République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Frères Mentouri – Constantine 1
Faculté des Sciences de la Technologie
Département d'Electronique

DOCUMENT DE TRAVAIL

# Travaux Pratiques

MASTER 1, Automatique et informatique industrielle

## Systèmes Embarqués et Systèmes Temps Réel

Réalisé et présenté par :

**Dr. Salah ABADLI**

**Année Universitaire : 2019 / 2020** (Semestre 2)

# TP 1, INITIATION A LA PROGRAMMATION DU PIC 16F877A
# PREMIERE APPROCHE DES LOGICIELS UTILISES

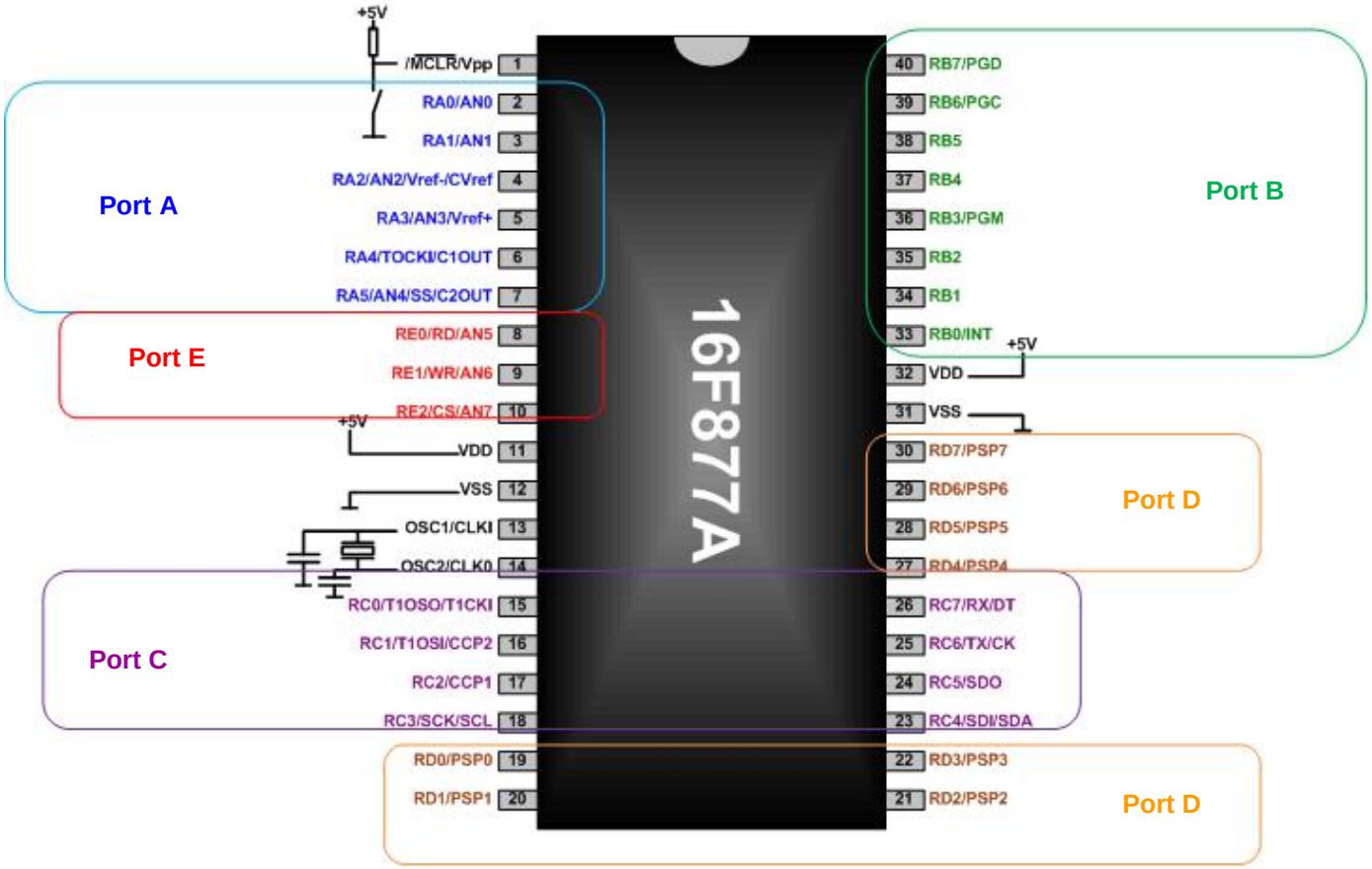
**Objectif** : L'objectif capital de ce premier TP est de se familiariser avec les outils employés pour réaliser les TPs de ce module. En bref, nous nous intéressons à la programmation du microcontrôleur PIC de Microship (PIC16F877A) et aux environnements de conception et de simulation (MikroC et Proteus ISIS).

