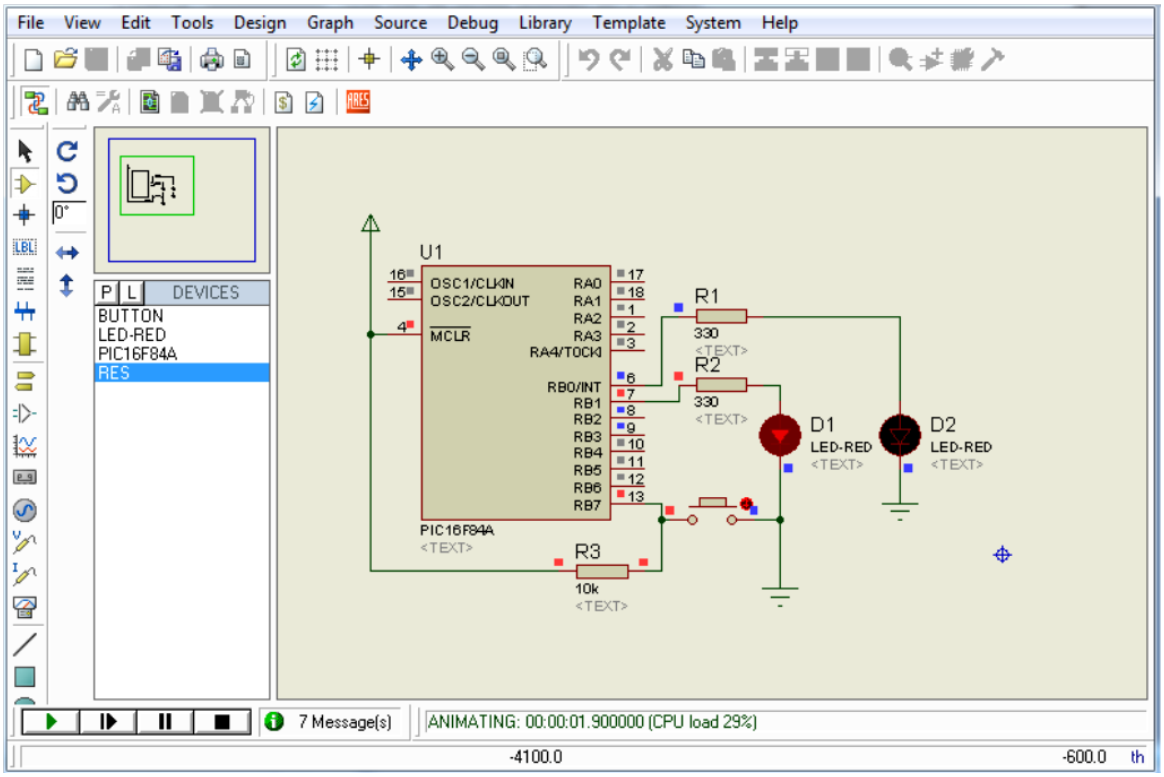


Figure 4-3

Avant de lancer la simulation, on doit charger le fichier .Hex. Pour procéder on double clique sur le PIC, et une fenêtre qui permet de rechercher le fichier .Hex, dans cette fenêtre on peut également ajuster la fréquence d'oscillation. Par défaut, cette valeur est de 1 MHz, à des fins de simulation dans ce cas, on doit sélectionner 4 MHz, cette fenêtre se présente comme suit:

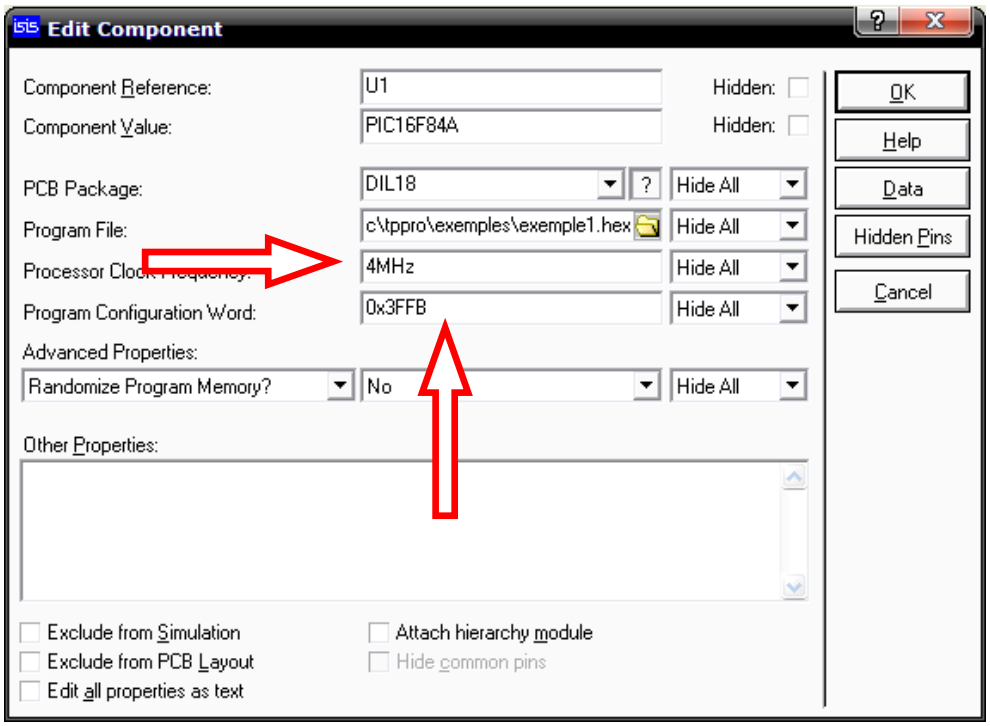


Figure 4-4

Pour des fins de la programmation physique du microcontrôleur, il est important de tenir compte des conditions de la configuration du PIC, chacun des microcontrôleurs Microchip possède un champ de mémoire qui caractérise le comportement réel de la PIC, entre elles on peut souligner, le type d'oscillateur ou horloge de traitement, par exemple quand on utilise un quartz de 4MHz, l'oscillateur doit être réglé sur XT, si l'oscillateur est de 20MHz on utilise la configuration HS. S'il s'agit d'un quartz d'une faible vitesse comme 400KHz, on utilise la configuration LP, et si un oscillateur est mis en œuvre en utilisant un circuit RC ou une résistance en série avec un condensateur, on utilise l'option RC.

De même, il est possible de configurer d'autres options telles que l'utilisation du chien de garde et la protection du code en mémoire FLASH et l'EEPROM, cette dernière option est très importante pour protéger les informations du microcontrôleur et d'empêcher le programme ou les données de la mémoire d'être lues. Pour modifier la configuration du PIC, on doit chercher *Edit Project* ...qui se trouve dans l'onglet Project du menu principal de MikroC PRO.

L'accès à cet outil est expliqué dans la figure suivante:

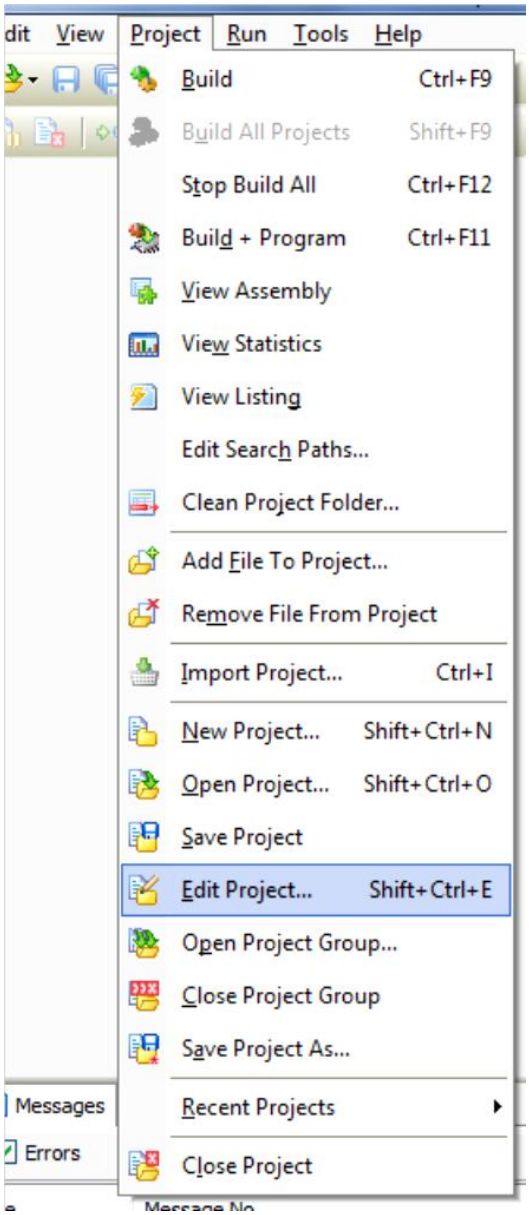

dit View Project Run Tools Help
Build Ctrl+F9
Build All Projects Shift+F9
Stop Build All Ctrl+F12
Build + Program Ctrl+F11
View Assembly
View Statistics
View Listing
Edit Search Paths...
Clean Project Folder...
Add File To Project...
Remove File From Project
Import Project... Ctrl+I
New Project... Shift+Ctrl+N
Open Project... Shift+Ctrl+O
Save Project
Edit Project... Shift+Ctrl+E
Open Project Group...
Close Project Group
Save Project As...
Recent Projects
Close Project
Messages
Errors

Figure 4-5

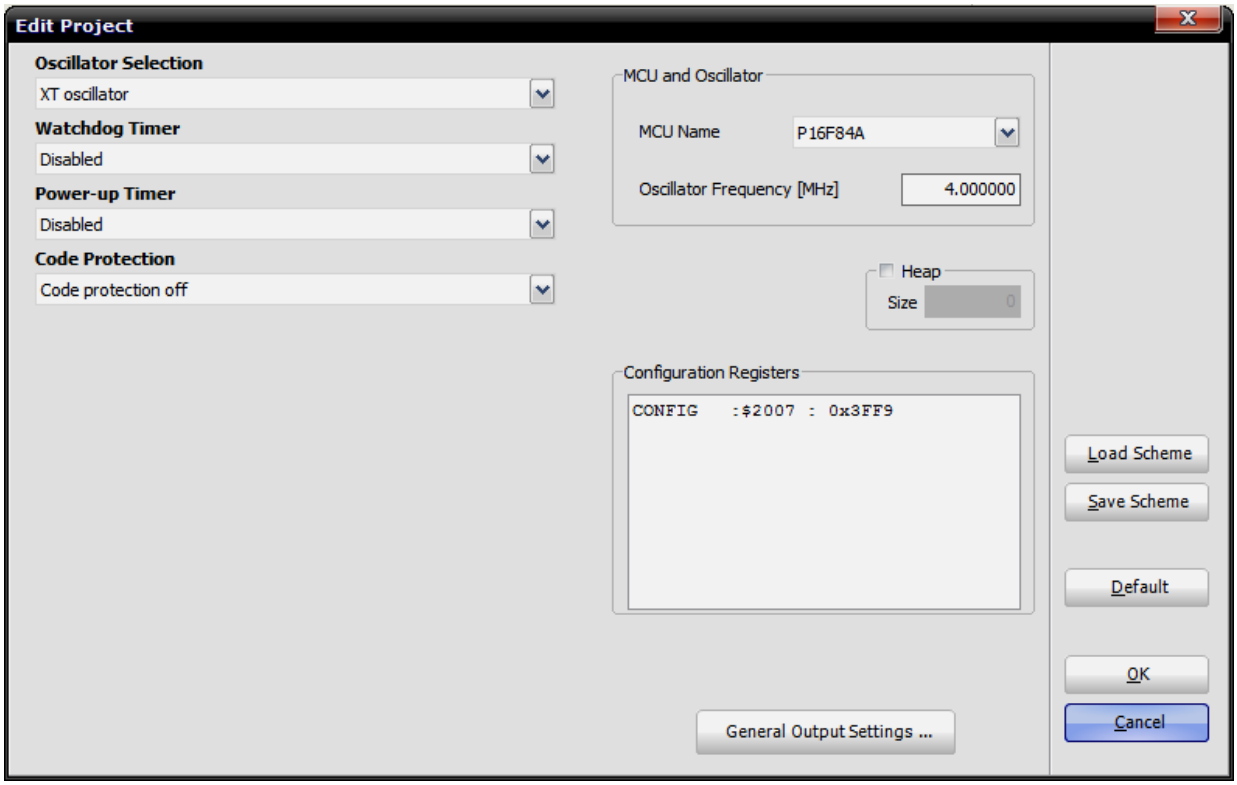

Edit Project
Oscillator Selection
XT oscillator
Watchdog Timer
Disabled
Power-up Timer
Disabled
Code Protection
Code protection off
MCU and Oscillator
MCU Name
P16F84A
Oscillator Frequency [MHz]
4.000000
Heap
Size
0
Configuration Registers
CONFIG :$2007 : 0x3FF9
Load Scheme
Save Scheme
Default
OK
Cancel
General Output Settings ...

Figure 4-6

5. Affichage des données

La conception des systèmes à microcontrôleurs implique parfois l'affichage des données à l'utilisateur. A cet effet on peut utiliser des afficheurs 7 segments, l'afficheur de caractère à cristaux liquides LCD et l'afficheur LCD à écran graphique. Dans ce chapitre, on étudie et on illustre ces dispositifs par des exemples.

5.1 Affichage à 7 segments

Un afficheur 7 segments est un dispositif qui permet de visualiser un nombre limité de caractères essentiellement numériques, mais il est possible de visualiser quelques caractères comme: b, d, E, A, ou F, C, -. L'afficheur 7 segments est un ensemble de LED, disposées de sorte qu'on visualise les caractères en activant les segments convenables.

Les afficheurs 7 segments ont une désignation standard, à chaque segment on attribue une lettre de "a" jusqu'à "g". La description et l'apparence physique de ces afficheurs sont illustrées sur les images suivantes:

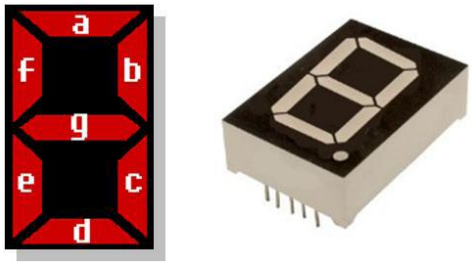


Figure 5-1

Les afficheurs à 7 segments sont fabriqués en deux formats; anode commune et cathode commune, les afficheurs à 7 segments existent également dans un format dynamique, ces derniers utilisent deux chiffres ou plus dans un seul boitier reliant tous les segments en parallèle, mais avec bornes communes séparées. Les figures suivantes montrent des afficheurs 7 segments dans leur forme dynamique:

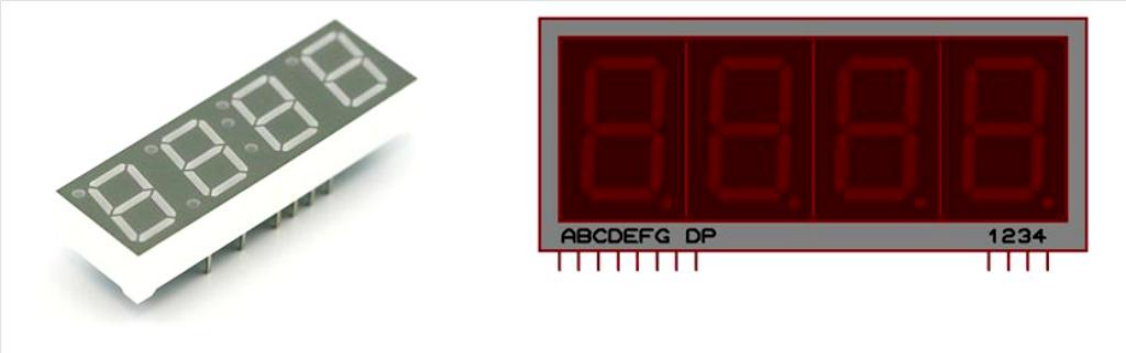


Figure 5-2

L'entrée DP, qu'on peut voir sur la figure ci-dessus, est le huitième segment. Il est mis en œuvre dans certains affichages et correspond à la virgule, il est utilisé si la demande l'exige.

5.1.1 Contrôle de l'affichage à 7 segments

Pour effectuer le premier exemple avec cet afficheur on doit savoir comment réaliser l'affectation des broches du PIC. A chaque segment de l'afficheur, on doit relier une broche du PIC. Pour faire une conception optimale, on peut attribuer les broches en ordre consécutif d'un port, par exemple le

segment "a" est relié à la broche RB0, le "b" à la broche RB1, et ainsi de suite, jusqu'au segment "g" à la broche RB6. Cependant, le développeur peut faire l'affectation des broches d'une manière arbitraire.

Il est important de connaitre un outil contenu dans le logiciel MikroC PRO qui permet d'éditer les digits de l'afficheur. Pour cela on utilise *Tools* dans la barre de menu, et on choisit l'outil *Seven Segment Editor*.

Avec cette action, ressort une nouvelle fenêtre où on peut éditer d'une manière simple les segments de l'afficheur à 7 segments. Au moment de l'édition de l'afficheur, apparait la valeur constante qui doit être utilisée sur les sorties du port désigné pour son contrôle. L'éditeur permet de mettre en œuvre des constantes pour l'afficheur anode ou cathode commune. De la même façon, on peut utiliser les valeurs en décimal ou hexadécimal. L'aspect visuel de l'éditeur est:

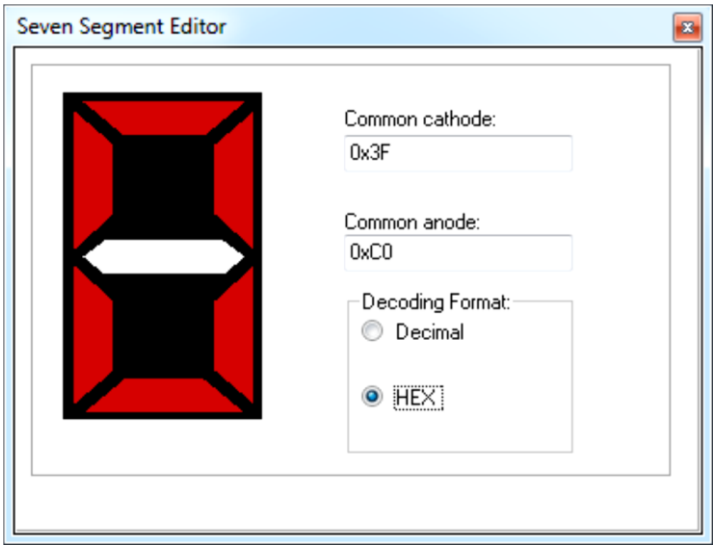


Figure 5-3

L'utilisation de cet outil signifie que toutes les broches de l'afficheur sont attribuées au même port, et dans un ordre consécutif comme on a vu ci-dessus. Pour la visualisation des chiffres dans l'afficheur, on doit organiser les informations de ces digits dans un ordre consécutif, dans le cas de cet exemple de 0 à 9. A cette fin, la forme la plus simple est l'utilisation d'un tableau de données contenant les codes des 10 chiffres. Dans l'exemple suivant, on déclare un tableau constant avec les codes des 10 chiffres, et on utilise un afficheur à cathode commune. On observe comment déclarer des constantes qui doivent être insérées avant la fonction *main*:

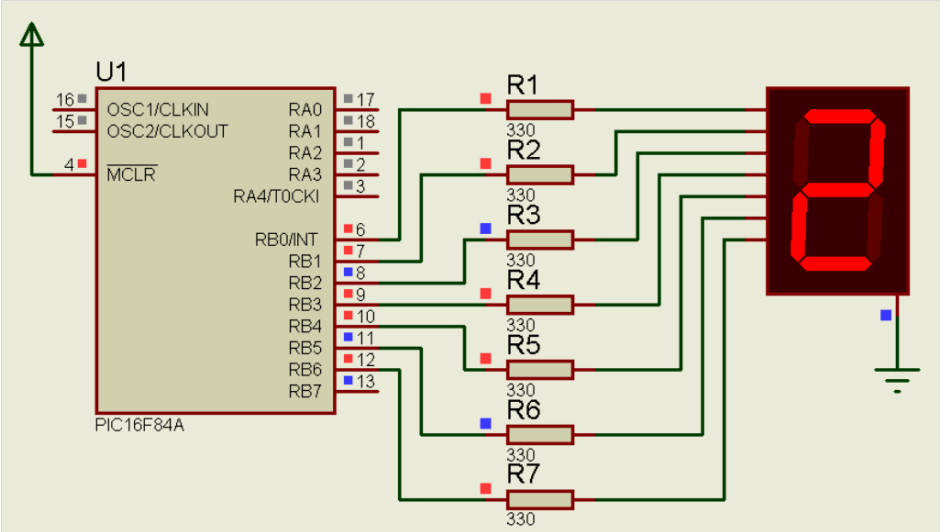
```
const unsigned short DIGITS[] =
{
        0x3F, //Code du digit 0
        0x06, // Code du digit 1
        0x5B, // Code du digit 2
        0x4F, // Code du digit 3
        0x66, // Code du digit 4
        0x6D, // Code du digit 5
        0x7D, // Code du digit 6
        0x07, // Code du digit 7
        0x7F, // Code du digit 8
        0x6F, // Code du digit 9
};
```

Pour afficher les chiffres dans l'afficheur à 7 segments, cet exemple va utiliser le port B du PIC 16F84A. La visualisation des numéros est contrôlée par une routine, qui change le chiffre après une temporisation réalisée par une fonction *delay_ms*, cette fonction qui est prédéfinie dans les bibliothèques du compilateur. Cette fonction a pour paramètre d'entrée un entier représentant le temps en millisecondes, pendant lequel le PIC effectue un retard, de la même manière on peut utiliser la fonction *delay_us*, qui est identique à *delay_ms* mais en microsecondes.

Alors, le code source du PIC pour cet exemple est le suivant:

```
const unsigned short DIGITS[] =
{
0x3F, //Code du digit 0
0x06, //Code du digit 1
0x5B, //Code du digit 2
0x4F, //Code du digit 3
0x66, //Code du digit 4
0x6D, //Code du digit 5
0x7D, //Code du digit 6
0x07, //Code du digit 7
0x7F, //Code du digit 8
0x6F, //Code du digit 9
};
void main ( void )
{
unsigned short COMPTEUR=0;
TRISB = 0;                          // Configuration du port B en sortie
while( 1 )                          //Boucle infinie
{
PORTB = DIGITS[COMPTEUR];           //Visualisation du chiffre correspondant à la valeur
                                    //de  la variable COMPTEUR.
COMPTEUR++;                         //Incrémentation la valeur de compteur.

delay_ms(1000);                     //Retard de 1 seconde.
}
}
```

L'étape suivante consiste à effectuer la simulation dans ISIS, pour cet effet, les dispositifs suivants sont découverts: 16F84A, RES, et 7SEG-COM-CATHODE. Pour la liaison entre le PIC et l'afficheur, on devrait utiliser les résistances de 330Ω. Par la suite, on réalise le circuit suivant :



Circuit 5-1

A l'exécution de la simulation, on doit voir sur l'afficheur un comptage des nombres de 0 à 9, avec une cadence d'une seconde entre chiffres et chiffre.