Les PIC sont des microcontrôleurs RISC (de l'anglais, Reduced Instructions Set Computer) ; qui signifie : calculateur à jeu réduit d'instructions. Ils sont dédiés aux applications qui ne nécessitent pas une grande une grande quantité de calculs complexes, mais qui demandent beaucoup de manipulations d'entrées / sorties. En effet, il existe plusieurs familles de microcontrôleurs, dont les plus connues sont : Atmel, Motorola, Microship, Intel,…etc. La famille des PIC à bus de données 8 bits est subdivisée, à l'heure actuelle, en 3 grandes catégories :

- Base-ligne : utilisent des mots d'instruction de 12 bits.
- Mid-Range : utilisent des mots d'instruction de 14 bits.
- High-End : utilisent des mots d'instruction de 16 bits.

## I- Brève découverte du PIC 16F877A

L'alimentation du circuit est assurée par les pattes **VDD** (4,5 à 6V) et **VSS** (GND). Elle permet le fonctionnement du PIC.
Il est possible d'utiliser un oscillateur avec un quartz allant jusqu'à 20 Mhz relié avec 2 condensateurs de découplage (**OSC1**/CLKI et **OSC2**/CLKO), du fait de la fréquence importante du quartz utilisé. Par conséquent, quelque soit l'oscillateur utilisé, l'horloge système base est obtenue en divisant la fréquence par 4.
La broche **MCLR** (Master Clear) a pour effet d'exciter la réinitialisation du PIC (RESET) lorsqu'elle est connectée à 0. Lorsque le RESET est activé, tous les registres sont initialisés.



Le PIC 16F877A, dispose de 5 ports :

- **Port A** : 6 pins I/O numérotées de RA0 à RA5. En effet, ces broches (sauf RA4) sont multiplexées avec les entrées du convertisseur analogique numérique (AN0…AN4). RA4 est multiplexée avec le timer0 (T0CKI).
- **Port B** : 8 pins I/O numérotées de RB0 à RB7. Possibilité de déclenchement d'interruptions.
- **Port C** : 8 pins I/O numérotées de RC0 à RC7. Partagé, avec liaison, les timers 1 et 2 et les modules CCP.
- **Port D** : 8 pins I/O numérotées de RD0 à RD7. Possibilité d'interfaçage avec un autre processeur et autres.
- **Port E** : 3 pins I/O numérotées de RE0 à RE2. Possibilité d'interfaçage avec un autre processeur et autres.

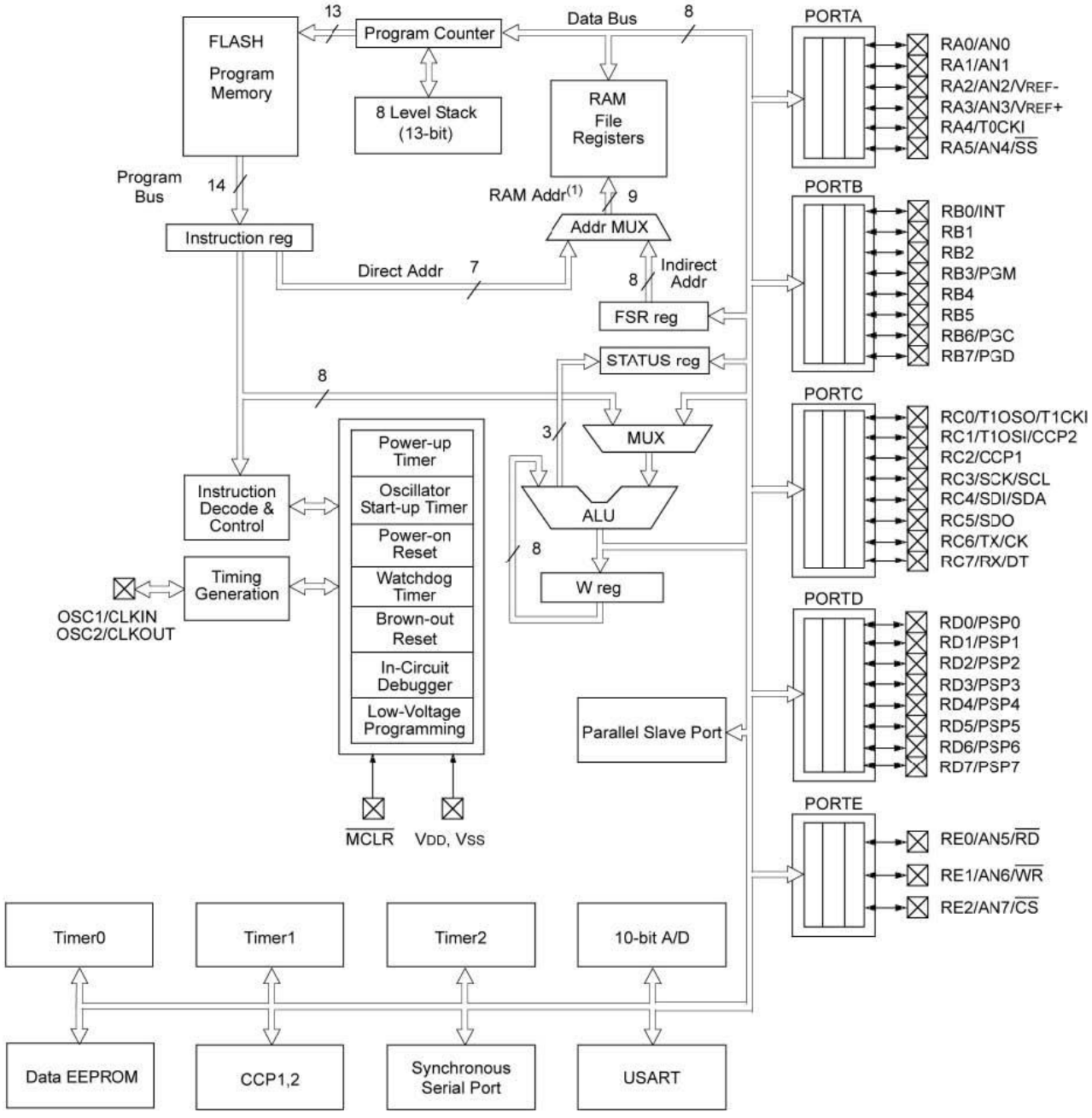
A chaque port correspondent 2 registres :

- Un registre direction pour programmer les lignes en entrée ou en sortie : TRISA, TRISB, TRISC, TRISD, TRISE.
- Un registre de données pour lire ou modifier l'état des broches : PORTA, PORTB, PORTC, PORTD, PORTE.

Pour déterminer les modes des ports (I/O), il faut sélectionner leurs registres TRISX :

- Le positionnement d'un bit à "1" programme la broche correspondante en entrée.
- Le positionnement d'un bit à "0" programme la broche correspondante en sortie.

**Remarque** : Toutes les lignes des PORTS peuvent fournir un courant de 25 mA par ligne de PORT. Une limite de 40 mA par PORT doit être respectée pour des questions de dissipation (échauffement).



Le PIC 16F877A, dispose de 3 timers :

- **Timer0** (8 bits) : il peut être incrémenté par les impulsions extérieures via la broche (TOCKI/ RA4) ou par l'horloge interne (Fosc/4).
- **Timer1** (16 bits) : il peut être incrémenté soit par l'horloge interne ; par des impulsions sur la broche TICKI/RCO ou par un oscillateur (RC ou quartz) connecté sur les broches TOSO/RCO et T1OSI/RCI.
- **Timer2** (8 bits) : il est incrémenté par l'horloge interne ; celle peut être pré-divisée.

**Remarque** : Le convertisseur Analogique / Digital du PIC 16F877A est à approximations successives et il possède une résolution de 10 bits (10-bit A/D). La liaison série USART est une interface asynchrone.

## II- MikroC PRO et Programmation du PIC 16F877A

La principale différence entre le microcontrôleur et le microprocesseur est que le microcontrôleur possède en interne le programme qu'il devra effectuer en fonction de l'application pour lequel il a été conçu.
Le langage machine est le langage compris par les processeurs. Ce langage est difficile à maitriser puisque chaque instruction est codée par une séquence propre de bits. Afin de faciliter la tâche au programmeur, on a créé différents langages plus ou moins évolués (langage de haut niveau).