5.1.2 Contrôle de l'affichage dynamique 7 segments

La mise en œuvre des afficheurs dynamiques utilise la même théorie d'un seul afficheur, l'affichage dynamique consiste à afficher un seul chiffre à la fois. Par exemple, si on veut afficher quatre chiffres, on active l'afficheur des unités, ensuite on l'éteint et on active le chiffre des dizaines, puis on l'éteint et le chiffre des centaines est activé, enfin on fait la même chose avec l'afficheur des milliers.

Ce processus doit être effectué à une vitesse de manière à tromper l'œil humain, de sorte à voir comme si tous les chiffres ont été actifs. Cette gestion d'affichage minimise les connexions électriques et la consommation d'énergie, car en réalité un seul afficheur est actif à la fois.

Vue la perception de l'œil humain, les modifications doivent être de 25Hz ou plus, donc tous les chiffres doivent être montrés pendant une durée égale à l'inverse de 25Hz ,est égale à 40ms. Pour cet exemple, on utilise 4 afficheurs donc le temps d'affichage de chaque chiffre doit être un quart de la période qui est 10ms. Le moyen le plus efficace pour afficher les chiffres est d'utiliser une fonction.

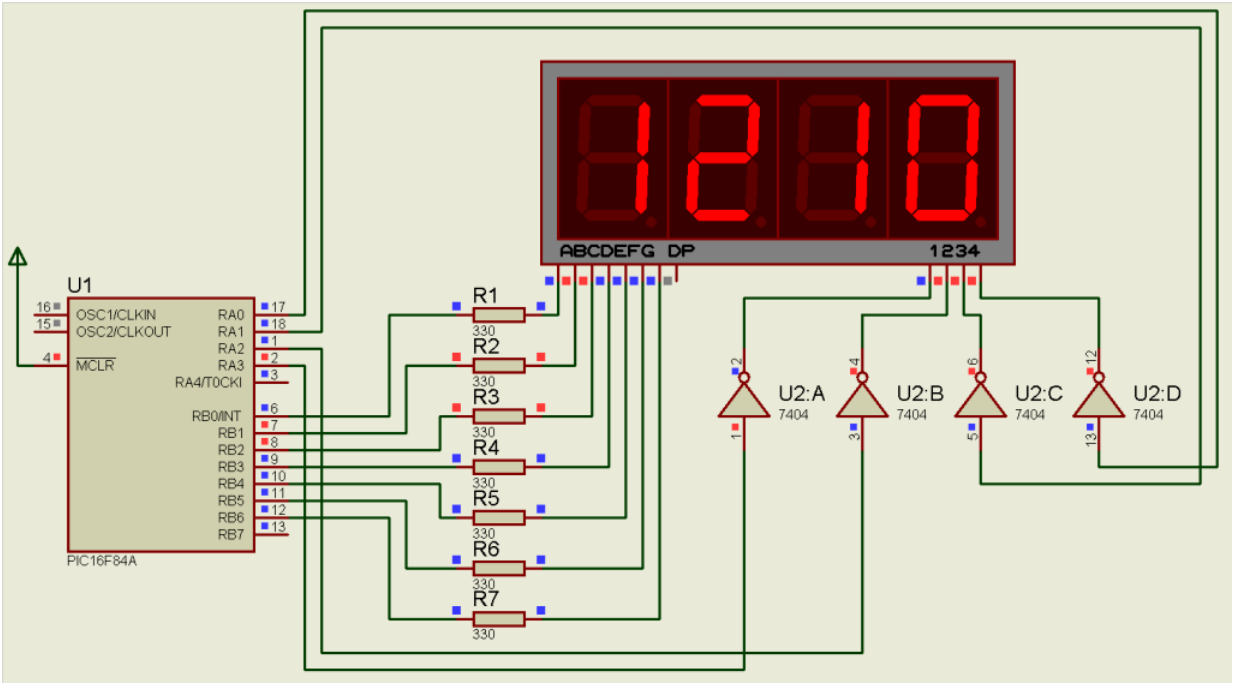
Avec 4 digits, il est possible d'afficher un nombre de 0 à 9999, à ce nombre on associe une variable de type entier. Pour voir chaque chiffre séparément, il est nécessaire de calculer chacun des chiffres, par exemple pour déduire le chiffre des milliers on divise le nombre par mille, puis soustraire les milliers du nombre entier, ce processus est répété jusqu'à atteindre les unités. L'activation des afficheurs doit se faire par un autre port, dans cet exemple elle est faite par le port A.

Pour comprendre ce processus, on observe et on analyse la fonction suivante:

```
                                        // Déclaration des constantes pour l'afficheur.
const unsigned short DIGITS[] =
{
0x3F, //Code du digit 0
0x06, //Code du digit 1
0x5B, //Code du digit 2
0x4F, //Code du digit 3
0x66, //Code du digit 4
0x6D, //Code du digit 5
0x7D, //Code du digit 6
0x07, //Code du digit 7
0x7F, //Code du digit 8
0x6F, //Code du digit 9
};
                                        //Fonction pour l'un affichage dynamique.
void Affichage( int Nombre )
{
unsigned short U;                       //Variable pour les unités.
unsigned short D;                       //Variable pour  les dizaines.
unsigned short C;                       //Variable pour les centaines.
unsigned short M;                       //Variable pour les milliers.
M = Nombre/1000;                        //Calcul des milliers.
C = (Nombre-M*1000)/100;                //Calcul des centaines.
D = (Nombre-M*1000-C*100)/10;           //Calcul des dizaines.
U = (Nombre-M*1000-C*100-D*10);         //Calcul des unités.
PORTB = DIGITS[U];                      //Visualisation des unités.
PORTA.F0=1;                              //Activer le premier afficheur
delay_ms(10);                           //Retard de 10ms
PORTA=0;                                //Désactiver tout les afficheurs.
```

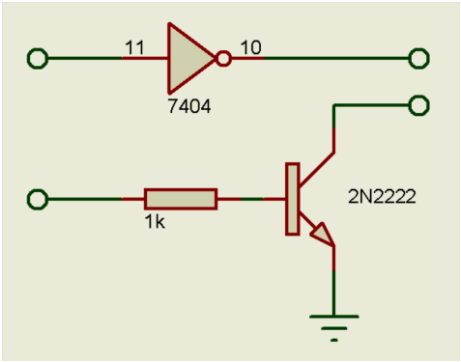
```
PORTB = DIGITS[D];                  // Visualisation des dizaines.
PORTA.F1=1;                         //Activer le second afficheur
delay_ms(10);                       // Retard de 10ms
PORTA=0;                            // Désactiver tout les afficheurs.
PORTB = DIGITS[C];                  //Visualisation des centaines.
PORTA.F2=1;                         //Activer le troisième afficheur
delay_ms(10);                       // Retard de 10ms
PORTA=0;                            // Désactiver tout les afficheurs.
PORTB = DIGITS[M];                  //Visualisation des centaines.
PORTA.F3=1;                         // Activer le troisième afficheur
delay_ms(10);                       // Retard de 10ms
PORTA=0;                            // Désactiver tout les afficheurs.
}

void main ( void )
{
unsigned short N=0;                 //Variable de comptage.
int Nombre=0;
TRISB = 0;                          //Configurer le port B en sortie
TRISA = 0;                          // Configurer le port A en sortie
PORTA = 0;                          //Désactiver tout les afficheurs
while( 1 )                          //Boucle infinie
{
                                    //Visualiser la valeur du Nombre.
Affichage(Nombre);                  //cette  fonction dure approximativement 40ms.
                                    //On compte 12 incrémentations de N pour faire une
                                    //incrémentation
                                    //du Nombre approximativement chaque 500ms.
N++;
if( N==12 )
{
N=0;                                //Initialiser le compteur N.
Nombre++;                           //Incrémentation de la valeur Nombre.
if( Nombre==10000 )                 //Test si Nombre vaut 10000
Nombre=0;                           //Initialisation à 0 si nombre = 10000.
}
}
}
```

On compile le programme. Finalement, on réalise la simulation sur ISIS avec les dispositifs suivants 16F84A, RES, 7SEG-MPX4-CC,7404. Le circuit est le suivant:



Circuit 5-2

Pour des raisons pratiques, les inverseurs 7404 peuvent être remplacés par un réseau de transistors, comme on le voit dans l'image ci-dessous:



Circuit 5-3

Lorsque la simulation est en marche, ce circuit devrait afficher un nombre de 0 à 9999 avec une cadence de 500ms entre chaque incrément.

5.2 Afficheur LCD

Les afficheurs de caractères à cristaux liquides LCD sont des modules préfabriqués qui contiennent des pilotes inclus. Ces écrans disposent d'un bus de données et un bus de commande, pour la manipulation de ces appareils, le compilateur MikroC PRO dispose d'une bibliothèque prédéfinie pour contrôler les LCD. L'apparence physique des LCD et celle dans ISIS sont présentées dans la figure suivante:

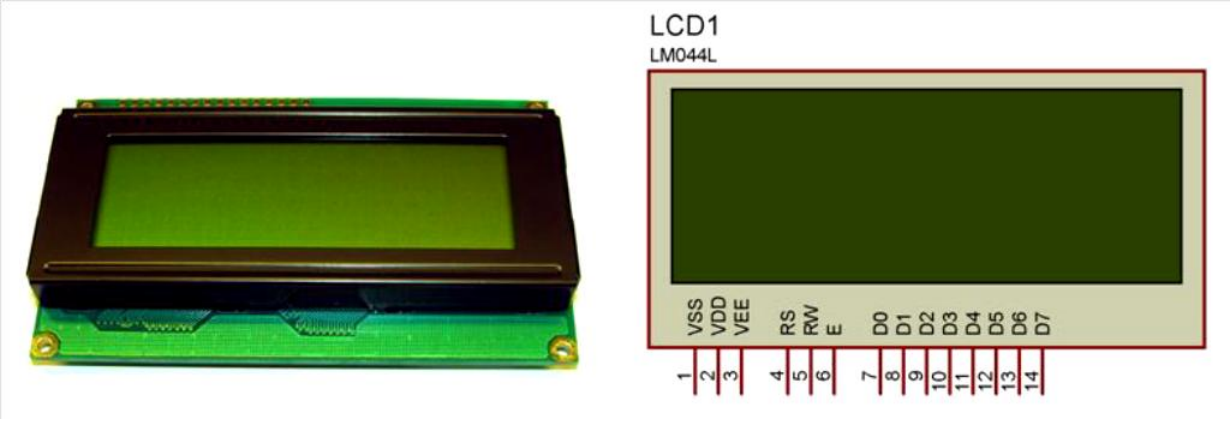


Figure 5-4

Les afficheurs LCD, permettent de visualiser les caractères figurant dans le code ASCII. En plus du code ASCII, LCD affiche jusqu'à 8 caractères conçus par le développeur. Une autre caractéristique fondamentale des afficheurs LCD, tenant physiquement sur 8 bits, est la possibilité de configurer les connexions avec seulement 4 bits. La connexion 8-bit implique un plus grand nombre de fils à utiliser, pour une meilleure vitesse de travail, par conséquent les connexions de 4 bits réduits le nombre de fils, mais diminue la vitesse. La bibliothèque prédéfinie dans MikroC PRO fonctionne avec une configuration de 4 bits.

Pour voir la bibliothèque prédéfinie pour ce dispositif et d'autres, contenues dans MikroC PRO, on choisit *View* dans la barre de menu, et on appuie sur l'un onglet : *Library Manager*. Lorsqu'on appuie sur cet onglet un menu montrant les différentes bibliothèques qu'on peut utiliser avec les PIC.

Dans ce nouveau menu, on identifie la bibliothèque *Lcd*, par la suite on peut appuyer sur une des fonctions contenues dans la bibliothèque pour voir l'aide. L'aspect visuel de cet outil est illustré dans la figure suivante:

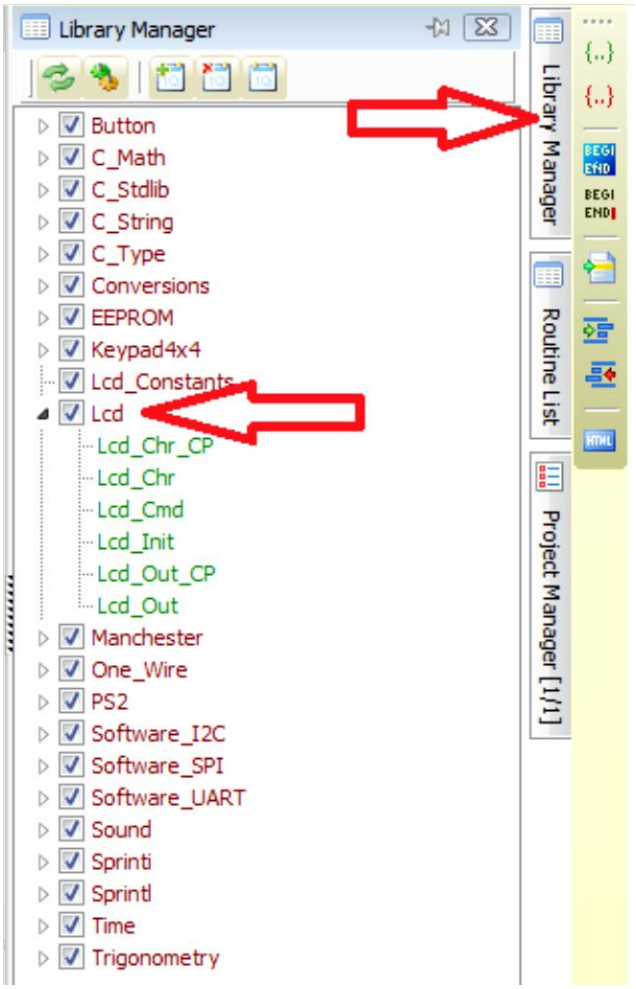


Figure 5-5

L'implémentation de l'afficheur LCD, requiert l'utilisation des instructions et des commandes séquentielles pour la configuration et l'utilisation de l'afficheur, cependant, la bibliothèque de MikroC PRO minimise ce travail, car elle se charge de faire tous ces réglages, ce qui rend beaucoup plus facile le travail développeur.

Comme première étape pour l'utilisation du LCD, il est nécessaire de définir les broches de connexion, puis l'exécution de la fonction d'initialisation du LCD : *Lcd_Init()*. La définition des broches de connexion est assurée par le développeur d'une façon arbitraire selon son choix. Pour répondre à cet objectif, on utilise la déclaration suivante des constantes :

```
//Broches de sortie du LCD
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D4 at RB0_bit;
//Bits de configuration TRIS
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D7_Direction at TRISB3_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D4_Direction at TRISB0_bit;
```

Pour changer les broches, il suffit de changer les noms des sorties du port utilisé dans l'exemple précédent. Comme on peut le voir dans la présentation ci-dessus seulement 6 broches sont nécessaires pour faire fonctionner le LCD, avec 4 bits de données et 2 bits de contrôle.

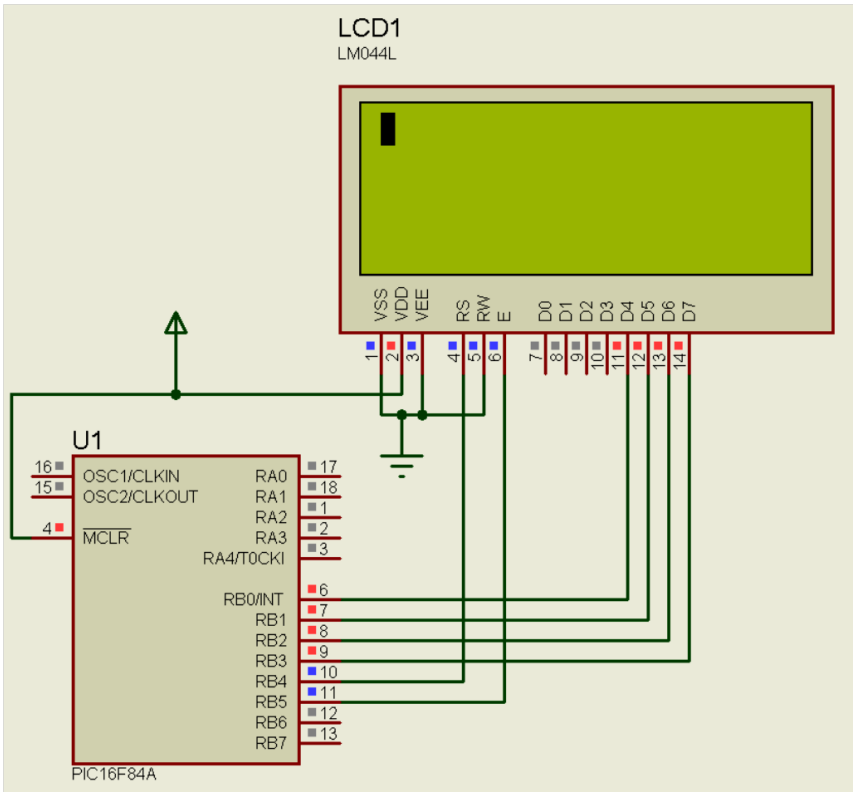
A la fin, on doit appeler la fonction d'initialisation dans la fonction *main* après configuration des ports. La fonction *main* doit être comme suit:

```
void main( void )
{
Lcd_Init();                    //Initialisation du LCD.
while(1)                       //Boucle infinie.
{
}
}
```

Après l'édition du code ci-dessus, l'afficheur LCD est initialisé et devrait être prêt à commencer l'affichage des caractères, se positionnant dans la première ligne et la première colonne, montrant le curseur clignotant.

Les LCD sont fabriqués dans différentes formes et couleurs, ils sont disponibles avec des écrans vert, bleu et jaune, avec des distributions de caractères sous forme de matrice comme les LCD 2 lignes, 16 colonnes. Ceux-ci sont connus comme 2x16, de la même manière on peut trouver des 1x16, 2x16, 2x8, 2x20, 4x20, entre autres. Pour les exemples de ce chapitre, on utilisera l'afficheur 4x20.

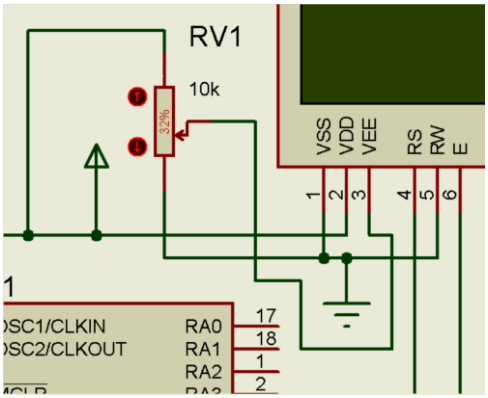
Pour démarrer la simulation de l'afficheur LCD, on cherche le dispositif de LM044L, et PIC 16F84A dans le simulateur ISIS. La référence LM044L dans ISIS correspond à un LCD de 4x20. Enfin, on effectue les connexions comme indiqué dans le circuit suivant:



Circuit 5-4

L'afficheur LCD a une broche nommée VEE, cette broche fonctionne comme contrôleur du contraste de l'écran, mais dans la simulation, elle n'a pas d'effet. Cette broche peut être reliée à la masse pour

générer le contraste le plus élevé, certains grands écrans nécessitent une tension négative externe pour contrôler le contraste. Pour des raisons pratiques, le contraste peut être ajusté par l'intermédiaire d'un potentiomètre, comme indiqué dans le circuit suivant:



Circuit 5-5

La prochaine étape consiste à afficher à l'écran des informations. A cet effet, on peut utiliser quatre fonctions. Deux de ces fonctions permettent d'afficher des caractères, et les deux autres des chaînes de caractères.

Pour afficher les caractères, on peut procéder de deux façons, la première affiche simplement les caractères dans l'ordre consécutif pris par l'afficheur et la seconde fonction imprime les caractères dans la ligne et la colonne désignées par le développeur.