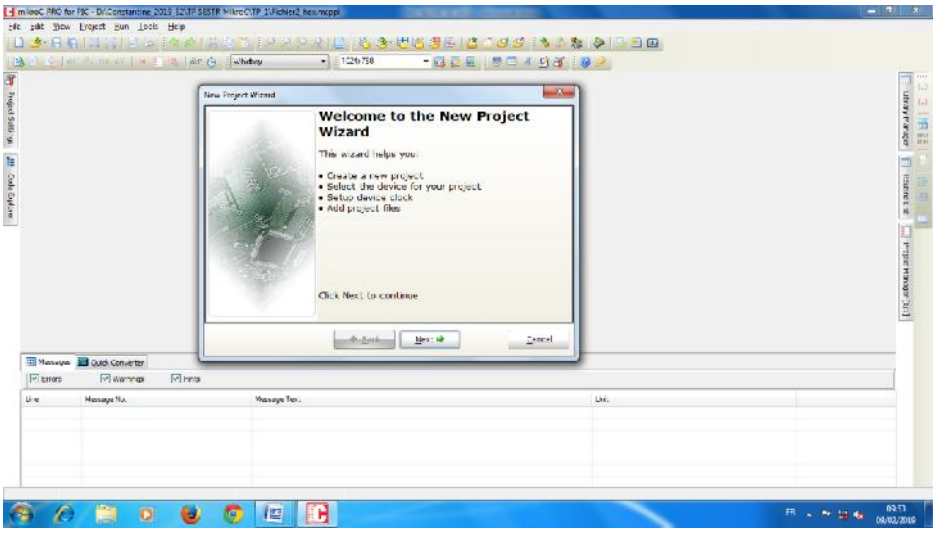
Pour programmer un microcontrôleur, il est possible d'utiliser différents types de langages de programmations de haut niveau ; tels que : C, C++, JAVA,…etc. Le programme réalisé est ensuite compilé dans le langage assembleur conçu par le constructeur du microcontrôleur. Le programme ainsi compilé sera injecté, par la suite, du PC dans la mémoire programmable du microcontrôleur.

En effet, les lignes des programmes que nous écrivons avec les différents types de langages constituent ce qu'on appel un programme source (xxxx.**m**, xxx.**c**,…etc). Le microcontrôleur ne comprend pas ces programmes. Pour que le microcontrôleur puisse comprendre nos programmes, il faut les compiler (Build) en langage machine, qui est une suite de codes machine (fichiers avec l'extension xxx.**hex**).

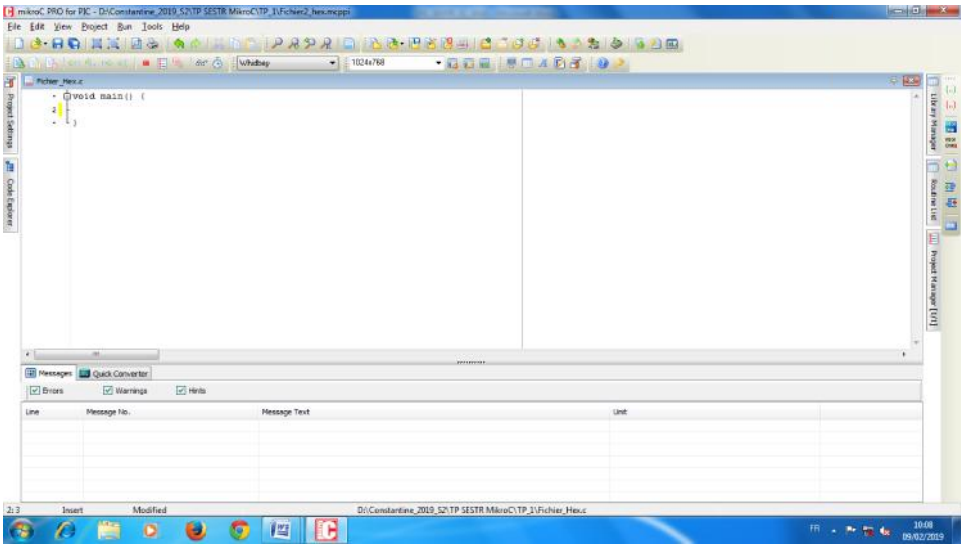
Le langage MikroC PRO (pour PIC) a trouvé une large application pour le développement de systèmes embarqués sur la base de microcontrôleur. Il assure une combinaison de l'environnement de programmation avancée IDE (Integrated Development Environment), et d'un vaste ensemble de bibliothèques pour le matériel, de la documentation complète et d'un grand nombre des exemples.

Le MikroC PRO est un compilateur C pour PIC bénéficier d'une prise en main très facile. Il a une capacité à pouvoir gérer la plupart des périphériques rencontrés dans l'industrie (Bus I2C, 1Wire, SPI, RS485, Bus CAN, cartes compact Flash, signaux PWM, afficheurs LCD et 7 segments...etc.), de ce fait il est un des outils de développement incontournable et puissant.

- Pour installer MikroC, télécharger depuis les serveurs et faire l'installation.
- L'environnement de démarrage du logiciel MikroC PRO est le suivant :



- Pour débuter un projet, il faut suivre les étapes suivantes :
  - ✓ Créer un nouveau dossier; dans le disque dure de votre PC.
  - ✓ Créer un nouveau projet (Project / New Projet Wizard / Next).
  - ✓ Sélectionner le type du PIC (P16F877A / Next).
  - ✓ Préciser le Clock (8.000000 MHz / Next).
  - ✓ Spécifier le chemin d'enregistrement (Ouvrir / chemin du dossier créé) puis donner un nom type mikroC Project (*.mcppi) et Enregistrer.
  - ✓ Une fois cette première étape terminée, saisissez votre programme, compilez, corriger les erreurs et exécuter.



## III- Proteus ISIS et vérification de l'application par simulation

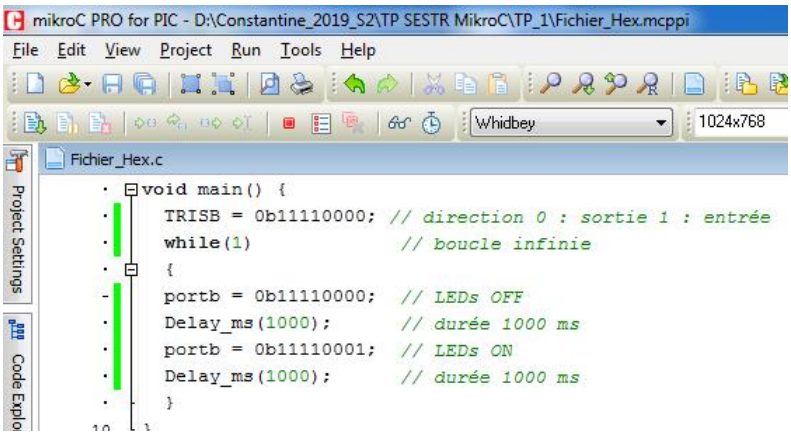
C'est un très bon logiciel de simulation en électronique. C'est un éditeur de schémas qui intègre un simulateur analogique, logique ou mixte. Proteus ISIS est capable de vérifier et simuler le comportement du PIC 16F877A.

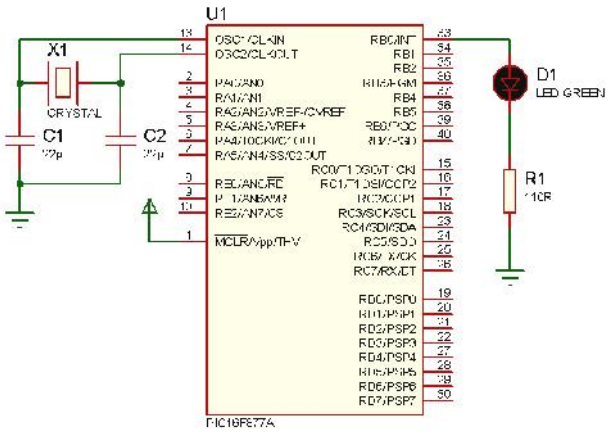
En effet, la simulation permet d'ajuster et de modifier les paramètres du circuit étudié comme si on manipulait un montage électronique réel. Ceci permettra d'accélérer le prototypage et de réduire le coût de réalisation.

## IV- Exemples pratiques de programmation du PIC 16F877A

**Exercice1** (Lancer Proteus ISIS et l'éditeur MikroC PRO)

- Réaliser le premier circuit de test qui permet de faire clignoter une LED (ON durant 1 sec et OFF durant 1 sec) ?




| Decimal | Binary | Octal | Hexadecimal |
|---|---|---|---|
| 0 | 0b00000000 | 00 | 0x00 |
| 1 | 0b00000001 | 01 | 0x01 |
| 128 | 0b10000000 | 0200 | 0x80 |
| 255 | 0b11111111 | 0377 | 0xFF |

**Remarque** : Il est possible d'accéder individuellement à 1 seul bit du PORT. A titre d'exemple, on écrit :

TRISB.F0 = 0 ;

PORTB.F0 = 1 ;