5.2.1 Fonctions d'affichage des caractères

La première fonction d'affichage des caractères est *Lcd_Chr_Cp (char out_char)* ; lorsque cette fonction est invoquée affiche sur l'écran le caractère qui correspond au code ASCII, qui est le paramètre d'entrée *out_char*. Avec l'impression d'un nouvel caractère sur l'afficheur LCD, le curseur se déplace automatiquement d'une position. Pour voir le fonctionnement de cette fonction, on observe l'exemple suivant:

```
void main( void )
{
Lcd_Init();                          //Initialisation du LCD.
Lcd_Chr_Cp('S');                     //cette fonction affiche lettre à lettre le mot  "Salut".
Lcd_Chr_Cp('a');
Lcd_Chr_Cp('l');
Lcd_Chr_Cp('u');
Lcd_Chr_Cp('t');
while(1)                             //Boucle infinie.
{
}
}
```

Après l'exécution de la simulation, on devrait observer ce qui suit sur l'afficheur LCD:

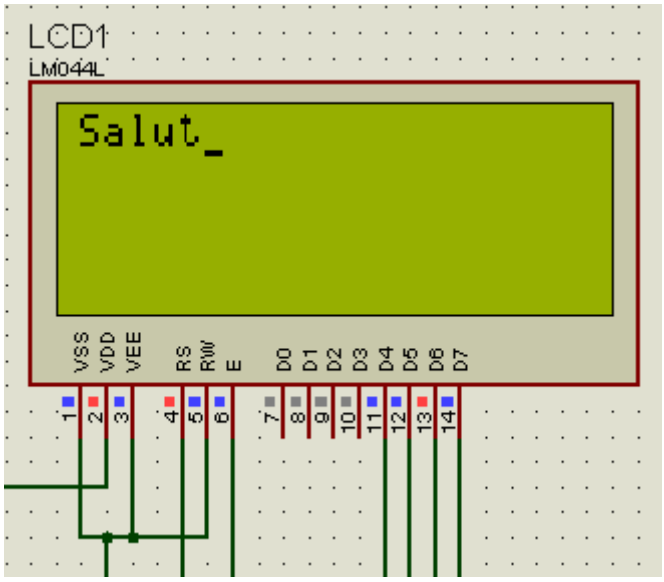


Figure 5-6

Pour l'affichage des caractères au moyen des cordonnées ligne, colonne, la fonction mise en œuvre: *Lcd_Chr(**char** row, **char** column, **char** out_char);* cette fonction imprime le caractère *out_char*, dans la colonne *column* et la ligne *row*. Dans l'exemple ci-dessous, on peut voir comment utiliser cette fonction:

```
void main( void )
{
Lcd_Init();                          //Initialisation du LCD
Lcd_Chr_Cp('S');                     //Ces functions impriment lettre à lettre le mot
                                     //"Salut".
Lcd_Chr_Cp('a');
Lcd_Chr_Cp('l');
Lcd_Chr_Cp('u');
Lcd_Chr_Cp('t');
Lcd_Chr( 1, 6, '1');                 //Imprime le caractère 1, dans  la ligne 1, colonne 6
Lcd_Chr( 2, 7, '2');                 // Imprime le caractère 2, dans  la ligne 2, colonne 7
Lcd_Chr( 3, 8, '3');                 // Imprime le caractère 3, dans  la ligne 3, colonne 8
Lcd_Chr( 4, 9, '4');                 // Imprime le caractère 4, dans  la ligne 4, colonne 9
while(1)                             //Boucle infinie.
{
}
}
```

Après l'exécution de la simulation, on devrait observer ce qui suit sur l'afficheur LCD:

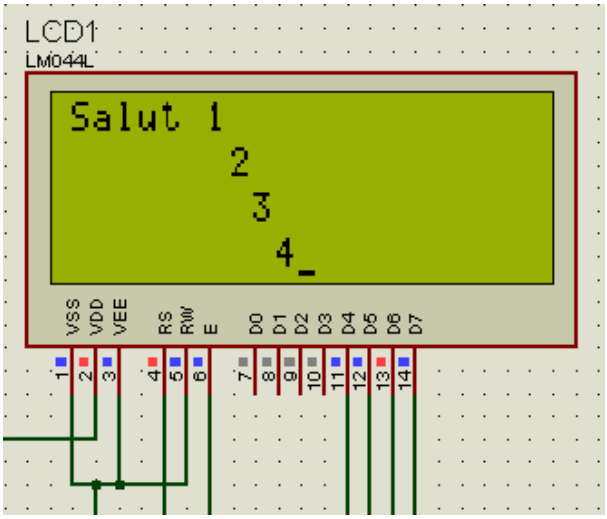


Figure 5-7

5.2.2 Fonctions d'affichage des chaînes de caractères :

L'utilisation de chaînes est similaire aux deux fonctions précédentes, pour cela on peut afficher des chaines de caractères à un point arbitraire à l'aide des coordonnées ligne et colonne. Pour imprimer une chaîne de caractères à la position courante du curseur, on utilise la fonction suivante: *Lcd_Out_Cp (char * text)* ; elle a un paramètre d'entrée unique qui est un pointeur vers la chaîne. Pour voir comment utiliser cette fonction, on observe l'exemple suivant:

```
void main( void )
{
Lcd_Init();                          //Initialisation du LCD.
Lcd_Out_Cp("salut tout le Monde");
while(1)                             //Boucle infinie.
{
}
}
```

La mise en œuvre de cette fonction peut être faite avec des chaînes de caractères constantes ou variables, des chaines constantes sont indiquées par des guillemets au début et à la fin du texte, par exemple "Salut tout le Monde ", la forme variable est déclarée : char texte [20] = "Salut tout le Monde ". Après avoir exécuté la simulation, on a comme montré dans la figure suivante:

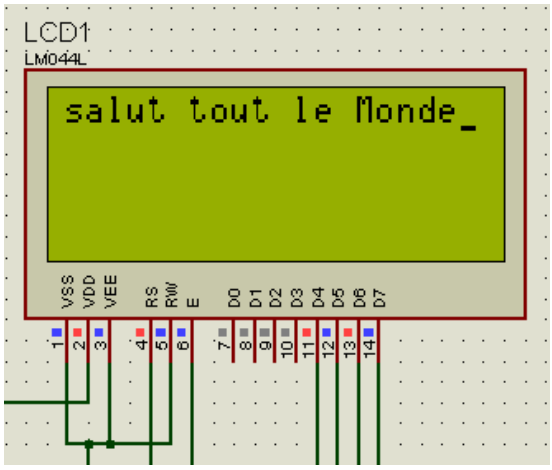


Figure 5-8

Pour imprimer une chaîne de caractères avec une coordonnée comme point de départ, on implémente la fonction : *Lcd_Out(**char** row, **char** column, **char** *text);*. Cette fonction est similaire à la précédente, à la différence qu'on inclut les données *row* et *column*, qui font référence respectivement à la ligne et la colonne. Pour bien comprendre le fonctionnement de cette fonction d'observe et on analyse l'exemple suivant:

```
void main( void )
{
Lcd_Init();                                  //Initialisation du LCD.
Lcd_Out( 1, 1, "Ligne 1, Colonne 1" );
Lcd_Out( 2, 2, "Ligne 2, Colonne 2" );
while(1)                                     //Boucle infinie.
{
}
}
```

Après la compilation et la simulation de cet exemple, on devrait observer sur l'écran LCD comme suit:

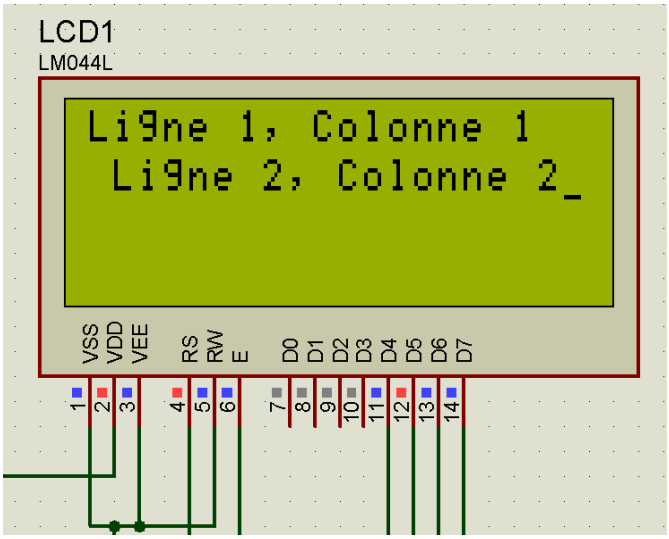


Figure 5-9

5.2.3 Affichage des valeurs numériques

La visualisation des valeurs numériques est indispensable dans de nombreux développements. Par exemple, lorsqu'on souhaite afficher l'état d'une variable, ou un capteur tel que : température, humidité, la pression, etc...

Pour atteindre cet objectif, on peut recourir aux fonctions de conversion prédéfinies par le compilateur. Pour cela, on peut utiliser la bibliothèque: *Conversions*, qui contient des fonctions qui convertissent une valeur numérique en une chaîne de caractères.

Si on veut afficher une valeur de type entier, on utilise la fonction *: IntToStr(**int** input, **char** *output)*; cette fonction a deux paramètres d'entrée sont: *input*, qui est une valeur entière à afficher, et *output*, qui est un pointeur sur une chaîne où on veut écrire la forme texte de la valeur *intput.* Pour comprendre ce type de conversion, on observe et on analyse l'exemple suivant:

```
void main( void )
```

```
{
int ENTIER=123;                    //Déclaration d'une variable entière avec  valeur
                                   //initiale 123.
char Text[20];                     //Chaine de caractères pour l'affichage des données.
Lcd_Init();                        //Initialisation du LCD.
IntToStr( ENTIER,Text );           //Fonction de conversion.
Lcd_Out_Cp(Text);                  //Impression du texte dans l'écran LCD.
while(1)                           //Boucle infinie.
{
}
}
```

L'impression des nombres entiers en chaîne de caractères avec cette fonction, réserve toujours un champ fixe de 7 caractères, c'est à dire si le nombre est inférieur à 7 chiffres le reste du texte est complété par des espaces vides. Après compilation du programme et simulation sur ISIS, on a un résultat comme dans la figure suivante:

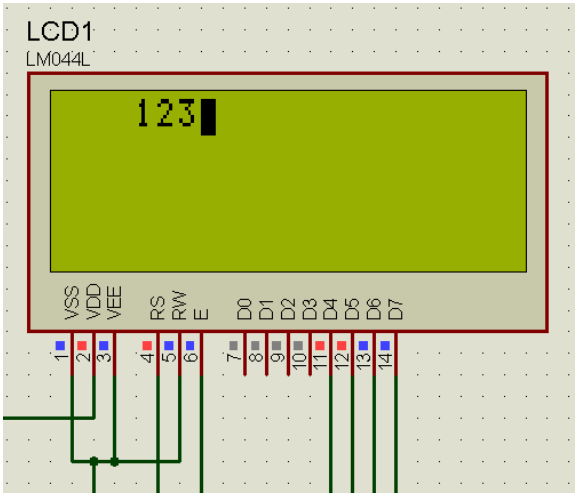
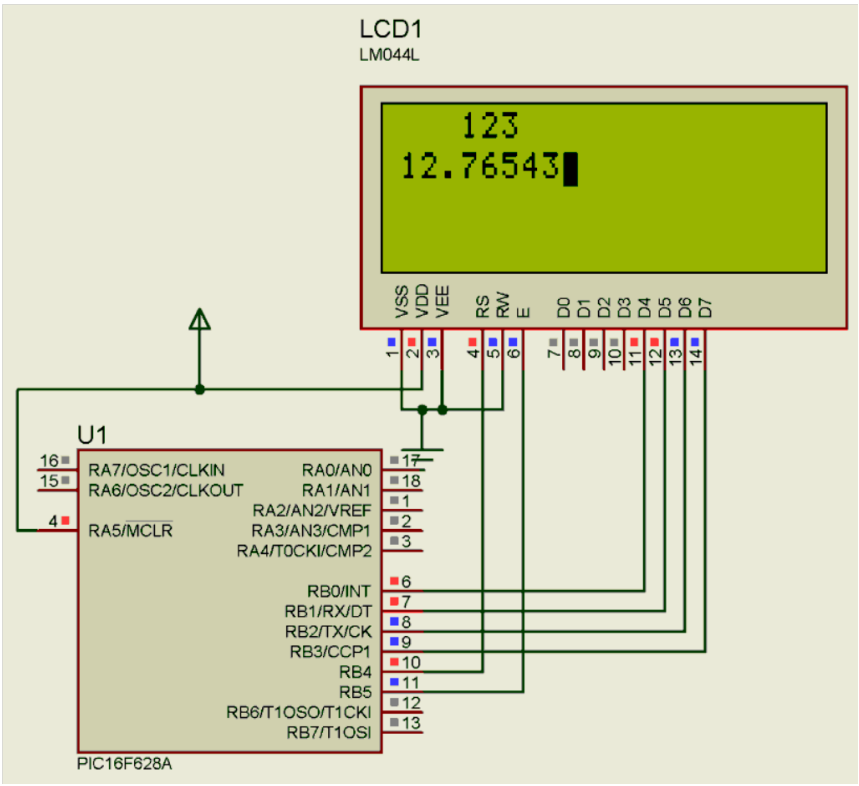


Figure 5-10

L'impression des nombres décimaux (avec virgule) peut être fait avec la fonction: *FloatToStr(**float** fnum, **char** *str);*. La philosophie de fonctionnement de cette fonction est identique à la précédente, qui fait des conversions d'entiers. Pour réaliser l'exemple suivant, on doit modifier la référence du microcontrôleur, cela est dû à la capacité de mémoire du PIC 16F84A, qui n'est pas suffisante pour les exercices suivants.

```
void main( void )
{
int ENTIER=123;                    //Déclaration d'une variable entière avec la valeur initiale
123.
float DECIMAL=12.76543;            //Déclaration d'une variable avec point décimal
                                   //initialisation à 12,76543.
char Text[20];                     // Chaine de caractères pour l'affichage des données.
Lcd_Init();                        //Initialisation du LCD.
IntToStr( ENTIER,Text );           //Fonction de conversion entière .
Lcd_Out(1,1,Text);                 //Impression du texte en sur l'écran LCD.
FloatToStr( DECIMAL,Text );        //Fonction de conversion décimale.
Lcd_Out(2,1,Text);                 // Impression du texte en sur l'écran LCD.
while(1)                           //Boucle infinie.
{
}
}
```

Après la simulation, on a le résultat sur la figure suivante:



Circuit 5-6

Le même processus peut être suivi pour autres types de variables telles que: *short* avec la fonction:

*ShortToStr(**short** input, **char** *output);*

Les variables *long* avec la fonction:

*LongToStr(**long** input, **char** *output);*

Des variables *unsigned short* avec la fonction:

*ByteToStr(**unsigned short** input, **char** *output);*

Variables *unsigned long* avec la fonction:

*LongWordToStr(**unsigned long** input, **char** *output);*

Les variables *unsigned int* avec la fonction:

*WordToStr(**unsigned int** input, **char** *output);*

6. Systèmes d'entrée des données

L'interaction de l'utilisateur avec les systèmes à microcontrôleurs nécessite des systèmes d'entrée de données. Pour ce faire, on utilise des périphériques tels que des interrupteurs, des claviers et même des claviers PS2 utilisés par les ordinateurs de bureau. Ce chapitre se concentre sur l'étude des boutons, interrupteurs et les Dip-Switch.

6.1 Utilisation de boutons

La mise en œuvre des boutons est l'un des alternatives les plus populaires dans les interfaces avec les utilisateurs. Les boutons sont simples à utiliser, et ils ont un coût, de mise en œuvre, économique. Le bouton-poussoir peut être normalement ouvert ou normalement fermé. L'implémentation de ces dispositifs est sujette aux effets de rebonds ou de bruit, quand ils changent d'état. Compte tenu du traitement rapide des microcontrôleurs, ces effets de bruit, font que le PIC peut détecter des changements et des états logiques indéfinis. Pour comprendre ce concept, on regarde le graphique ci-dessous qui montre le comportement de ce bruit:

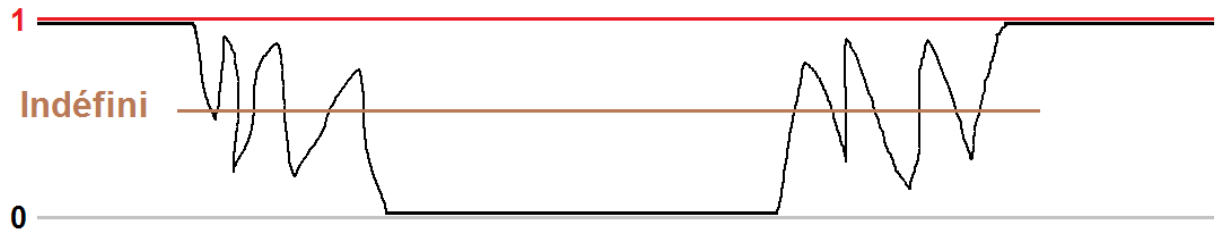
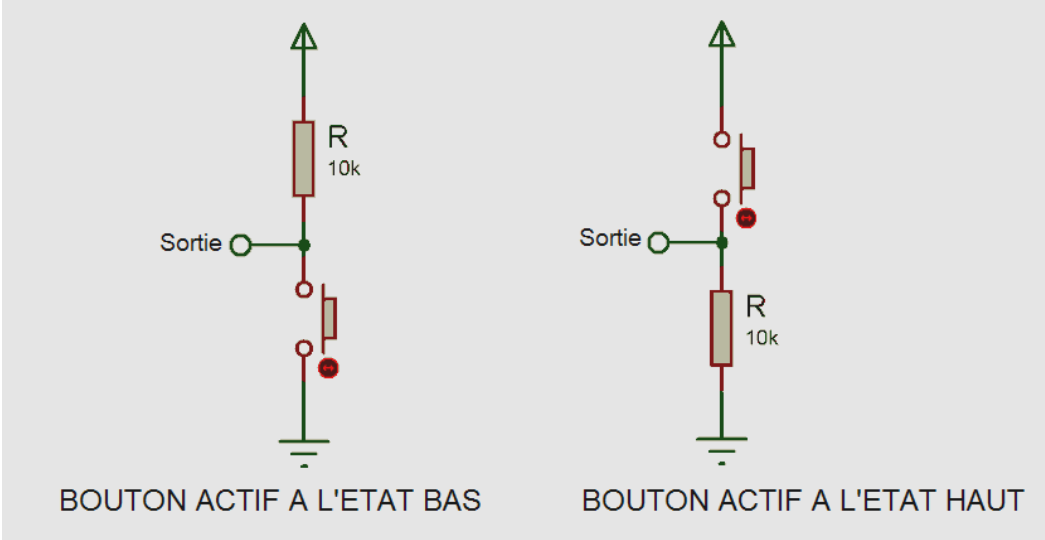


Figure 6-1

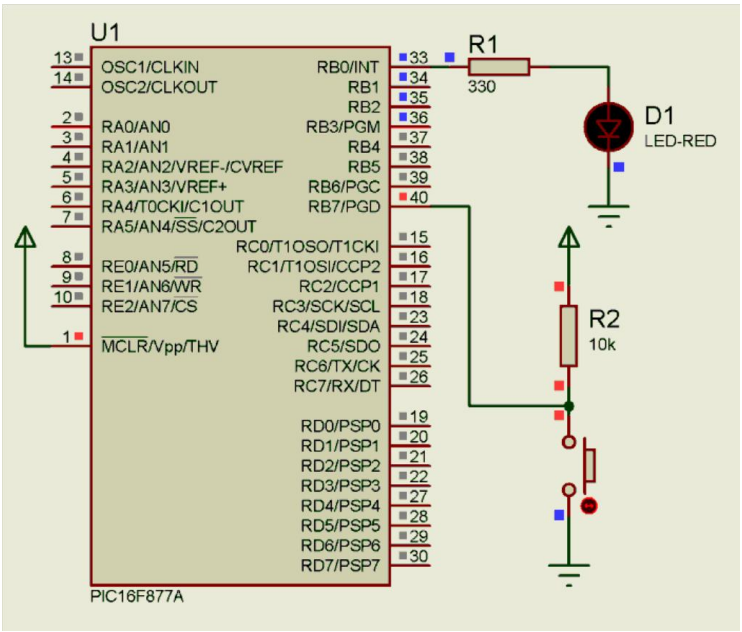
Lorsque des changements de tension traversent la zone indéfinie, génèrent des rebondissements de l'état haut à l'état bas et vice versa. Le microcontrôleur les détecte comme des impulsions. Pour éviter l'effet du bruit ou des rebonds, on devrait faire une temporisation en attendant la stabilisation de l'état du signal. La durée moyenne d'un rebond est de 1 à 5 millisecondes, indique qu'un retard doit être effectué supérieur à ce temps pour attendre la stabilisation. Un retard convenable à cet effet est supérieur ou égal à 10 millisecondes. Ce délai devrait s'appliquer après la détection du premier changement sur le bouton.

Pour le traitement des boutons et l'élimination des rebondissements, le compilateur MikroC PRO dispose d'une bibliothèque *Button*, qui peut être touvé dans la palette ou l'onglet des bibliothèques. Cette bibliothèque contient la seule fonction: ***unsigned short*** *Button(****unsigned short*** **port, **unsigned short** pin, **unsigned short** time, **unsigned short** active_state);* où *port* est le port où le bouton est connecté, *pin* est le bit du port où le bouton est connecté, *time* est le temps d'attente en millisecondes du bruit et *active_state* est l'état logique pour lequel on souhaite l'activation du bouton. La fonction retourne 0 si le bouton n'est pas actif et 255 si elle est active. Installation des boutons peut être effectuée de deux manières, actif à l'état haut ou actif à l'état bas, la figure suivante montre comment configurer les deux possibilités:



Circuit 6-1

La décision d'utiliser l'activation à l'état haut ou bas, dépend du développeur qui doit analyser la manière la plus simple au moment de la conception. On rappelle que dans tous les cas l'activation finale peut être inversée avec la fonction de la bibliothèque. Pour analyser et étudier l'utilisation de cette bibliothèque, on crée un nouveau projet avec les composants: PIC 16F877A, BUTTON, RES, et LED-RED. Le circuit correspondant dans ISIS est le suivant:



Circuit 6-2

Le programme respectif du microcontrôleur est le suivant:

```
void main( void )
{
TRISB=0xF0;                          //Configuration des ports.
PORTB=0;
while(1)                             //Boucle infinie
{
if( Button(&PORTB, 7, 100, 0) )      //Test de d'état du bouton sur RB7,activé à l'état bas
PORTB.F0=1;                          // Allumer le voyant si le bouton est actif.
else
```

```
PORTB.F0=0;                          //Eteindre la LED s'il est relaché.
}
}
```

A l'exécution de la simulation, la LED doit s'allumer lorsque le bouton est pressé, et s'éteindre lorsqu'il est relâché.

L'exemple suivant commute l'état de la LED :

```
void main( void )
{
TRISB=0xF0;                          //Configuration des ports.
PORTB=0;
while(1)                             //Boucle infinie.
{
if( Button(&PORTB, 7, 100, 0) )      //Test de l'état actif du bouton.
{
if( PORTB.F0==1 )                    // Commutation de  l'état de la LED.
PORTB.F0=0;
else
PORTB.F0=1;
                                     //Attente que le bouton soit no actif.
while( Button(&PORTB, 7, 100, 0) );
}
}
}
```

6.2 Utilisation des Dip-Switch

Les Dip-Switch, sont des dispositifs mécaniques contenant plusieurs interrupteurs dans un seul boitier. Ces dispositifs permettent de configurer de façon simple les caractéristiques binaires des systèmes à microcontrôleurs. Les Dip-Switch, sont disponibles dans le commerce dans des tailles, des couleurs, et des nombres d'interrupteurs différents. Leurs aspects physiques et leurs schémas dans le simulateur ISIS sont les suivants:

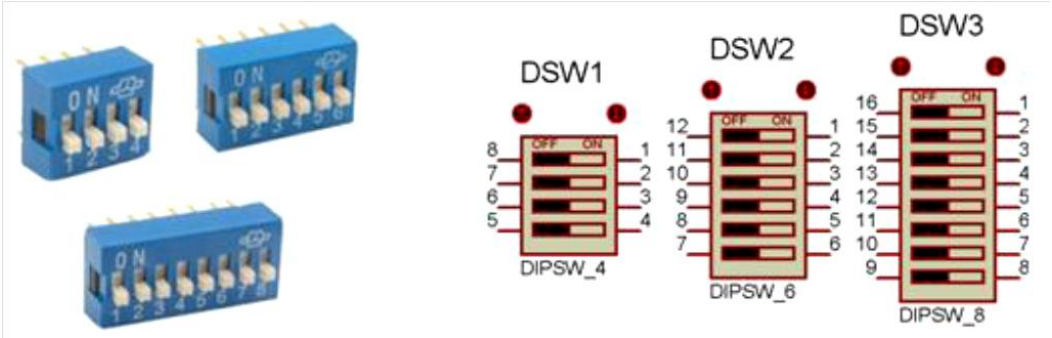
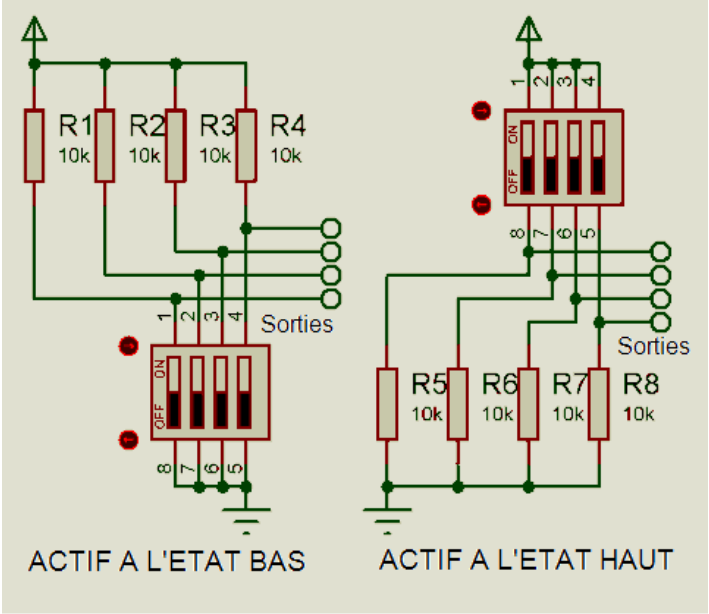


Figure 6-2

L'utilisation du Dip-Switch est similaire au bouton. Il peut être configuré de la même manière avec activation sur l'état haut ou bas. Ces configurations peuvent être appréciés dans le circuit suivant:

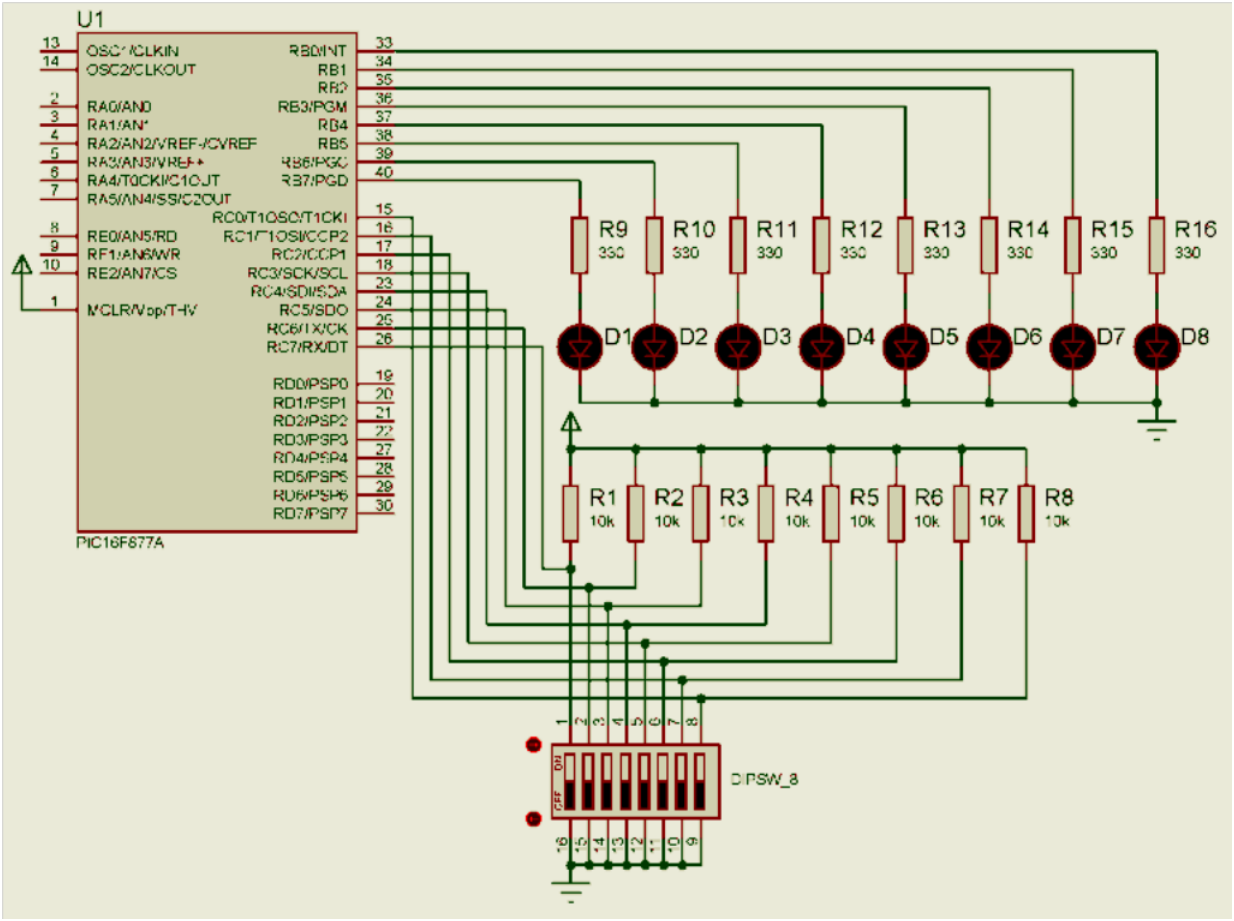


Circuit 6-3

Pour comprendre l'utilisation du Dip-Switch avec les microcontrôleurs PIC, on observe et on analyse l'exemple suivant:

```
void main( void )
{
TRISB = 0;                    //Configuration de ports.
PORTB = 0;
TRISC = 255;
while(1)                      //Boucle infinie.
{
PORTB = ~PORTC;               //Sortir dans le port B le complément du port C.
}
}
```

Pour exécuter la simulation dans ISIS, on utilise les dispositifs suivants: 16F877A, RES, LED-RED, DIPSW_8. Puis le circuit suivant est mis en œuvre dans ISIS:



Circuit 6-4

En exécutant la simulation, les LED montrent l'état du Dip-Switch.

7. Conversion AD et DA

La conversion analogique-numérique et numérique analogique est un processus par lequel on peut prendre ou délivrer un échantillon d'un signal continu de tension. L'utilisation des conversions est très utile pour le traitement numérique des signaux. La conversion analogique numérique, ou ADC, peut être faite avec des microcontrôleurs qui intègrent ce type de convertisseur. Le processus de conversion numérique-analogique est possible avec des éléments externes faciles à implémenter, même on peut effectuer cette conversion par des modules PWM incorporés dans certains des microcontrôleurs.

7.1 Conversion AD, ou ADC

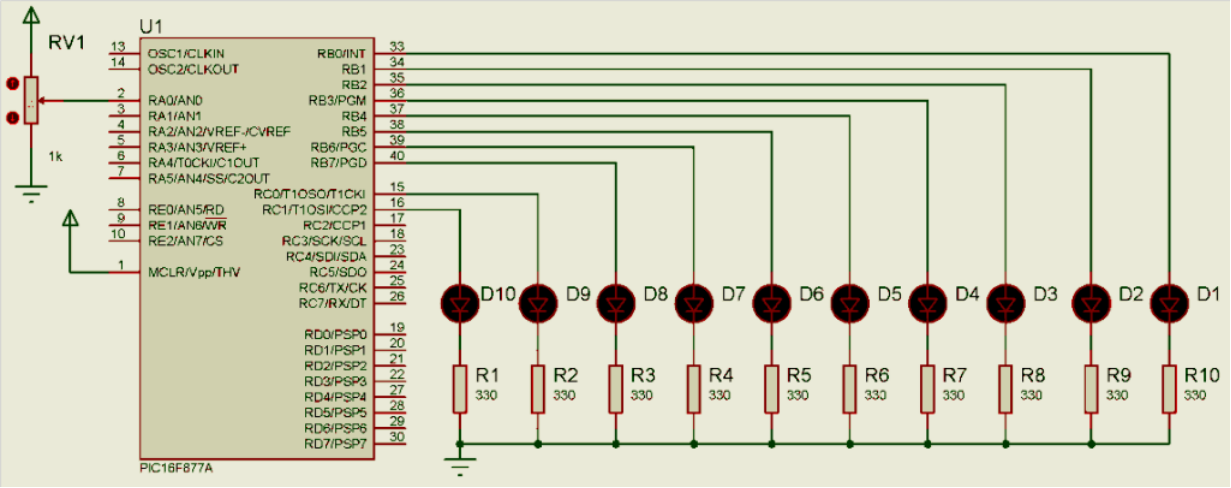
Ce processus est réalisé avec un convertisseur interne du microcontrôleur. Ce module est intégré dans la plupart des microcontrôleurs de moyenne et haute gamme. La conversion implémentée sur les PICmicro a une résolution de 10 bits, ce qui permet un nombre allant de 0 à 1023, proportionnel aux valeurs de référence, qui sont par défaut 0 volts et 5 volts.

Cela signifie que si une entrée analogique a une tension de 0 volt, le résultat est à 0, et si la tension est de 5 volts, le résultat de la conversion est de 1023, de la même façon si la tension est de 2,5 volts, le résultat est de 512. En fonction de la complexité d'un microcontrôleur, le PIC peut avoir jusqu'à huit entrées du signal analogique. Cependant, il est à noter que le microcontrôleur a un seul module interne de conversion, mais plusieurs canaux de lecture des tensions analogiques mais pas simultanément.

Pour effectuer ces conversions le compilateur MikroC PRO offre une bibliothèque prédéfinie : *ADC* pour faire la conversion. Cette bibliothèque contient une fonction appelée *ADC_Read(**unsigned short channel)*.* Cette fonction retourne le résultat de la conversion de la chaine spécifiée par le paramètre *channel*. Pour utiliser cette bibliothèque, on peut observer et analyser l'exemple suivant:

```
void main( void )
{
//Déclaration des variables.
unsigned int Donnee;
//Initialisation des ports.
TRISB = 0;
TRISC = 0;
PORTB = 0;
PORTC = 0;
while(1)                                    //Boucle infinie.
{
Donnee = ADC_Read(0);                       //Conversion sur le canal 0.
//Sortir les 8 bits de poids le plus faible sur le port B.
PORTB = Donnee&0xFF;
// Sortir les 2 bits de poids le plus fort sur le port C.
PORTC = (Donnee>>8)&0x03;
}
}
```

Pour simuler cet exemple, on doit utiliser les dispositifs suivant: 16F877A, RES, LED-RED, POT-HG, le circuit suivant sur ISIS :



Circuit 7-1

Après l'exécution de la simulation, les LED montrent le code binaire correspondant à la valeur de la conversion analogique-numérique de la tension appliquée par le potentiomètre à l'entrée analogique 0.

7.2 Conversion DA ou DAC

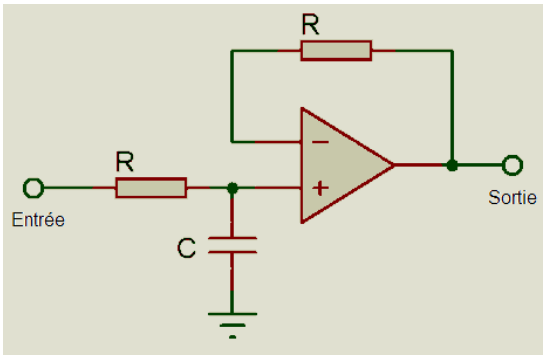
Cette conversion est possible grâce à deux stratégies, la première consiste à utiliser le module PWM du microcontrôleur, et la seconde est la mise en œuvre d'un ensemble externe de résistance pour obtenir la valeur analogique.

7.2.1 Conversion DA avec PWM

La conversion numérique-analogique, avec le module de PWM, consiste à prendre le signal modulé en largeur d'impulsion et d'effectuer la démodulation par l'intermédiaire d'un filtre passe-bas. Ce filtre doit avoir la fréquence de coupure très inférieure à la fréquence d'échantillonnage avec un rapport proche de 10, car son but est d'éliminer la fréquence d'échantillonnage. Le calcul de la fréquence de coupure de ce filtre est régi par la formule suivante:

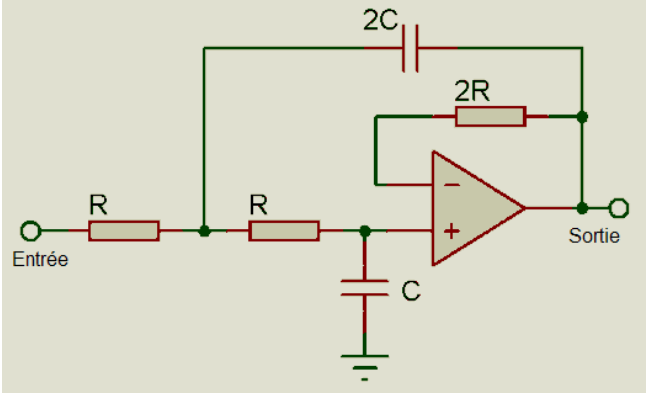
$$F_C = \frac{1}{2\pi RC} \qquad \textit{Équation 7-1}$$

Le circuit suivant met en œuvre un filtre passe-bas du premier ordre:



Circuit 7-2

L'implémentation d'un filtre passe bas second ordre :



Circuit 7-3

L'expression de la fréquence de coupure de ce filtre est :

$$F_C = \frac{1}{2\pi RC\sqrt{2}} \qquad \textit{Équation 7-2}$$

La fréquence attribuée à la porteuse PWM, doit être beaucoup plus grande que la fréquence du signal modulant. Pour expliquer le fonctionnement de cette conversion, on désigne 20 échantillons correspondant à une période d'un signal sinusoïdal. Les échantillons et la forme d'onde peut être vu dans la figure suivante:

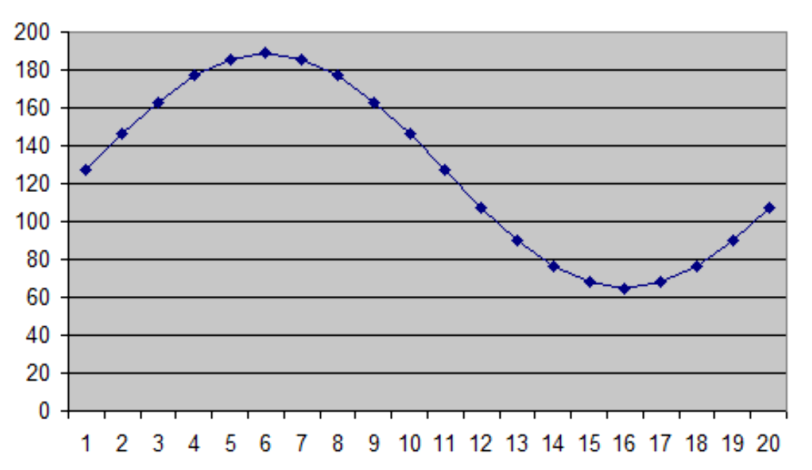


Figure 7-1

La mise en œuvre de la bibliothèque PWM, fait au moyen de quatre fonctions sont:

- *PWM1_Init(**const long** freq);*, cette fonction initialise le module PWM à la fréquence de la porteuse *freq*.
- *PWM1_Set_Duty(**unsigned short** duty_ratio);*, Cette fonction définit le rapport cyclique de travail par le paramètre *duty_ratio*, ce paramètre peut prendre des valeurs de 0 à 255, où 0 est 0% et 255 est 100% de la période de la porteuse.

La figure suivante illustre le comportement d'un signal PWM, selon la période de travail, la relation f_{pwm} et T_{pwm}:

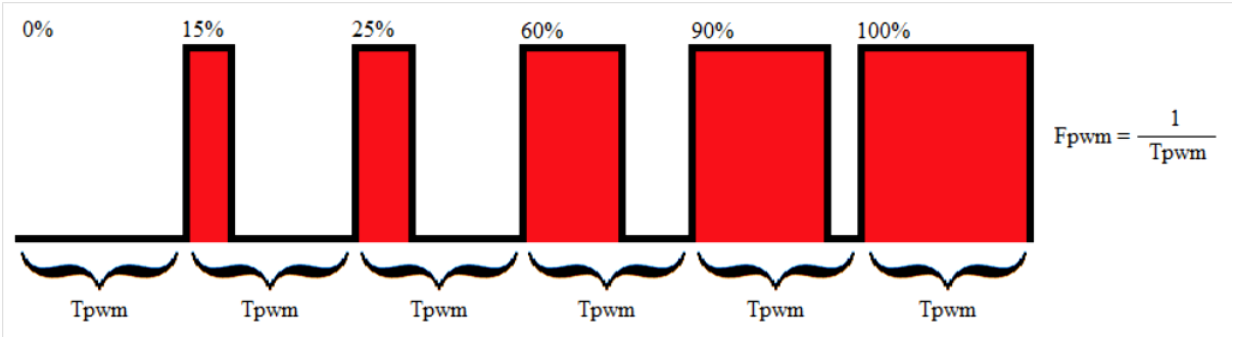


Figure 7-2

Enfin, il y a des fonctions: *PWM1_Start ()* et *PWM1_Stop ()*;, Qui active et désactive respectivement le signal PWM.

Pour la simulation doit créer un projet en MikroC PRO, avec le code source suivant:

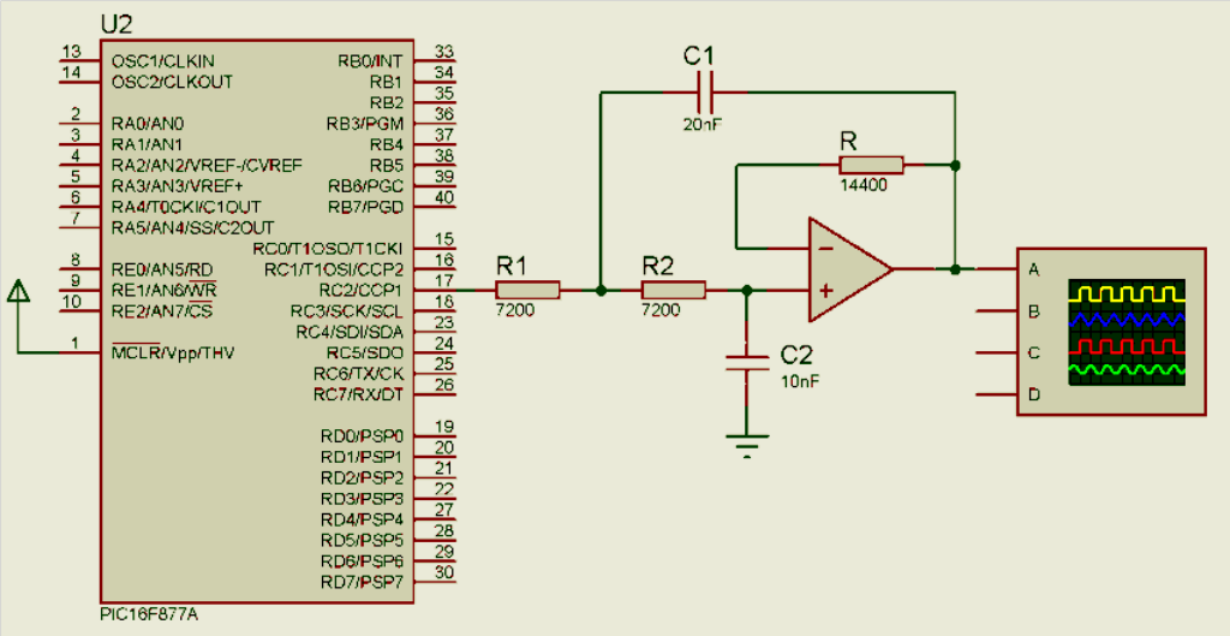
```
//Déclaration des constantes
//pour le signal sinus.
const unsigned short Sinus[20] =
{
127, 146, 163, 177, 185, 189, 185,
177, 163, 146, 127, 107, 90, 76,
68, 65, 68, 76, 90, 107
};
void main( void )
{
unsigned short n=0;                    //Déclaration des variables.
PWM1_Init(15625);                      //Configuration du module PWM à Fpwm=15.625K
Hz.

                                       //Initialisation de signal PWM.
PWM1_Start();
while(1)                               //Boucle infinie.
{
for( n=0; n<20; n++ )                  //Boucle pour parcourir les 20 échantillons
                                       //d'une période du signal sinus.
{

PWM1_Set_Duty( Sinus[n] );             //Modification du rapport cyclique du PWM.
delay_us(50);                          //Retard de 50us.

}
}
}
```

L'exécution de la simulation nécessite les dispositifs: 16F877A, RES, PAC, OP1P et des instruments virtuels: OSCILLOSCOPE. Ce dernier est un oscilloscope virtuel 4 canaux simultanés, idéal pour l'affichage des signaux analogiques et même numériques.

Ce circuit met en œuvre un filtre passe-bas pour éliminer f_{pwm} fréquence porteuse dans ce cas est 15.6KHz, le filtre a une fréquence de coupure d'environ 1,5 kHz.

Dans les applications futures, il est recommandé de calculer les filtres, définir une valeur arbitraire pour le condensateur C, entre 100 pF et 100nF et calculer la valeur de R, en utilisant les équations ci-dessus.

Avec les éléments ci-dessus, on construit le circuit suivant:



Circuit 7-4

Au cours de la simulation, on peut voir sur l'oscilloscope le signal analogique de forme sinusoïdale, produit de la reconstitution numérique du PIC, par l'intermédiaire du module PWM, et le filtre passe-bas. Le signal visualisé par l'oscilloscope peut être apprécié sur la figure suivante:

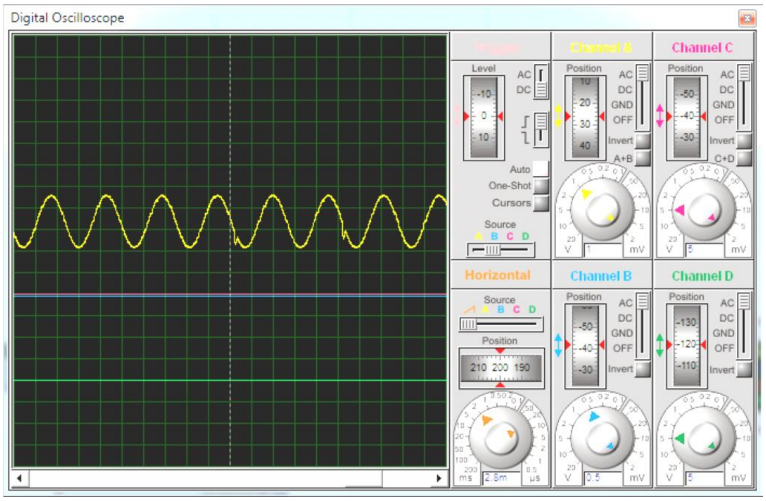


Figure 7-4

7.2.2 Conversion DA à réseau R- 2R :

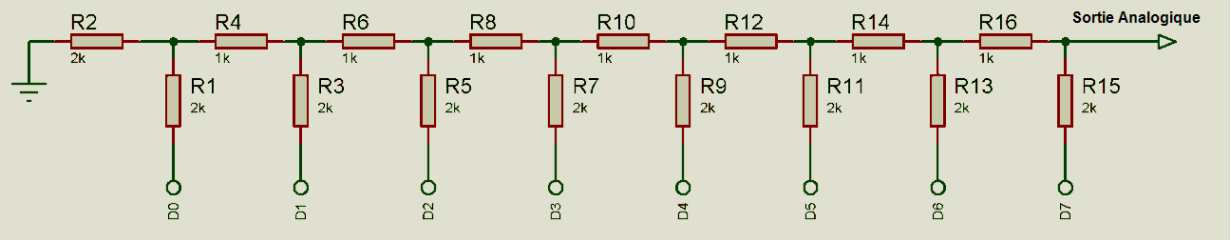
Le réseau des résistances R-2R, permet la conversion d'un nombre binaire en une valeur de tension proportionnelle. Cette disposition permet de mettre en œuvre un nombre indéterminé de bits, à la différence du module PWM, qui est limité à 8 bits. Cela signifie que, avec le réseau R-2R, on peut effectuer des conversions de 8, 16, 32, 64, ou d'un nombre de n bits en fonction du nombre de broches disponibles dans un microcontrôleur.