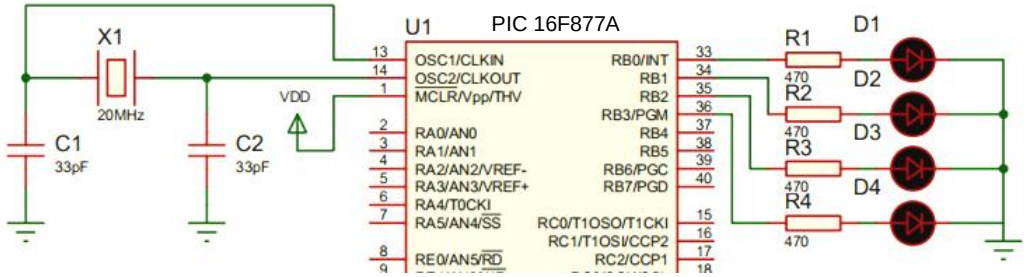
portb.F0 = 0 ;

- Visualiser le contenu du fichier enregistré avec l'extension **.hex** ?

```
:020000001628C0
:0E000600831203132108A00200882000800DC
:1000140003208A110A128000840AA00A0319A10A83
:08002400F003031D0A28080087
:10002C00831603130610831206140B30FB002630C4
:10003C00FC005D30FD00FD0B2128FC0B2128FB0B87
:10004C00212800000000006100B30FB002630FC00BD
:10005C005D30FD00FD0B3028FC0B3028FB0B3028ED
:08006C000000000019283928EA
:02400E004A2F37
:00000001FF
```

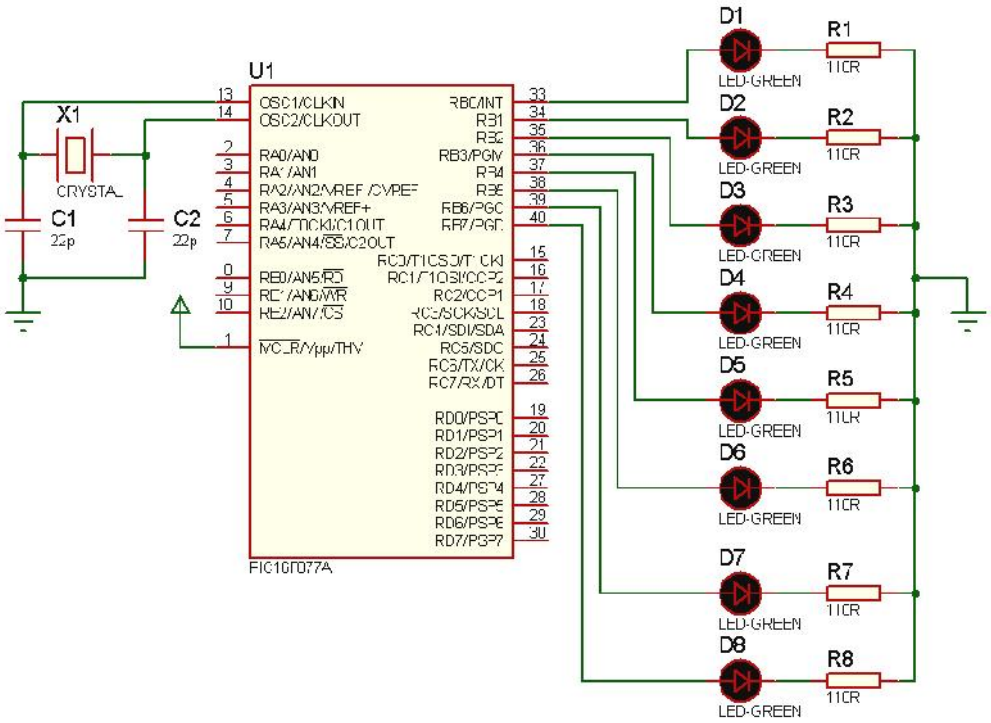
**Exercice 2**

- Dans le même projet, réaliser un 2ème circuit de test qui permet de faire clignoter 4 LEDs en même temps ?
- Vérifier le fonctionnement de votre programme ? Visualiser le fichier enregistré avec l'extension **.hex** ?