La mise en œuvre du convertisseur R-2R est facile à réaliser, car il est constitué d'un réseau de résistances où l'une est le double de l'autre, d'où le nom de R-2R.

Il est à noter que plus on utilise de bits, plus la résolution est meilleure, donc une plus grande qualité du signal reconstruit.

L'inconvénient notable de cette disposition est l'augmentation du matériel exigé, et par conséquent, l'utilisation d'un plus grand nombre de broches du port.

Dans le circuit ci-dessous, on peut voir la configuration des résistances pour une conversion de 8 bits:

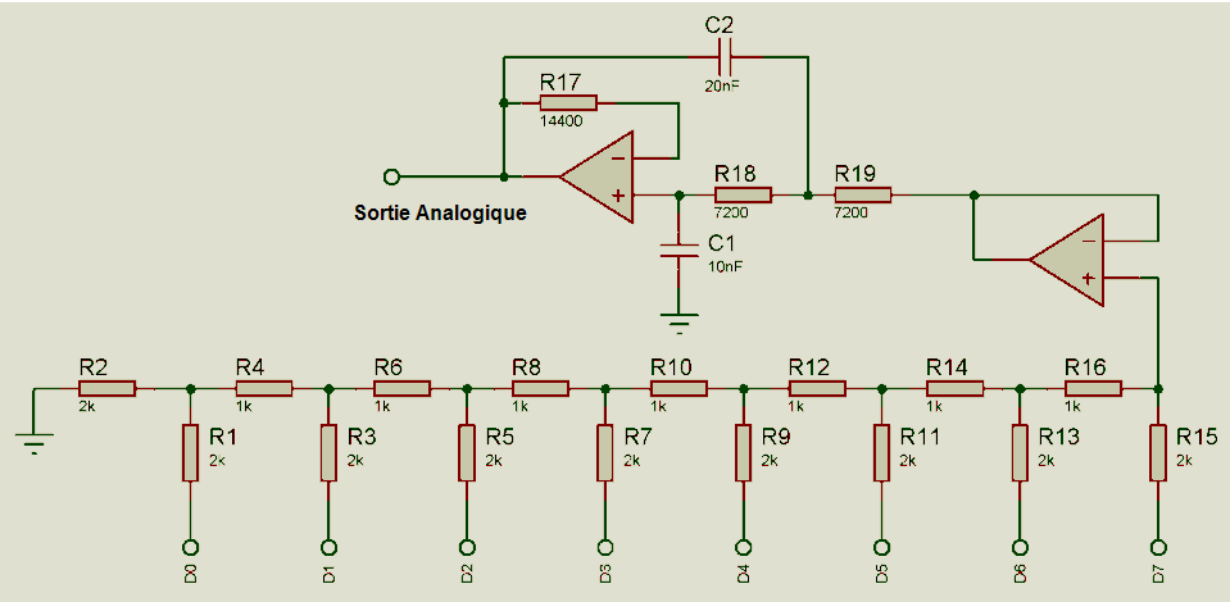


Circuit 7-5

Cette disposition peut être étendue avec la même architecture de réalisation pour un convertisseur d'une résolution plus élevée, ce qui augmente le nombre d'entrées D, qui est le même nombre de bits. Une autre caractéristique de ce type de conversion est qu'il ne nécessite pas des bibliothèques spécialisées, il suffit simplement de placer la valeur numérique à convertir sur un port, et la valeur analogique sera présente dans la sortie.

De la même façon qu'on a procédé pour la conversion avec PWM, il est important de supprimer la fréquence d'échantillonnage pour éviter les composantes de signal.

Le schéma complet du convertisseur avec un suiveur pour l'adaptation d'impédances, suivi d'un filtre passe bas de fréquence de coupure 1.5KHz:



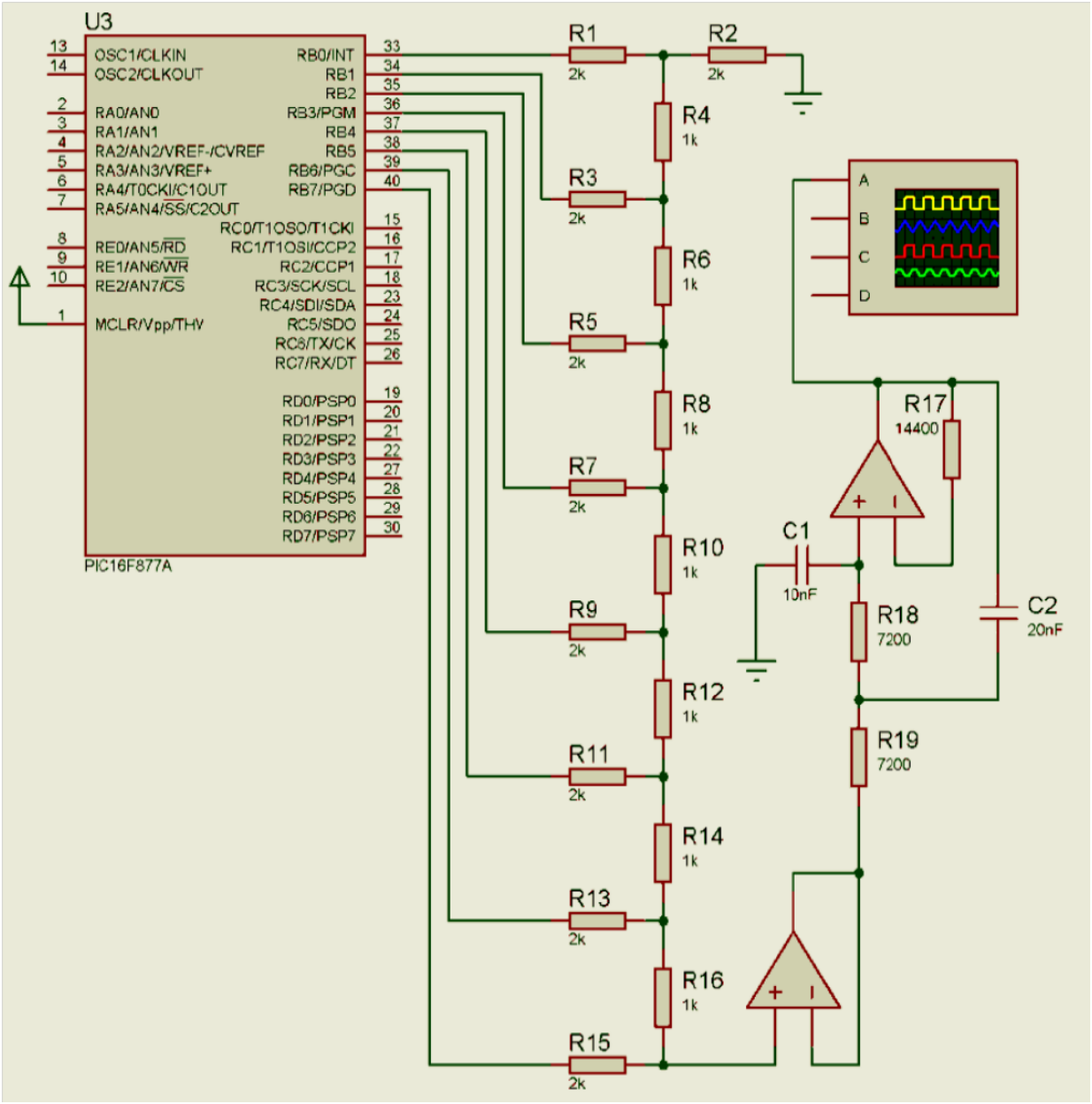
Circuit 7-6

Pour montrer l'application de cette technique de conversion, on modifie l'exemple de la conversion PWM, à cet effet, on observe et on analyse le code source suivant:

```
const unsigned short Sinus[20] =          //Déclaration des constantes pour le signal sinus.
{
127, 146, 163, 177, 185, 189, 185,
177, 163, 146, 127, 107, 90, 76,
68, 65, 68, 76, 90, 107
};
```

```
void main( void )
{
unsigned short n=0;                    //Déclaration des variables.
TRISB = 0;                             //Configuration des ports.
PORTB = 127;
while(1)                               //Boucle infinie.
{
for( n=0; n<20; n++ )                  //Boucle pour parcourir les 20 échantillons
                                       //d'une période du signal sinus.
{

PORTB = Sinus[n];                      //Modification de la valeur de l'échantillon sortie sur
                                       le //port B.
delay_us(50);                          //Retard de 50us.
}
}
}
```

On édite et on compile ce programme, on procède à la simulation dans ISIS, du circuit suivant:



Circuit 7-7

Le résultat attendu est cette simulation est similaire à la simulation du convertisseur avec PWM, la sortie de l'oscilloscope devrait montrer une courbe identique.

8. Les PICs

8.1 Les microcontrôleurs

Un microcontrôleur est un composant électronique Autonome doté :

- ✓ d'un microprocesseur,
- ✓ de la mémoire RAM,
- ✓ de la mémoire permanente
- ✓ des interfaces d'E/S parallèle, série (RS232, I2C …)
- ✓ des interfaces d'E/S analogique
- ✓ des Timer pour gérer le temps
- ✓ d'autres modules plus au moins sophistiqués selon la taille des µC.

Le microcontrôleur est généralement moins puissant qu'un microprocesseur en terme de rapidité ou de taille mémoire, il se contente le plus souvent d'un bus 8 ou 16 bits. Ceci en fait un composant très bon marché. Il est parfaitement adapté pour piloter les applications embarquées dans de nombreux domaines d'application.

On retrouve un microcontrôleur (± puissant) dans chaque équipement électronique :

- ✓ Informatique (souris, modem …)
- ✓ Vidéo (Appareil photos numérique, caméra numérique …)
- ✓ Contrôle des processus industriels (régulation, pilotage)
- ✓ Appareil de mesure (affichage, calcul statistique, mémorisation)
- ✓ Automobile (ABS, injection, GPS, airbag)
- ✓ Multimédia (téléviseur, carte audio, carte vidéo, MP3, magnétoscope)
- ✓ Téléphones (fax, portable, modem)
- ✓ Electroménager (lave-vaisselle, lave-linge, four micro-onde)

Un microcontrôleur peut être programmé une fois pour toutes afin qu'il effectue une ou des tâches précises au sein d'un appareil électronique. Mais les µC récents peuvent être reprogrammés et ceci grâce à leur mémoire permanente de type FLASH (d'où le terme flasher un appareil).

Plusieurs Constructeurs se partagent le marché des microcontrôleurs, citons INTEL,MOTOROLA, AMTEL, ZILOG, PHILIPS et enfin MICROCHIP avec ses PICs très populaires.

Les microcontrôleurs, quelque soit leurs constructeurs, ont des architecture très similaires et sont constitués de modules fondamentaux assurant les mêmes fonctions : UAL, Ports d'E/S, interfaces de communications série, Interfaces d'E/S analogiques, Timer et horloge temps réels… On peut dire que seul le langage de programmation (Assembleurs) constitue la différence majeure entre deux microcontrôleurs (similaires) venant de deux constructeurs différents.

8.2 Les PICs de Microchip

Les PICs sont des microcontrôleurs à architecture RISC (Reduce Instructions Construction Set), ou encore composant à jeu d'instructions réduit. L'avantage est que plus on réduit le nombre d'instructions, plus leur décodage sera rapide ce qui augmente la vitesse de fonctionnement du microcontrôleur.

Les PICs sont subdivisées 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 la 16F84 et la 16F877), et la famille High-End, qui utilise des mots de 16 bits.

Les PICs sont des composants STATIQUES, Ils peuvent fonctionner avec des fréquences d'horloge allant du continu jusqu'à une fréquence max spécifique à chaque circuit.

On se limite dans ce document à la famille Mid-Range et particulièrement aux PIC 16F84/877, sachant que si on a tout assimilé, on pourra facilement passer à une autre famille, et même à un autre microcontrôleur.

8.3 Identification d'un PIC

Pour identifier un PIC, on utilise simplement son numéro.

Les 2 premiers chiffres indiquent la catégorie du PIC, 16 indique une PIC Mid-Range.

Vient ensuite parfois une lettre L : Celle-ci indique que le PIC peut fonctionner avec une plage de tension beaucoup plus tolérante.

Ensuite, on trouve l'une des indications suivantes :

- C indique que la mémoire programme est une EPROM ou plus rarement une EEPROM
- CR pour indiquer une mémoire de type ROM
- F pour indiquer une mémoire de type FLASH.

On note à ce niveau que seule une mémoire FLASH ou EEPROM est susceptible d'être effacée.

Puis on trouve les derniers chiffres identifiants précisément le PIC. (84/877)

Enfin on voit sur les boîtiers « -XX » dans laquelle XX représente la fréquence d'horloge maximale que le PIC peut recevoir. Par exemple –04 pour une 4MHz.

Donc, un 16F84-04 est un PIC Mid-Range (16) dont la mémoire programme est de type FLASH (F) donc réinscriptible, de type 84 et capable d'accepter une fréquence d'horloge de 4MHz.

Une dernière indication que on trouve est le type de boîtier. Le boîtier PDIP par exemple est un boîtier DIL 18 broches, avec un écartement entre les rangées de 0.3''.

9. Le PIC 16F84

9.1 Organisation du 16F84

Les microcontrôleurs de la société Microchip les plus rencontrées sont les 16F84 et 16F87X. Leur structure est du type HARVARD et se composent d'une CPU à architecture RISC 8 bits.
Le 16F84, en boîtier DIP 18 broches, peut fonctionner à une fréquence maximale de 10 MHz pour un cycle d'instruction de 400ns. Il est constitué des éléments suivants :

- ✓ 1 K mots de 14 bits de mémoire programme du type Flash,
- ✓ 68 octets de RAM,
- ✓ 64 octets d'EEPROM,
- ✓ 2 ports d'E/S (un de 8 bits et un de 5 bits),
- ✓ Un Timer 8 bits,
- ✓ Un Watchdog,
- ✓ Une interruption externe.

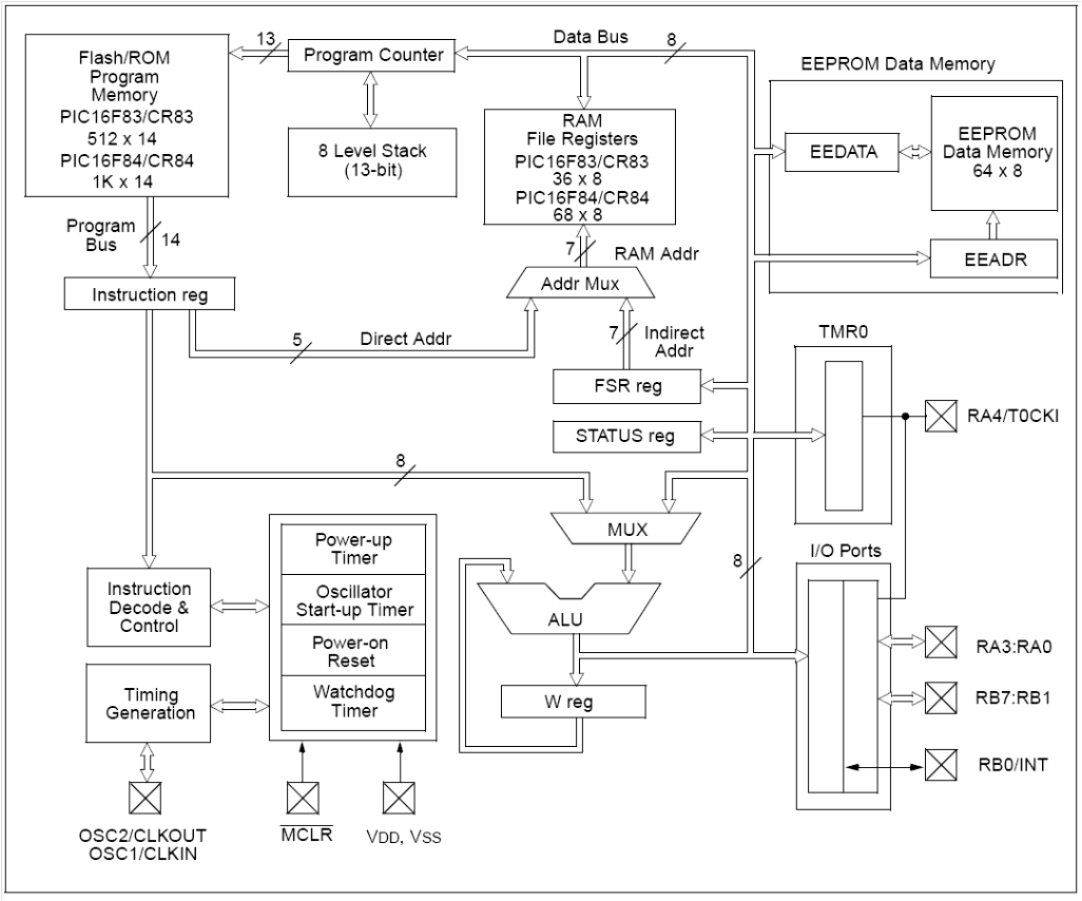


Figure 9-1

9.2 La mémoire du PIC

La mémoire du PIC 16F84 est divisée en 3 parties : la mémoire programme, la RAM et l'EEPROM.

La mémoire programme

La mémoire programme est constituée de 1K mots de 14 bits. C'est dans cette zone qu'on va écrire le programme. Ceci fait 2Kbytes maximum sur PC.

La mémoire EEPROM

La mémoire EEPROM (Electrical Erasable Programmable Read Only Memory), est constituée de 64 octets qu'on peut lire et écrire depuis un programme. Ces octets sont conservés après une coupure de courant et sont très utiles pour conserver des paramètres semi-permanents. Leur utilisation implique une procédure spéciale qu'on verra par la suite, car ce n'est pas de la RAM, mais bien une ROM de type spécial. Il est donc plus rapide de la lire que d'y écrire.

La mémoire RAM

La mémoire RAM est celle qui est très utilisée. Toutes les données qui y sont stockées sont perdues lors d'une coupure de courant. La mémoire RAM est organisée en 2 banques pour la 16F84. La RAM est subdivisée de plus en deux parties. Dans chacune des banques on trouve des « cases mémoires spéciales » appelées REGISTRES SPECIAUX et des cases mémoires « libres » dont on peut se servir.

9.3 L'Horloge

L'horloge peut être soit interne soit externe. L'horloge interne est constituée d'un oscillateur à quartz ou d'un oscillateur RC. Avec l'oscillateur à Quartz, on peut avoir des fréquences allant jusqu'à 10 MHz selon le type de µC.

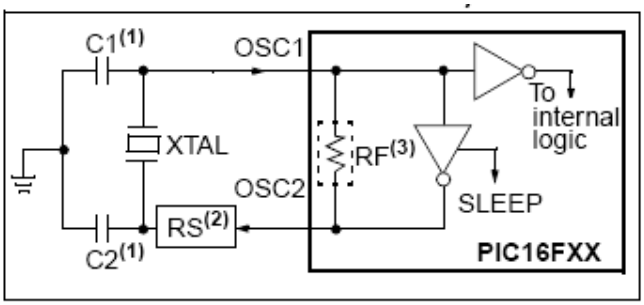


Figure 9-2

Le filtre passe bas (Rs, C1, C2) limite les harmoniques dus à l'écrêtage et Réduit l'amplitude de L'oscillation, il n'est pas obligatoire.

Avec un oscillateur RC, la fréquence de l'oscillation est fixée par VDD, Rext et Cext. Elle peut varier légèrement d'un circuit à l'autre.

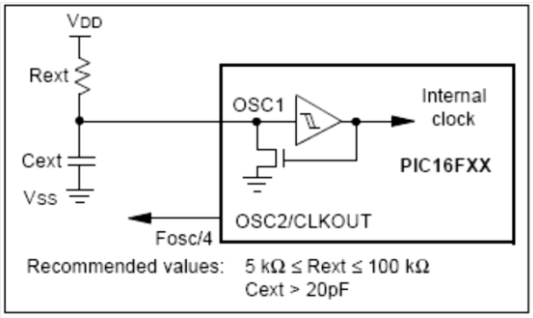


Figure 9-3

Dans certains cas, une horloge externe au microcontrôleur peut être utilisée pour synchroniser la PIC sur un processus particulier.

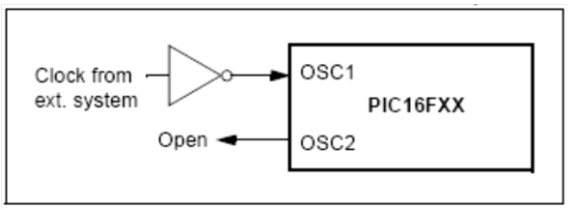


Figure 9-4

Quelque soit l'oscillateur utilisé, l'horloge système dite aussi horloge instruction est obtenue en divisant la fréquence par quatre. Par la suite on utilisera le terme Fosc/4 pour désigner l'horloge système. Avec un quartz de 4 MHz, on obtient une horloge instruction de 1 MHz, soit le temps pour exécuter une instruction de 1µs.

9.4 L'ALU et l'accumulateur W

L'ALU est une Unité Arithmétique et logique 8 Bits qui réalise les opérations arithmétiques et logique de base. L'accumulateur W est un registre de travail 8 bits, toutes les opérations à deux opérandes passent par lui. On peut avoir :

- ✓ Une instruction sur un seul opérande qui est en général un registre situé dans la RAM ;
- ✓ Une instruction sur 2 opérandes. Dans ce cas, l'un des deux opérandes est toujours l'accumulateur W, l'autre peut être soit un registre soit une constante.

Pour les instructions dont un des opérandes est un registre, le résultat peut être récupéré soit dans l'accumulateur, soit dans le registre lui-même.

9.5 Brochage du PIC

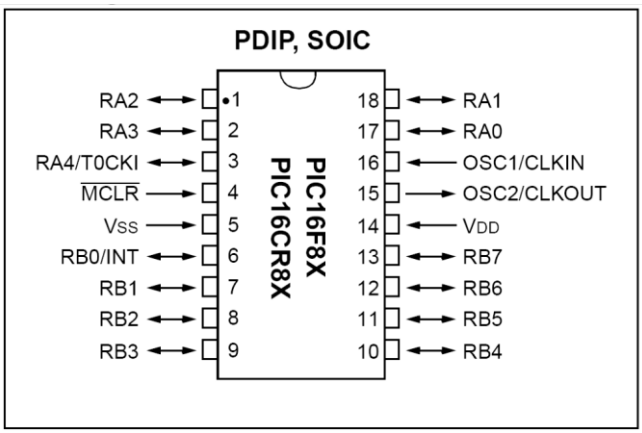


Figure 9-5

- VSS et VDD : broches d'alimentation (3 à 5,5V).
- OSC1 et OSC2 : signaux d'horloges, ces broches peuvent recevoir un circuit RC ou un résonateur.
- CLKIN : peut être connectée à une horloge externe (0 à 4, 10 ou 20 MHz).
- MCLR : Reset (Master Clear).
- RA0, ... , RA4 : 5 entrées/sorties du port A.
- RB0, ... , RB7 : 8 entrées/sorties du port B.
- T0CKI : Entrée d'horloge externe du Timer TMR0.
- INT : entrée d'interruption externe.