**Exercice 3**

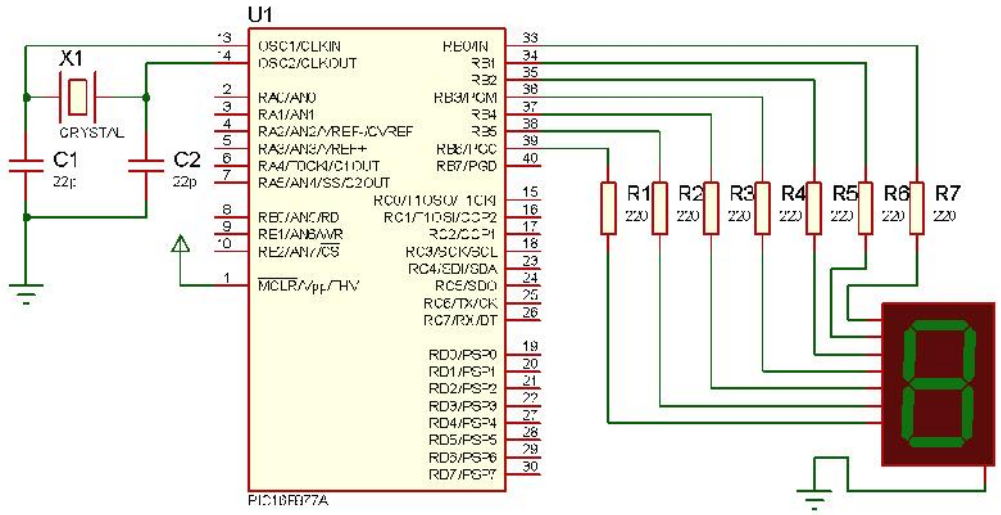
- Dans le même projet, réaliser un 3ème circuit de test qui permet de faire allumer 1 LED parmi 8 durant 0,5 sec?
- Vérifier le fonctionnement de votre programme ?




**Exercice 4**

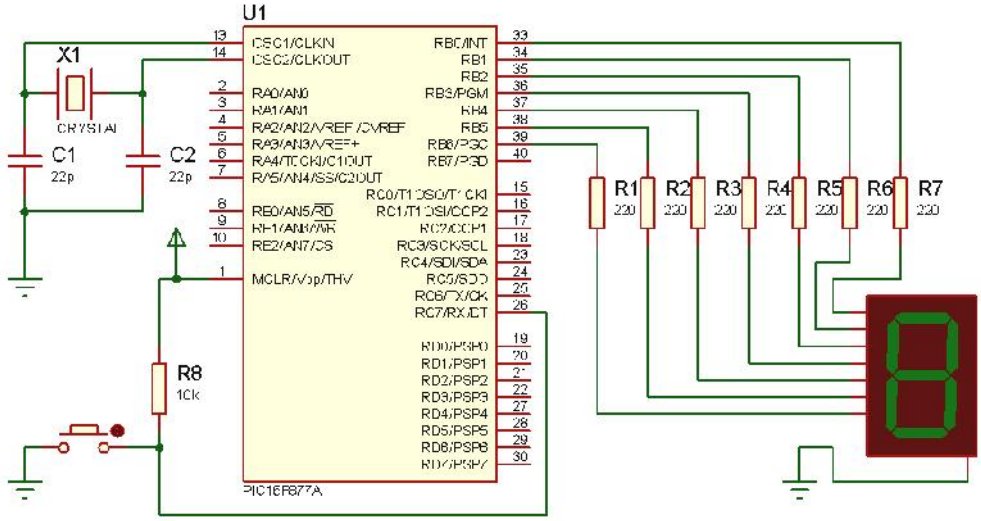
- Dans le même projet, réaliser un 4ème exercice de test qui permet d'afficher un chiffre 7 segments de 0 à 9 avec une cadence de 500ms ? (déclarer un tableau tab[10] contenant les 10 codes 7 segments des chiffres de 0 à 9 et avec une boucle for repérer le code à envoyer).
- Vérifier le fonctionnement de votre programme via PROTEUS ISIS ?