9.6 Les registres

Chaque registre provoque un fonctionnement spécial de la PIC ou la mise en service d'une fonction particulière.

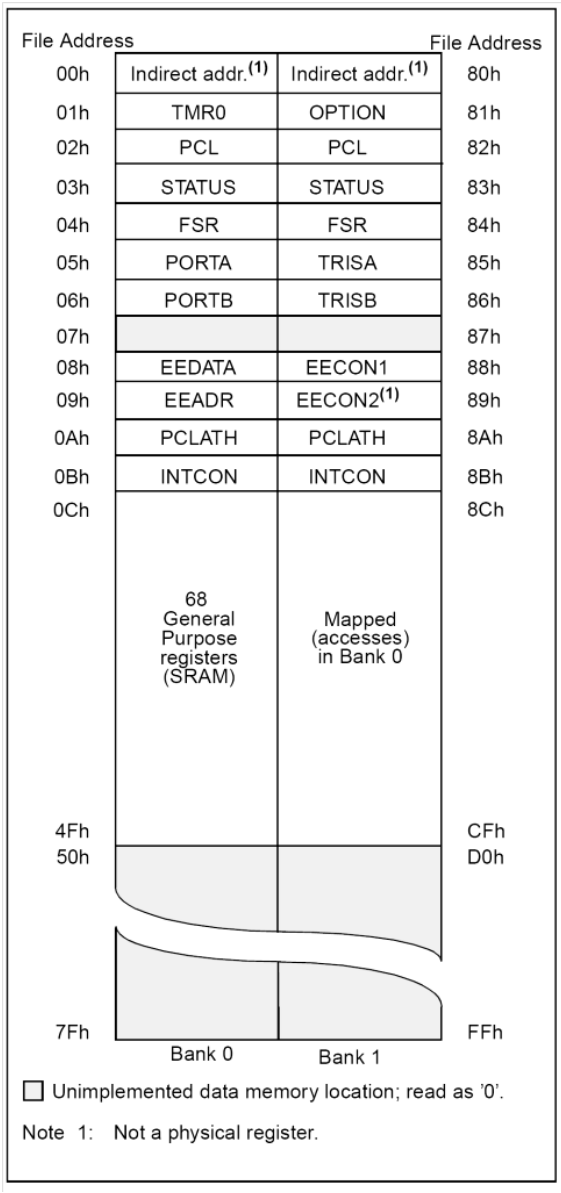


Figure 9-5

Certains registres sont identiques dans les 2 banques (FSR par exemple). Cela signifie qu'y accéder depuis la banque 0 ou 1 ne fait pas de différence.

- ✓ La banque 0 utilise les adresses de 0x00 à 0x7F,
- ✓ la banque 1 allant de 0x80 à 0xFF.
- ✓ Les zones en grisé sont des emplacements non utilisés (et non utilisables).
- ✓ L'emplacement 0x00 est un emplacement auquel on ne peut pas accéder.

Pour la grande majorité des registres, chaque bit a une fonction spéciale.

9.6.1 Registre d'état

Le registre d'état STATUS, son contenu est le suivant :

STATUS	IRP	RP1	RP0	/T0	/PD	Z	DC	C
03	7	6	5	4	3	2	1	0

Adresse 03 ou 83H

STATUS.7 IRP est un bit destiné à permettre l'accès à plus de 256 mots de RAM par adressage indirect, il n'est pas utilisé par le 16C84

STATUS RP1-RP0 Sont les deux bits de sélection de la PAGE en RAM, seul 6-5 RP0 est utilisé par le 16C84. Le second peut être utilisé comme bit d'usage général mais cela est déconseillé pour des raisons de compatibilité des Logiciels avec d'autres boîtiers de la famille 16CXX.

STATUS.4 /TO (Time Out) Est mis à 1 lors de la mise sous tension ou de la remise à zéro du chien de garde, il est basculé à 0 si le chien de garde déborde.

STATUS.3 /PD (Power Down) Est mis à 1 à la mise sous tension, à 0 par une instruction SLEEP

STATUS.2 Z (Zéro) Mis à 1 si le résultat d'une opération est nulle.

STATUS.1 DC (Digit Carry) C'est la retenue intermédiaire de l'arithmétique DCB

STATUS.0 C Est la retenue en addition ou soustraction.

Tableau 9-1

9.6.2 Registres du Timer

Adresse 01	**TMR0** est le compteur utilisé par le TIMER et le chien de garde
Adresse 02 ou 82H	**PCL**: compteur programme ou plus exactement l'octet bas de l'adresse dans la ROM de programme. Les 5 bits supplémentaires sont contenus dans le registre **PCLATH** d'adresse 0A (ou 8AH).

Tableau 9-2

9.6.3 Registres des ports

Adresse 05 ou 85H	le PORT_A et 85H le registre de direction TRIS_A de ce port (1=entrée,0=sortie)
Adresse 06 ou 86H	Le PORT_B et 86H le registre de direction TRIS_B

Tableau 9-3

9.7 Les interruptions

Le 16F84 dispose de 4 sources d'interruptions :

- Une source externe via la broche RB0/INT
- Le débordement du TIMER
- un changement de l'état des bornes 4 à 7 du port B
- La programmation de l'EEPROM de données

• Registre INTCON

Les interruptions sont définies et autorisées à partir du registre INTCON

Adresse 0B ou 8B	**INTCON**	GIIE	EEIE	RTIE	INTE	RBIE	RTIF	INTF	RBIF
	0B	7	6	5	4	3	2	1	0
	INTCON.7 GIE est le bit d'autorisation de toutes les interruptions, au 0 aucune interruption n'est active, c'est son état au RESET. INTCON.6 EEIE valide l'interruption associée à l'EEPROM de données INTCON.5 RTIE Mis à 1 il autorise les interruptions associées au débordement du compteur du TIMER (RTCC) INTCON.4 INTE C'est la validation de l'interruption extérieure reçue sur RB0 INTCON.3 RBIE Valide ou non les interruptions provoquées par un changement d'état sur les fils 7 à 4 du port B. INTCON.2 RTIF S'il est mis à 1 ce bit indique qu'une interruption à été décelée sur l'entrée. INTCON.1 INTF. Ce bit détecte cette interruption même si elle n'est pas autorisée INTCON.0 RBIF Indique un changement d'état sur les 4 fils RB7-4.								

Tableau 9-4

9.8 L'EEPROM de données

Elle ne fait pas partie du champ mémoire normal et n'est accessible que par une procédure spéciale mettant en œuvre les registres EEDATA EEADR EECON1 et 2

Adresse 88H	EECON1				EEIF	WRERR	WREN	WR	RD
	88H	7	6	5	4	3	2	1	0
		EECON1.4 EEIF EEPROM Interrupt Flag est mis à 1 lorsqu'une écriture est terminée une interruption lui est associée, il ne faut pas oublier que l'écriture d'un octet dans l'EEPROM prend 10mS. EECON1.3 WREER Mis à 1 si une erreur s'est produite pendant l'écriture EECON1.2 WREN ce bit doit être mis à 1 pour autoriser une écriture, il est au 0 Au reset EECON1.1 Doit être forcé à 1 pour écrire une donnée, est remis à 0 par le logiciel lorsque l'écriture est achevée EECON1.0 RD Doit être mis à 1 pour lire un octet, est remis automatiquement au 0 par le circuit.								

Tableau 9-5

Procédure de lecture d'un octet en EEPROM :

1. Écrire l'adresse dans EEARD
2. Mettre à 1 le bit RD=EECON1.0
3. Lire le contenu de EEDATA

Procédure d'écriture dans l'EEPROM :

1. Charger l'adresse dans EEADR
2. Charger la donnée à écrire dans EEDATA
3. Exécuter le programme de sécurisation suivant :

→ Charger 55 dans W
→ Transférer W dans EECON2 (adresse 89H)
→ Charger AA dans W
→ Transférer W dans EECON2
→ Monter à 1 le bit WR de EECON1

Il n'existe pas de procédure d'effacement car toute écriture dans une case mémoire EEPROM efface son contenu précédent.

10. Le PIC 16F877

10.1 Les éléments de base du PIC 16F877

Les 16F87X, en boîtier DIP 28 ou 40 broches, peuvent fonctionner à une fréquence maximale de 20 MHz pour un cycle d'instruction de 200ns. Ils sont constitués des éléments suivants :

- ✓ Mémoire programme du type Flash peut atteindre 8 K mots de 14 bits,
- ✓ 64 à 256 octets d'EEPROM,
- ✓ 128 à 368 octets de RAM,
- ✓ des ports E/S,
- ✓ des Timers,
- ✓ Un convertisseur A / N 10 bits avec 5 ou 8 entrées multiplexées,
- ✓ Un chien de garde (WatchDog).
- ✓ Une interface série synchrone (SPI) ou asynchrone (USART).

PIC	FLASH	RAM	EEPROM	I/O	A/D	Port//	Port Série
16F870	2K	128	64	22	5	NON	USART
16F871	2K	128	64	33	8	PSP	USART
16F872	2K	128	64	22	5	NON	MSSP
16F873	4K	192	128	22	5	NON	USART/MSSP
16F874	4K	192	128	33	8	PSP	USART/MSSP
16F876	8K	368	256	22	5	NON	USART/MSSP
16F877	8K	368	256	33	8	PSP	USART/MSSP

Tableau 10-1

10.2 Organisation du 16F877

Le 16F877 dispose de 8Ko de mémoire Flash pour le programme, 368 octets de RAM et 256 octets d'EEPROM. Il possède 5 ports E/S (3 de 8 bits, un de 6 bits et un de 3 bits), ces 33 entrées/sorties sont configurables individuellement. Son convertisseur A/N est de 10 bits avec 8 entrées multiplexées. En plus des 3 Compteur/Timer de 8 bits, il possède 2 Timers 8 bits et un Timer 16 bits. Il a 15 sources d'interruption. Il dispose de deux interfaces série, une synchrone (SPI) et l'autre asynchrone (USART).

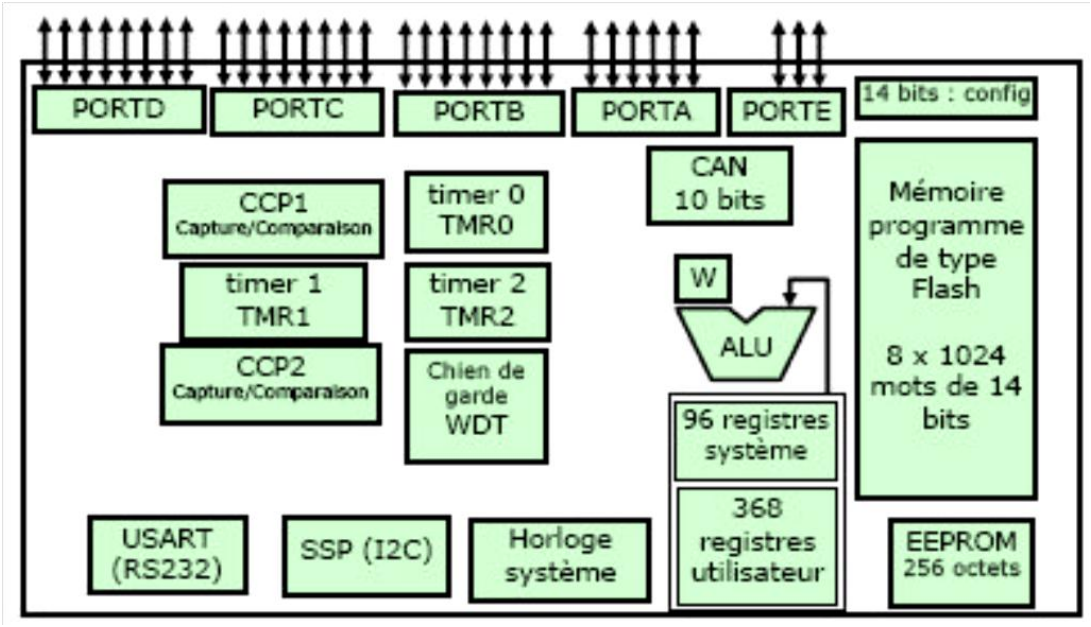


Figure 10-1

10.3 Organisation de la mémoire RAM

L'espace mémoire RAM adressable est de 512 positions de 1 octet chacune :

- ✓ 96 positions sont réservées au SFR (Special Function Registers) qui sont les registres de configuration de la PIC.

- ✓ Les 416 positions restantes constituent les registres GPR (General Propose Registers) ou RAM utilisateur. Sur le 16F876 et 16F877, 3 blocs de 16 octets chacun ne sont pas implantés physiquement d'où une capacité de RAM utilisateur de 368 GPR.

Pour accéder à la RAM, on dispose de deux modes d'adressage.

10.4 Registres

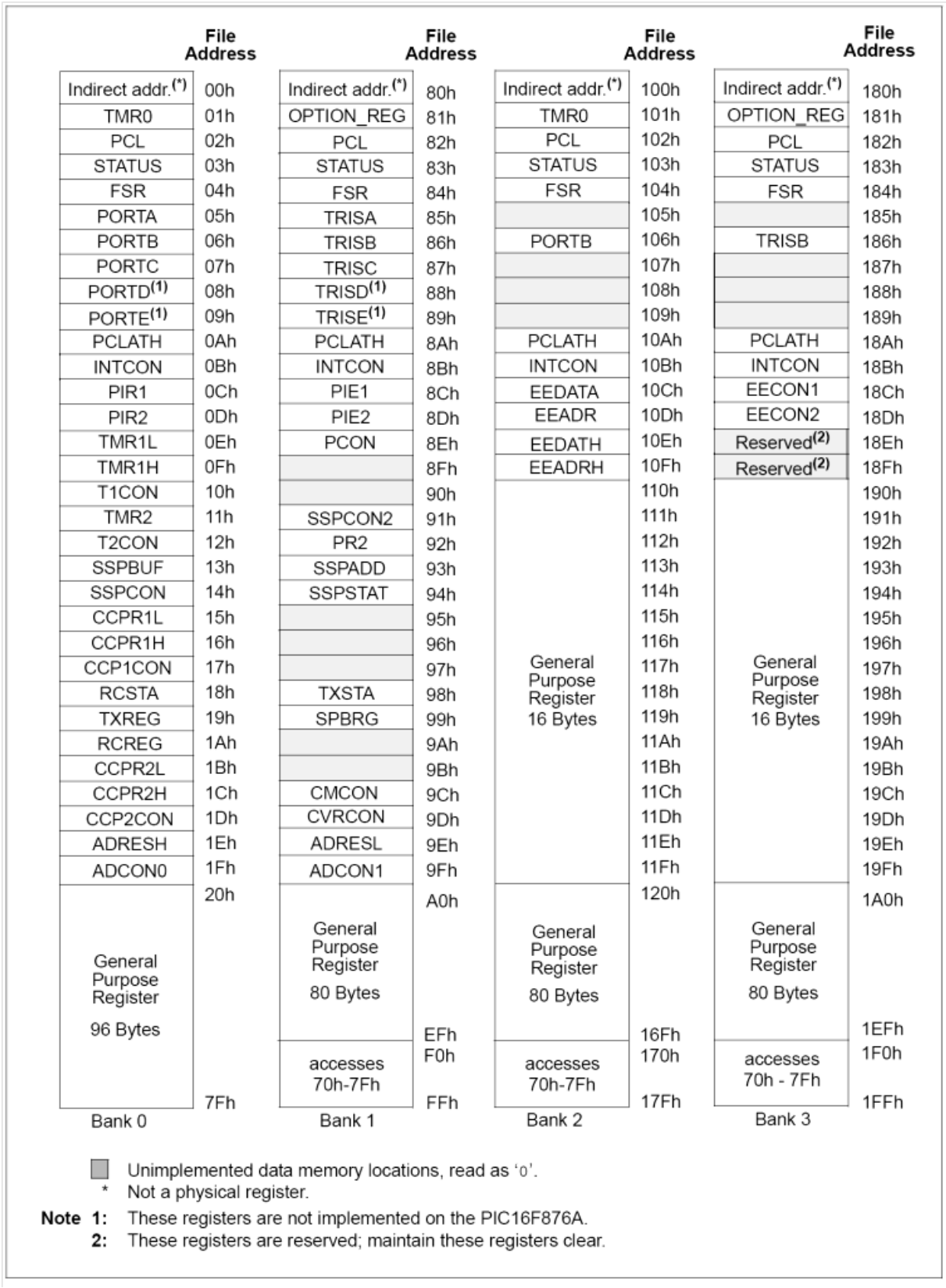

File Address
Indirect addr.(*) 00h
TMR0 01h
PCL 02h
STATUS 03h
FSR 04h
PORTA 05h
PORTB 06h
PORTC 07h
PORTD(1) 08h
PORTE(1) 09h
PCLATH 0Ah
INTCON 0Bh
PIR1 0Ch
PIR2 0Dh
TMR1L 0Eh
TMR1H 0Fh
T1CON 10h
TMR2 11h
T2CON 12h
SSPBUF 13h
SSPCON 14h
CCPR1L 15h
CCPR1H 16h
CCP1CON 17h
RCSTA 18h
TXREG 19h
RCREG 1Ah
CCPR2L 1Bh
CCPR2H 1Ch
CCP2CON 1Dh
ADRESH 1Eh
ADCON0 1Fh
20h
General Purpose Register
96 Bytes
7Fh
Bank 0
Indirect addr.(*) 80h
OPTION_REG 81h
PCL 82h
STATUS 83h
FSR 84h
TRISA 85h
TRISB 86h
TRISC 87h
TRISD(1) 88h
TRISE(1) 89h
PCLATH 8Ah
INTCON 8Bh
PIE1 8Ch
PIE2 8Dh
PCON 8Eh
8Fh
90h
SSPCON2 91h
PR2 92h
SSPADD 93h
SSPSTAT 94h
95h
96h
97h
TXSTA 98h
SPBRG 99h
9Ah
9Bh
CMCON 9Ch
CVRCON 9Dh
ADRESL 9Eh
ADCON1 9Fh
A0h
General Purpose Register
80 Bytes
EFh
F0h
accesses 70h-7Fh
FFh
Bank 1
Indirect addr.(*) 100h
TMR0 101h
PCL 102h
STATUS 103h
FSR 104h
105h
PORTB 106h
107h
108h
109h
PCLATH 10Ah
INTCON 10Bh
EEDATA 10Ch
EEADR 10Dh
EEDATH 10Eh
EEADRH 10Fh
110h
111h
112h
113h
114h
115h
116h
117h
118h
119h
11Ah
11Bh
11Ch
11Dh
11Eh
11Fh
General Purpose Register
16 Bytes
120h
General Purpose Register
80 Bytes
16Fh
170h
accesses 70h-7Fh
17Fh
Bank 2
Indirect addr.(*) 180h
OPTION_REG 181h
PCL 182h
STATUS 183h
FSR 184h
185h
TRISB 186h
187h
188h
189h
PCLATH 18Ah
INTCON 18Bh
EECON1 18Ch
EECON2 18Dh
Reserved(2) 18Eh
Reserved(2) 18Fh
190h
191h
192h
193h
194h
195h
196h
197h
198h
199h
19Ah
19Bh
19Ch
19Dh
19Eh
19Fh
General Purpose Register
16 Bytes
1A0h
General Purpose Register
80 Bytes
1EFh
1F0h
accesses 70h - 7Fh
1FFh
Bank 3
Unimplemented data memory locations, read as '0'.
* Not a physical register.
Note 1: These registers are not implemented on the PIC16F876A.
2: These registers are reserved; maintain these registers clear.

Figure 10-2

10.5 Le convertisseur A/N

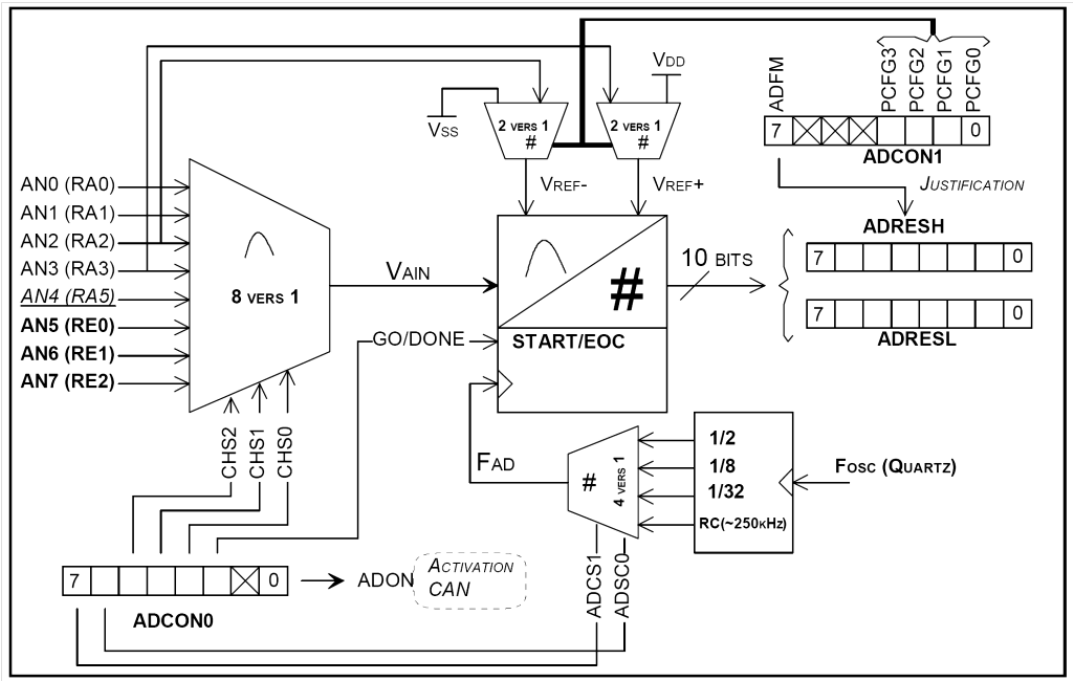


Figure 10-3

La PIC 16F877 possède 8 entrées analogiques (RA0..RA3, RA5 et RE0..RE2 pour les PICs disposant du port E)
Ces entrées sont multiplexées vers un CAN à approximation successive (SAR = Successive Approximation Register). Celui-ci a une résolution de 10 bits.
Le CAN est précédé d'un échantillonneur/bloqueur permettant une stabilité de la tension d'entrée pendant toute la durée de la conversion.

11. Les capteurs

La mise en œuvre de capteurs dans les systèmes à microcontrôleurs est utile dans de nombreuses applications. Les capteurs permettent d'obtenir des lectures des grandeurs physiques telles que la température, l'accélération, la pression, l'humidité, la vitesse du vent, la luminosité, le contraste entre autres. Son application est importante dans les systèmes de contrôle, de la robotique et de l'instrumentation.

11.1 Capteur de température LM35

Le capteur LM35 est un dispositif actif à trois bornes permettant de mesurer des plages de température ambiante de -55 à 150 degrés Celsius ou centigrades. Ce dispositif est facile à mettre en œuvre car il ne comporte que deux bornes de polarisation et une sortie de tension directement proportionnel à de la température. Ce capteur peut être alimenté de 4V à 30 V et possède une sortie de 10mV par degré Celsius. L'aspect physique du capteur, son brochage et comment il se présente à ISIS, sont illustré dans la figure suivante:

Figure 11-1

Pour effectuer la lecture de la tension de sortie du capteur, on met en œuvre le module CAN du microcontrôleur. La sortie maximale du capteur est de 1,5V, lorsque la température est de 150 degrés Celsius. Il est donc important de changer la valeur de la référence positive du convertisseur analogique-numérique, afin d'améliorer la résolution de la mesure de tension. Pour l'exemple suivant la tension de référence positive de l'ADC sera fixée à 2,5 volts. Prendre la référence positive 2,5V, le convertisseur donne un résultat binaire 1023 lorsque la tension à convertir est de 2,5V. Dans le cas de ce capteur, va être défini par les relations suivantes:

$$\frac{1023}{2{,}5} = \frac{Radc}{Vadc} \quad équation\ 11-1$$

Où Radc est le résultat binaire de conversion AN. A partir de cet équation, on peut déduire que la tension Vadc lue par le convertisseur analogique-numérique est le suivant:

$$Vadc = \frac{2{,}5.Radc}{1023} \quad équation\ 11-2$$

Travailler avec la relation du capteur : 10mV/°C, on peut se référer à l'équation suivante:

$$\frac{10mV}{1°C} = \frac{Vadc}{N°C} \qquad équation\ 11-3$$

Où N est la température en degré Celsius enregistrée par le capteur, de cet équation on peut déduire que la température N est :

$$N = \frac{Vadc}{10.10^{-3}} \qquad équation\ 11-4$$

Remplaçant l'équation(2) en (4), on obtient la relation suivante :

$$N = \frac{2{,}5.Radc}{10{,}23} = 0{,}244.Radc \qquad équation\ 11-5$$

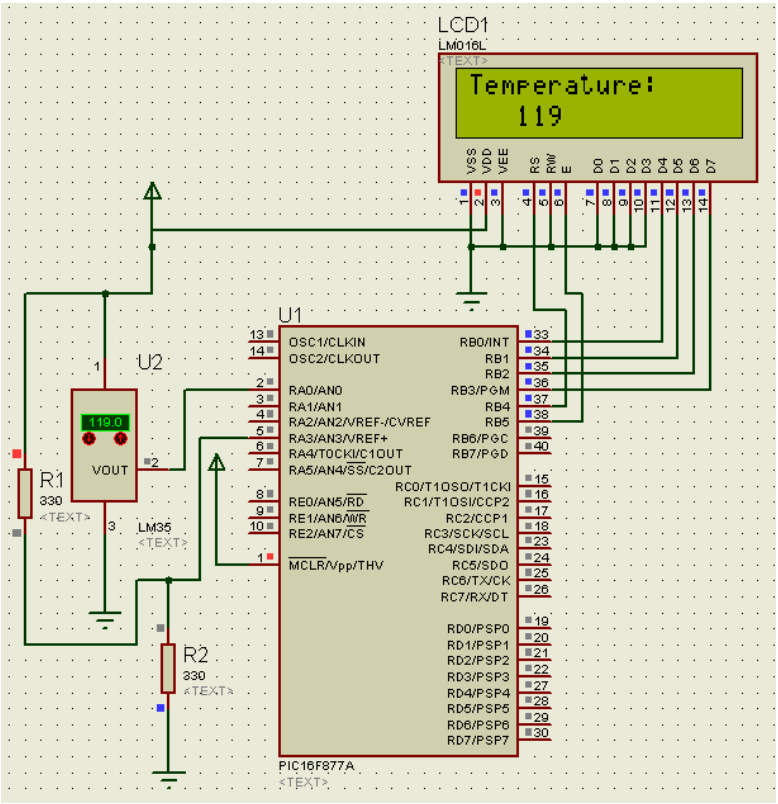
Cette relation doit être mise en œuvre avec la conversion AD, dans le programmation du PIC.

Pour mettre en contexte l'utilisation de ce capteur, on peut observer et analyser le code source suivant pour PIC 16F877A:

```
//Définition des broches du LCD
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D4 at RB0_bit;
//Définition des TRIS du LCD
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D7_Direction at TRISB3_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D4_Direction at TRISB0_bit;
void main( void )
{
//Déclaration des variables.
unsigned int Radc, TemI;
float Tem;
char Text[16];
//Configuration du module ADC avec la broche AN3
//comme tension de référence positive.
ADCON1 = 0b11000001;
//Initialization du LCD.
Lcd_Init();
//Effacer le curseur.
Lcd_Cmd(_LCD_CURSOR_OFF);
//Affichage du texte.
Lcd_Out( 1, 1, "Temperature:");

while(1)                    //Boucle infinie.
{
//Lecture du canal 0 du module CAN.
Radc = ADC_Read(0);
//Utilisation de l'équation (5).
Tem = 0.244*Radc;
//Convertir le résultat en un nombre entier.
TemI = Tem;
//Convertir le nombre entier en une chaine de caractères.
IntToStr( TemI, Text );
//Affichage du résultat.
Lcd_Out( 2, 1, Text);
//Retard de 100m secondes.
```

```
delay_ms(100);
}
}
```

Après l'édition et la compilation du programme, on doit construire un circuit dans ISIS avec les dispositifs suivants: 16F877A, IC, LM35, et LM016L, ceci peut être vu dans la figure suivante:



Circuit 11-1

Les résistances 330Ω constituent un diviseur de tension pour créer la référence de 2,5 volts. Au cours de la simulation, on peut changer la valeur de température au capteur LM35 pour le voir travailler....

11.2 Capteur de pression :

La pression est une quantité physique, représentée par un vecteur qui est défini comme étant la force appliquée sur une surface déterminée. La pression est notée en des unités différentes selon le système, la pression peut être donnée: psi, Pascal, atmosphères, etc. Pour des raisons pratiques, on va travailler avec le capteur de pression MPX4115, ce capteur est caractérisé en Kilo Pascal, et peut mesurer des pressions comprises entre 15 et 115 kPa, ou entre 2,18 et 16,7 psi. L'apparence physique, le brochage et commet il se présente sur ISIS de ce dispositif est dans la figure suivante:

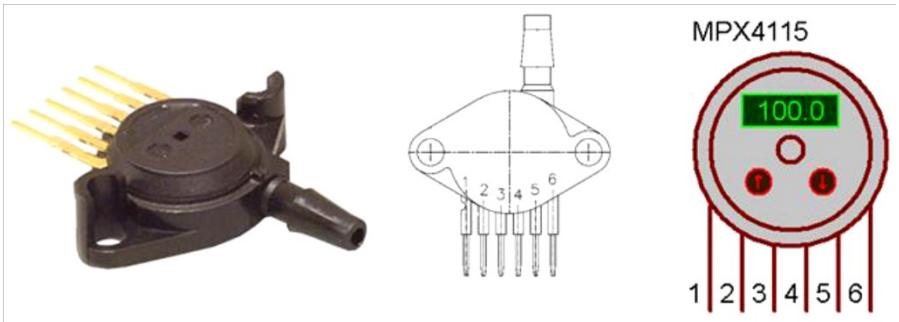


Figure 11.2

PIN	Fonction
1	Vout
2	GND, référence
3	Vs, alimentation
4, 5,6	No connectés

Tableau 11-1

En fonction du capteur mis en place, chaque capteur a une relation de la sortie ou une fonction de transfert, dans le cas particulier de la fonction de transfert du capteur MPX4115 indiquée par le fabricant est:

$$Vout = Vs(0{,}009.P - 0{,}095) \qquad équation\ 11-6$$

Où Vout est la tension de sortie du capteur, P est la pression en kilopascals, et Vs la tension d'alimentation du capteur. Résolution de l'équation (6), la pression P, on obtient:

$$P = \frac{111{,}11}{Vs}.Vout + 10{,}555 \qquad équation\ 11-7$$

En fonction du convertisseur AD du PIC configuré avec 10 bits de résolution on peut conclure:

$$\frac{5}{1023} = \frac{N}{Radc} \qquad équation\ 11-8$$

Donc on va extraire la tension n de l'équation (8) et en la remplaçant dans l'équation (7), on peut obtenir la relation ou l'équation qui va être utilisée dans le microcontrôleur pour déterminer la pression. La relation est la suivante:

$$P = \frac{0{,}54306.Radc}{Vs} + 10{,}555 \qquad équation\ 11-9$$

Pour le cas particulier de cet exemple, on utilise une source d'alimentation de 5V pour le PIC et le capteur. Donc Vs est égale à 5V, la relation finale devient :

$$P = 0{,}1086.Radc + 10{,}555 \qquad équation\ 11-10$$

Pour démontrer le fonctionnement pratique de ces capteurs, on peut observer et analyser le code source suivant:

```
//Définition des broches du LCD
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D4 at RB0_bit;
//Définition des TRIS du LCD
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D7_Direction at TRISB3_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D4_Direction at TRISB0_bit;

void main( void )
{
//Déclaration des variables.
```

```
unsigned int Radc, PreI;
float Pre;
char Text[16];
//Initialization du LCD.
Lcd_Init();
//Effacer le cursor.
Lcd_Cmd(_LCD_CURSOR_OFF);
//Affichage du texte.
Lcd_Out( 1, 1, "Pression en KPa:");

while(1) //Boucle infinie.
{
//Lecture du canal 0 du module CAN.
Radc = ADC_Read(0);
//Utilisation de l'équation (9).
Pre = 0.10861*Radc+10,5555;
//On prend la partie entière du résultat.
PreI = Pre;
//On convertit le nombre entier en chaine de caractères.
IntToStr( PreI, Text );
//Affichage du résultat.
Lcd_Out( 2, 1, Text);
//Retard de 100m secondes.
delay_ms(100);
}
}
```

TP 1 Prise en main

I. Création d'un premier projet

Vous disposez d'un petit fascicule en anglais intitulé ***Creating the first project in mikroC for PIC***. Ce texte vous indique la marche à suivre pour créer et compiler un projet.

1. Manipulation :

Suivre les instructions (jusqu'à l'exécution - run) avec les recommandations suivantes :

- nom du projet : tp1a
- Chemin du projet d:\tp_pic\tp1. Ce dossier doit être créé en remplaçant rep_perso par un nom qui vous est propre
- Le type (device) est à lire sur la puce elle-même. En principe 16F887.
- Choisir les fusibles par défaut.
- Le programme suivant est à saisir :

```
1   void main ()
2   {
3           PORTB = 0;
4           TRISB= 0;
5           while (1) {
6           PORTB = ~PORTB; // Clignoter PORTC
7           delay_ms(1000);
8           }
9   }
```

Si vous voulez un fonctionnement correct de Delay_ms, il faut configurer aussi la fréquence du quartz (8 MHz) correctement.

2. Manipulation

Aller dans le dossier de votre projet et examiner les fichiers .hex, .mcl, .asm, .lst. Que contiennent ces fichiers, quelle est leur utilité ?

Par quelles instructions assembleur sont traduites les lignes suivantes ?

- PORTB = 0;
- TRISB= 0;

II. Exécution pas-à-pas, déboguage

Créer un nouveau projet toujours dans votre dossier tp1 nommé tp1b. Le programme à saisir est le suivant :

```
1   void main ()
2   {
3           int k ;
4           PORTB = 0;
5           TRISB= 0;
6           for (k=0;k<256;k++) {
7           PORTB = k;      // Comptage binaire sur PORTB
8           delay_ms(500);}
9   }
```

3. Manipulation

Pour utiliser le débogueur reportez-vous à la photocopie ci-après.

Name	Description	Function key
Debug	Starts Debugger.	[F9]
Run/Pause debugger	Runs or pauses program execution and debugging.	[F6]
Toggle Breakpoints	During debugging, program is executed until a breakpoint is reached. At that point, the *Toggle Breakpoints* option sets new breakpoints or removes those already set at the current cursor position. To view all the breakpoints, select *Run >View Breakpoints* from the drop-down menu. Double click on an item from the list locates the breakpoint.	[F5]
Run to cursor	Program is executed until the cursor position is reached.	[F4]
Step Into	Executes the current C/Pascal/Basic (single- or multi-cycle) program line, then halts. If the program line is a routine call, steps into the routine and halts at the first instruction within it.	[F7]
Step Over	Executes the current C/Pascal/Basic (single- or multi-cycle) program line, then halts. If the program line is a routine call, enters the routine and halts at the first instruction following the call.	[F8]
Flush RAM	Flushes current PIC microcontroller RAM. All RAM memory values will be changed according to values in the *Watch* window.	-
Stop Debugger	Stops Debugger.	[Ctrl+F2]
Step Out	Executes all remaining program lines within the subroutine. It halts immediately upon exit from the subroutine.	[Ctrl+F8]
Disassembly View	Instead of program written in high-level program language (*Basic* in this very case), the assembly version of the same program will appear. Each program line written in Basic is divided in assembly instructions executed by the microcontroller.	[Alt+D]

- Compiler le projet (Build - CTRL+F9)
- Lancer le débogueur (Start Debugger - F9)
- Suivre en pas à pas l'exécution du programme (Step Into par exemple). Vérifier l'allumage correct des diodes du PORTB.

Prenez l'habitude à partir de maintenant de réaliser vos programme en version runtime, c'est-à-dire sans débogueur. Celui-ci ne sera utilisé que si nécessaire, c'est à dire pour retrouver une erreur subtile dans un programme.

TP2 des leds

I. Rappels

On rappelle qu'en C le OU booléen se fait par ||, le ET booléen par &&. Nous aurons besoin du OU bit à bit | et du ET bit à bit &. Soit le contenu d'un registre C sur 8 bits,

b7	b6	b5	b4	b3	b2	b1	b0
1	1	1	0	0	0	1	1

1. Préparation :

- Vous désirez mettre le bit b2 à 1 sans changer les autres bits, comment faites-vous ?
- Vous désirez mettre le bit b6 à 0 sans changer les autres bits, comment faites-vous ?

II. Exemple

On vous donne un programme C qui fait clignoter une led (poids faible) sur le portC.

```
1   void main ()
2    {
3
4               PORTC = 0;
5               TRISC= 0;
6               while (1) {
7               PORTC = 0x01;
8               Delay_ms(1000);
9               PORTC = 0x00;
10              Delay_ms(1000);
11               }
12  }
```

1. Exercice 2.1:

Écrire ce programme, le charger et l'exécuter. Modifiez-le pour faire clignoter RC1.

III. Exercices

1. Exercice 2.2:

Écrire un chenillard simple : une led se déplaçant sur le PORTC (de haut en bas) et en utilisant le même type de temporisation que dans le programme exemple. Utilisez l'un des opérateurs >> ou <<.

2. Exercice 2.3:

Écrire un chenillard double : un chenillard de haut en bas et simultanément de bas en haut qui se croisent.

TP3 Ports en entrée/sortie

I. Rappels

Les ports A, B, C, D et E sont des ports d'entrée/sortie dont chaque bit peut être utilisé soit en entrée soit en sortie, de façon indépendante. Ainsi chaque bit possède un satellite : TRISA, TRISB, TRISC, TRISD et TRISE qui permet de déterminer le sens de chaque bit (0: Output, 1 : input). Par exemple :

```
1  void main ()
2   {
3          TRISA= 0b00000100; // tous les bits en sortie pour PORTA sauf RA2
4          PORTA = 0;
5   }
```

est un programme qui positionne des entrées et sorties sur le PORTA.

II. Exercices

1. Exercice 3.1:

Écrire un programme qui positionne en entrée les quatre bits de poids faible du PORTB et en sortie les quatre autres. Le programme doit alors en permanence copier les 4 bits de poids faibles vers les quatre bits de poids fort.

2. Exercice 3.2:

Écrire un programme qui comporte les éléments suivants :

- un compteur binaire sur le PORTC (256 états). Placer une temporisation de 100 ms entre chaque état.
- L'appui sur RB2 (bit b2 du PORTB) doit remettre à zéro le compteur.
- Les accès aux bits seront effectués par des accès directs (en écrivant PORTB.F2 pour accéder au bit b2 du PORTB). Puis dans un second temps avec des masques.

3. Exercice 3.3: Changement d'état d'une diode

Réaliser un programme avec le cahier des charges suivant :

- En début de programme le bit RC0 (bit b0 du PORTC) doit être allumé.
- Ensuite un front montant sur RB0 (bit b0 du PORTB) provoquera un changement d'état de la diode.
- Les accès aux bits seront effectués par des accès directs (en écrivant PORTB.F2 pour accéder au bit b2 du PORTB). Puis dans un second temps avec des masques.

4. Exercice 3.4: Compteur

Reprendre le compteur de l'exercice 3.2 avec les modifications suivantes :

- un front descendant sur RA2 (bit b2 du PORTA) provoquera le RAZ du compteur
- un front descendant sur RA1 (bit b1 du PORTA) incrémentera le compteur.

TP4 afficheurs LCD

I. Présentation

Ce composant est spécialement fabriqué pour être utilisé avec les microcontrôleurs. Il est utilisé pour afficher des différents messages sur un écran à cristaux liquides miniature. Il peut afficher des messages sur deux lignes de 16 caractères chacune. Il permet d'afficher toutes les lettres de l'alphabet, les lettres grecques, des signes de ponctuation, des symboles mathématiques, etc. Il est également possible d'afficher des symboles construits par l'utilisateur.

Autres fonctions utiles comprennent déplacement automatique des messages (gauche et droite), l'apparence du curseur.

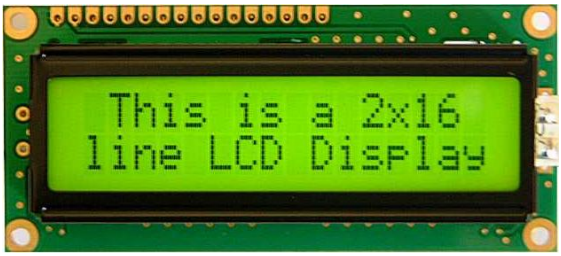


Figure 1

II. Exercices

1. Exercice 4.1:

En vous aidant de l'aide du mikroC PRO for PIC (figure 2) Écrire un programme qui affiche le message « Salam! » Sur le LCD 2x16, conformément à la (figure 3).

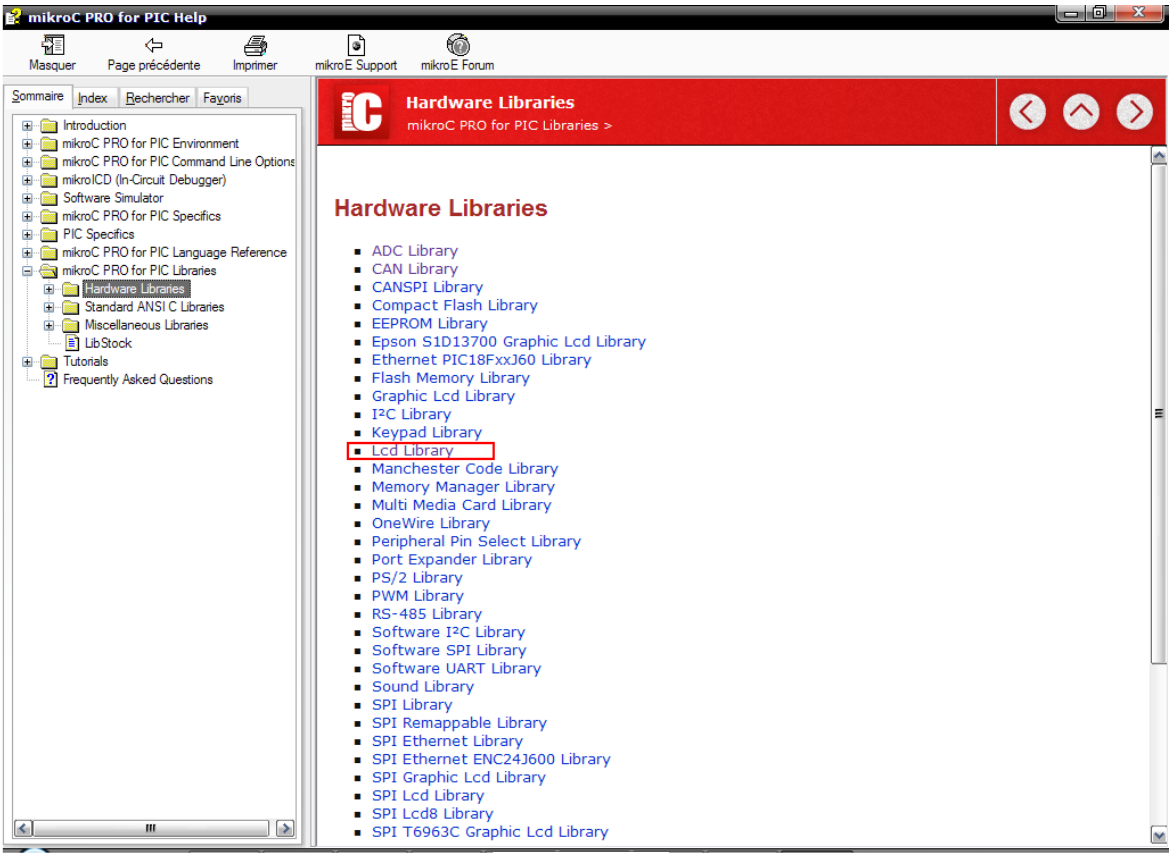


Figure 2

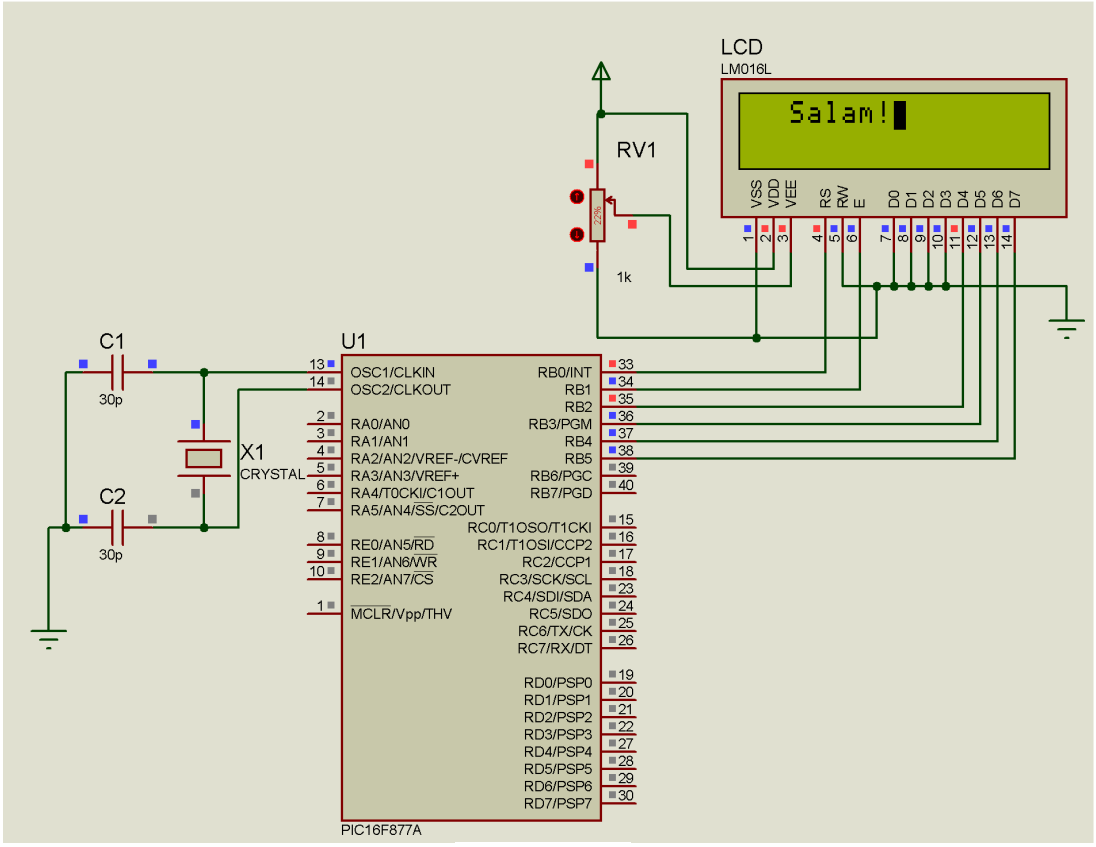
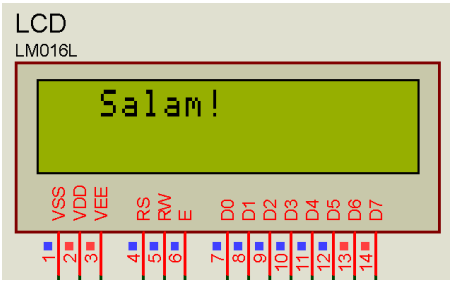


Figure 3

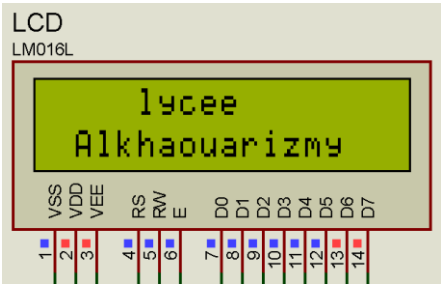
2. Exercice 4.2:

Écrire un programme qui comporte les éléments suivants :

- Effacer le curseur.



- Modifier le programme pour écrire le message « Lycée Alkhaouarizmy » sur deux lignes de l'afficheur LCD.



3. Exercice 4.3: Décalage à droite et à gauche du texte :

Réaliser un programme avec le cahier des charges suivant :

- Modifier le programme de l'exercice 4.2, pour décaler le message 16 fois à droite et 16 fois gauche.
- Effacer le l'afficheur LCD.

TP5 Convertisseur analogique numérique du PIC16F877

I. *Présentation* du C.A.N. (Convertisseur Analogique Numérique).

Le PIC 16F877 possède 8 entrées analogiques (RA0..RA5 et RE0..RE2 pour les PICs disposant du port E) multiplexées vers un C.A.N. à approximation successive (SAR = Successive approximation register). Ce dernier est précédé d'un échantillonneur / bloqueur permettant une stabilité de la tension d'entrée pendant toute la durée de la conversion.

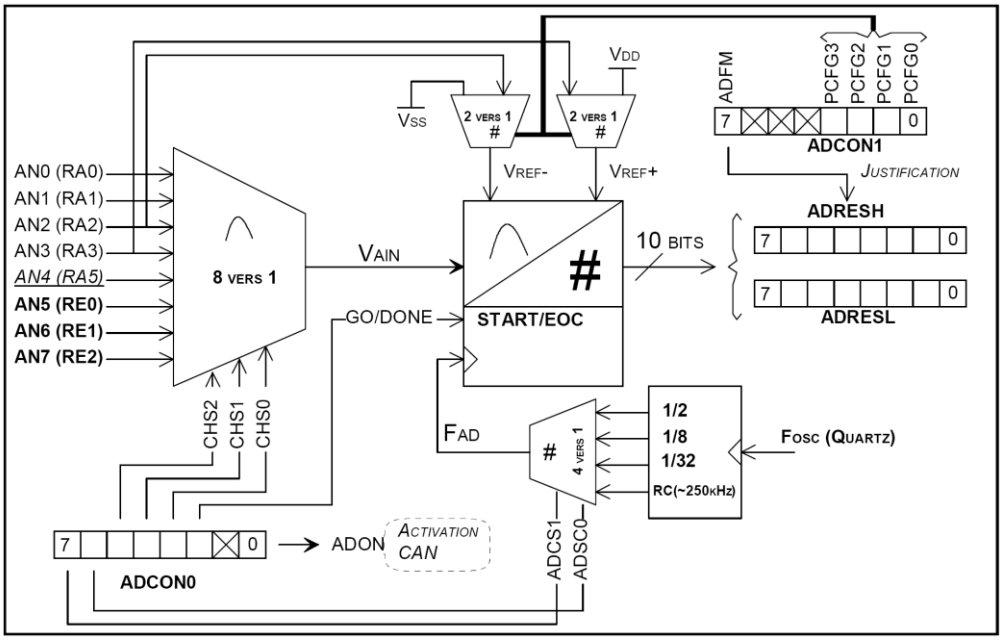


Figure 1

II. *Exemple*

On désire afficher la valeur en provenance d'un convertisseur analogique numérique sur un LCD.
La figure ci-dessous explique comment et un programme d'exemple se trouve parmi les démonstrations fournies dans la rubrique d'aide.

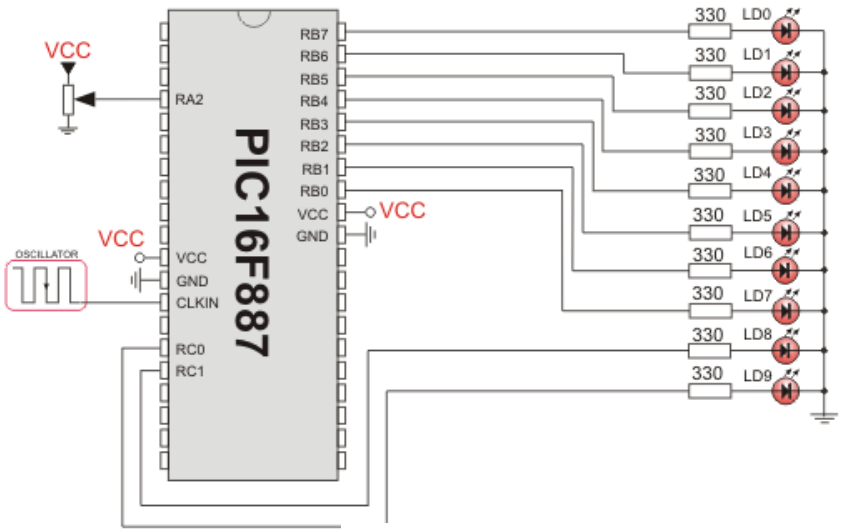


Figure 2

```
1   unsigned int temp_res;
2
3   void main() {
4
5   TRISA = 0xFF;                 // PORTA is input
6   TRISB = 0;                    // PORTB is output
7   TRISC = 0;                    // PORTC is output
8
9   do {
10  temp_res = ADC_Read(2);       // Get 10-bit results of AD conversion
11  PORTB = temp_res;             // Send lower 8 bits to PORTB
12  PORTC = temp_res >> 8;        // Send 2 most significant bits to RC1, RC0
13  } while(1);
14  }
```

1. Exercice 5.1:

Écrire un programme qui lit le convertisseur et affiche le résultat sur l'afficheur.

Adapter pour que votre programme lise RA3 (contre RA2 dans le programme d'exemple).

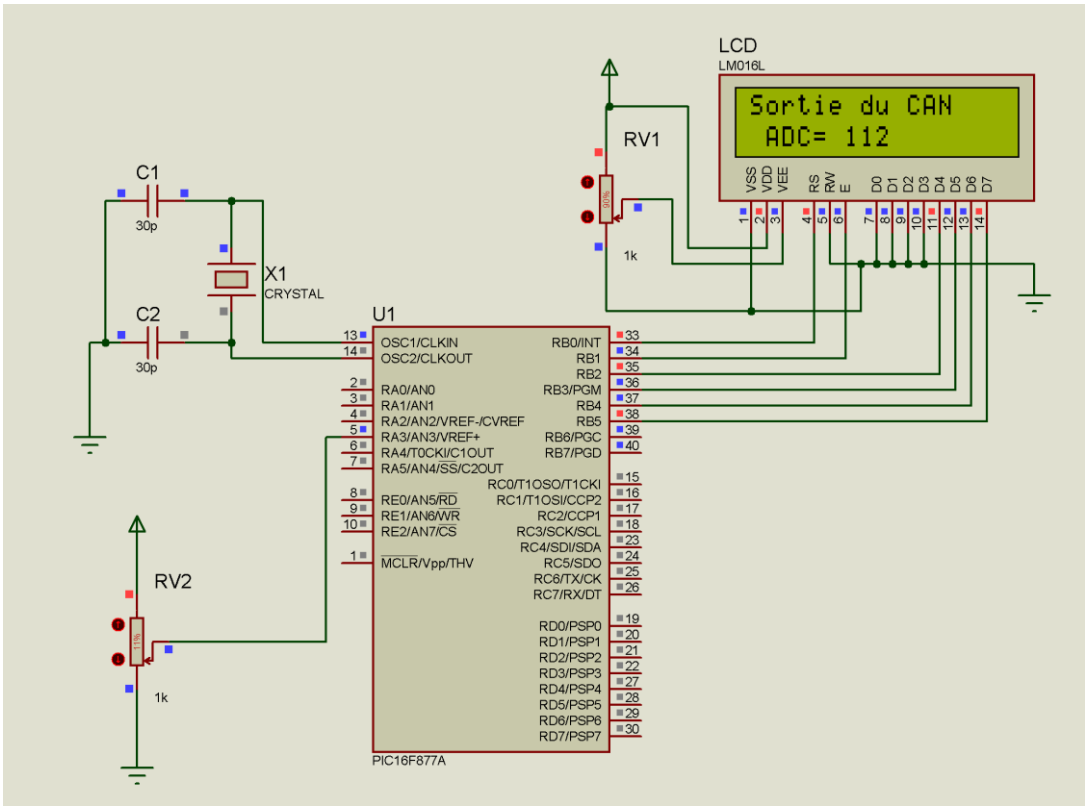


Figure 3

2. Exercice 5.2:

Modifier le programme précédent pour qu'il affiche la valeur de la tension correspondante à la valeur numérique donnée par le convertisseur conformément à la figure ci-dessous :

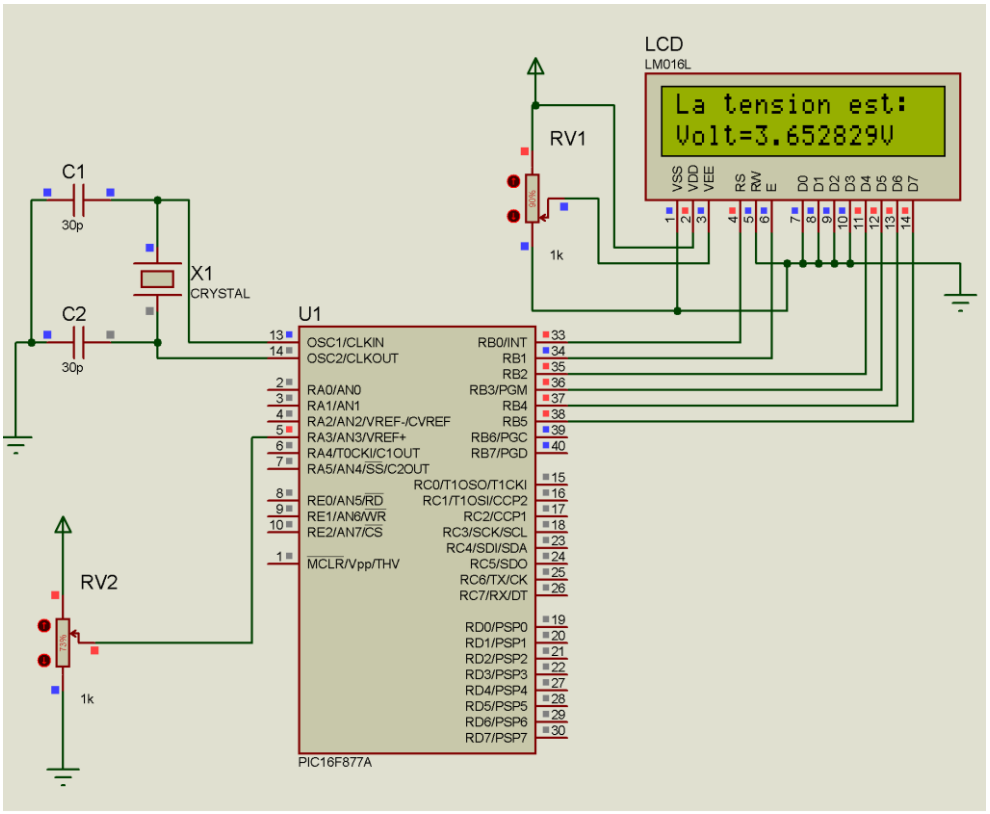
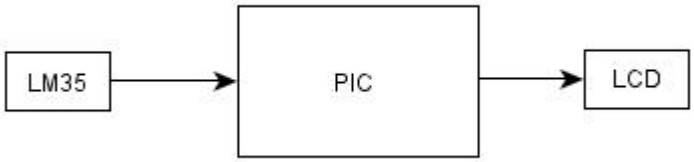

LCD
LM016L
La tension est:
Volt=3.652829V
VSS VDD VEE RS RW E D0 D1 D2 D3 D4 D5 D6 D7
RV1
1k
C1
30p
X1
CRYSTAL
C2
30p
U1
OSC1/CLKIN
OSC2/CLKOUT
RA0/AN0
RA1/AN1
RA2/AN2/VREF-/CVREF
RA3/AN3/VREF+
RA4/T0CKI/C1OUT
RA5/AN4/SS/C2OUT
RE0/AN5/RD
RE1/AN6/WR
RE2/AN7/CS
MCLR/Vpp/THV
RB0/INT
RB1
RB2
RB3/PGM
RB4
RB5
RB6/PGC
RB7/PGD
RC0/T1OSO/T1CKI
RC1/T1OSI/CCP2
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RD0/PSP0
RD1/PSP1
RD2/PSP2
RD3/PSP3
RD4/PSP4
RD5/PSP5
RD6/PSP6
RD7/PSP7
PIC16F877A
RV2
1k

Figure 4

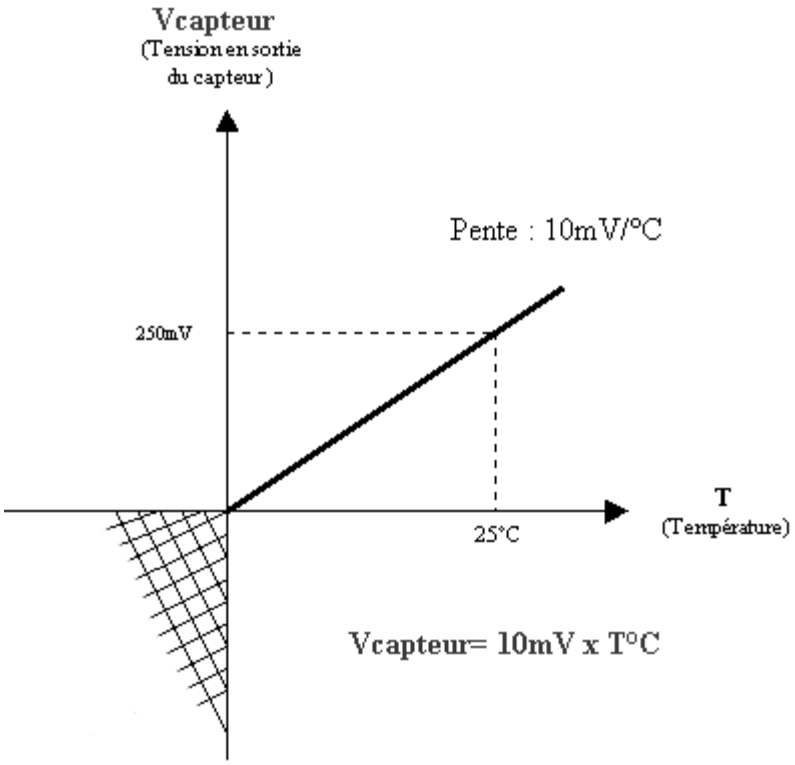
Exercice 1 :

Écrire le programme pour réaliser la mesure et l'affichage sur LCD de la température en utilisant un capteur de température LM35. La température mesurée est comprise entre 0°C et 150°C.

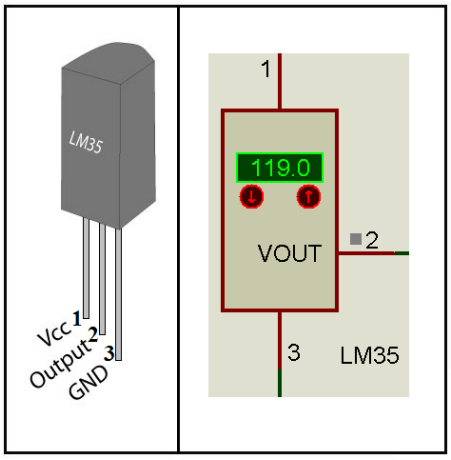


- **Caractéristiques du capteur de température LM 35**

  - **Le capteur LM35 dispose d'une caractéristique linéaire Vcapteur = f (T°) suivante :**



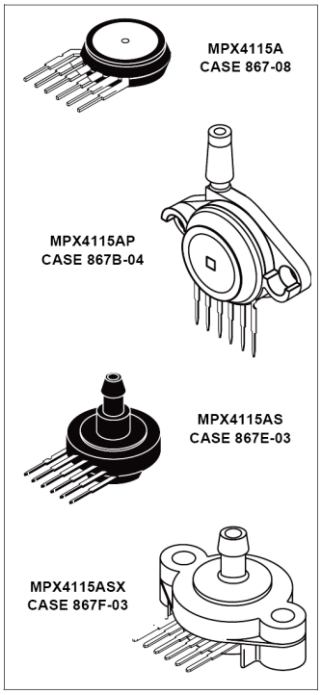
Caractéristiques principales	
Plage de la tension d'alimentation	**0,2 Volt à 35 Volts**
Sensibilité	**10 mV / °C**
Précision	**+/- 0,5°C (à 25°C)**
Type de boîtier	**TO 92**

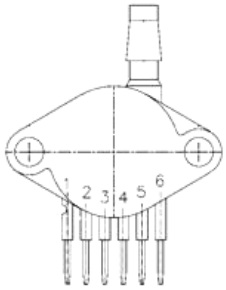


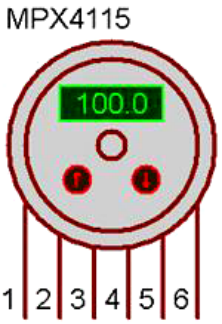
Exercice 2 :

Écrire le programme pour réaliser la mesure et l'affichage sur LCD de la pression en utilisant un capteur de pression MPX4115. La pression mesurée est comprise entre 15 et 115KPa.

- **Caractéristiques du capteur de pression MPX4115 :**







PIN	Fonction
1	Vout
2	GND, référence
3	Vs, alimentation
4, 5,6	No connectés

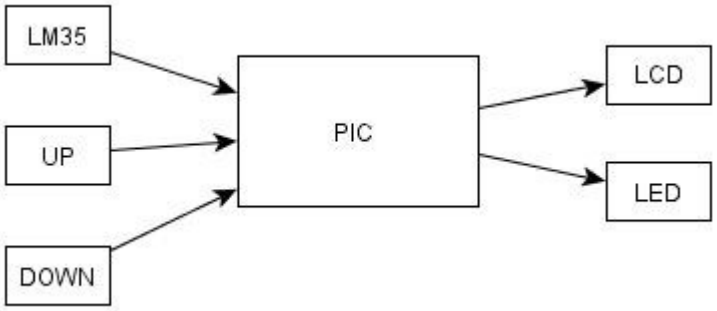
Transfer Function (MPX4115)

Nominal Transfer Value: $V_{out} = V_S (P \times 0.009 - 0.095)$

$V_S = 5.1\ V \pm 0.25\ Vdc$

Exercice 3:

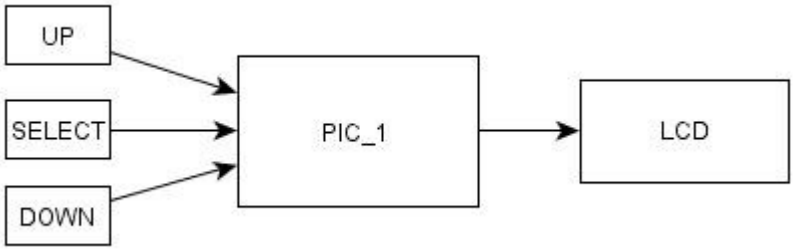
Écrire un programme de thermostat utilisant le LM35, un bouton UP, un bouton DOWN, un afficheur LCD et une LED:



- Appui sur UP: Tréglée ++
- Appui sur DOWN: Tréglée --
- Affichage: Tmesurée et Tréglée
- Si Tmesurée < Tréglée, LED allume
- Si Tmesurée ≥ Tréglée, LED éteint

Exercice 4:

Faire l'algorithme de gestion de l'affichage d'un menu sur LCD série:



Au début:

→	Choix 1
	Choix 2
	Choix 3
	Choix 4

Appui sur DOWN:

	Choix 1
→	Choix 2
	Choix 3
	Choix 4

Appui sur DOWN:

	Choix 1
	Choix 2
→	Choix 3
	Choix 4

Appui sur DOWN:

```
    Choix 1
    Choix 2
    Choix 3
→   Choix 4
```

Appui sur DOWN:

```
    Choix 2
    Choix 3
    Choix 4
→   Choix 5
```

Appui sur UP:

```
    Choix 2
    Choix 3
→   Choix 4
    Choix 5
```

Le code ASCII pour → est 0x7E

Exercice 5:

Écrire le programme pour faire défiler un texte de gauche à droite sur l'afficheur LCD:

```
                    L
```

```
                   Ly
```

```
                  lyc
```

```
                 lyce
```

...

```
Lycee  Alkhaouarizmy
```

```
ycee  Alkhaouarizmy
```