**Remarque** : MikroC vous permet de déterminer le code 7 segments ; en allant dans : Tools → Seven Segment Editor.



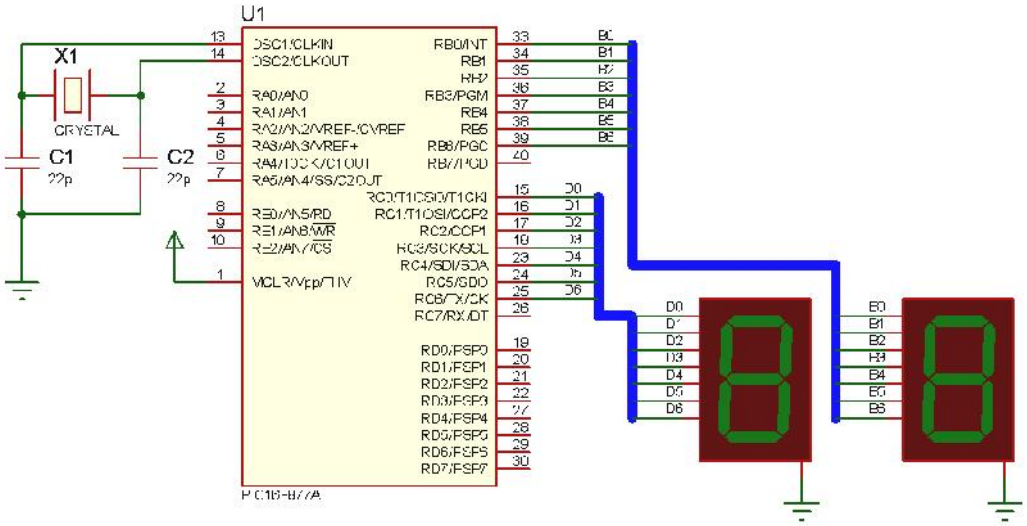

**Exercice 5**

- Dans le même projet, réaliser un 5ème exercice de test qui permet d'incrémenter le chiffre affiché en appuyant un bouton poussoir ?
- Vérifier le fonctionnement de votre programme via PROTEUS ISIS ?




**Exercice 6**

- Dans le même projet, réaliser un 6ème exercice de test qui permet de contrôler l'affichage de deux chiffre (2 digits) 7 segments de 0 à 99 avec une cadence de 500ms ? (déclarer un tableau tab[10] contenant les 10 codes 7 segments des chiffres de 0 à 9 et avec des boucles for repérer le code à envoyer) ?
- Vérifier le fonctionnement de votre programme via PROTEUS ISIS ?

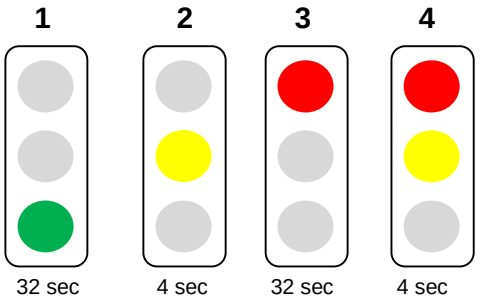


# TP 2, PROGRAMMATION DU PIC 16F877A
# EXEMPLES DE GESTION DES FEUX TRICOLORES

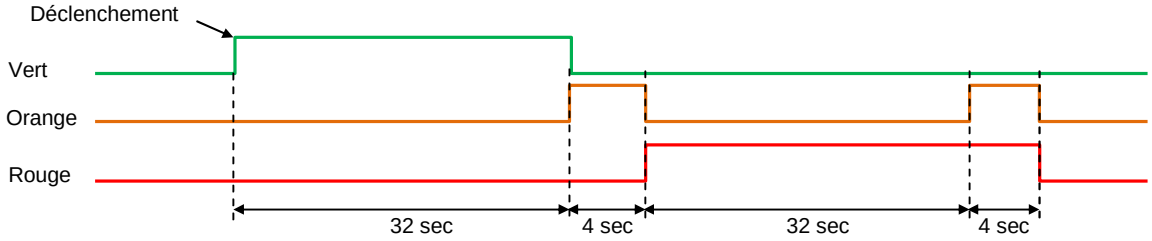
**Objectif** : L'objectif de ce 2ème TP est toujours de se familiariser avec la programmation du microcontrôleur PIC 16F877A et aux environnements de conception et de simulation (MikroC et Proteus ISIS). On cherche simplement à réaliser un programme en MikroC qui organise la circulation des véhicules dans un croisement externe de route (carrefour).

## I- Gestion d'un seul feu tricolores

Réaliser un programme en MikroC qui permet de piloter le feu tricolore de la figure ci-dessous. Vérifier le bon fonctionnement du programme réalisé en utilisant Proteus ISIS.

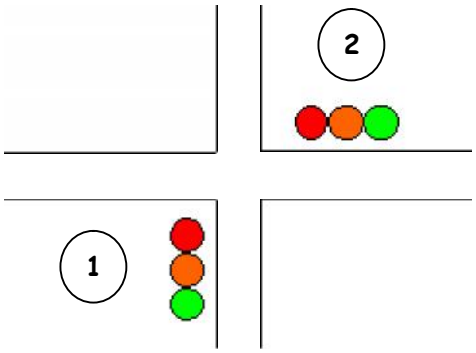



En effet, le chronogramme de fonctionnement est le suivant :

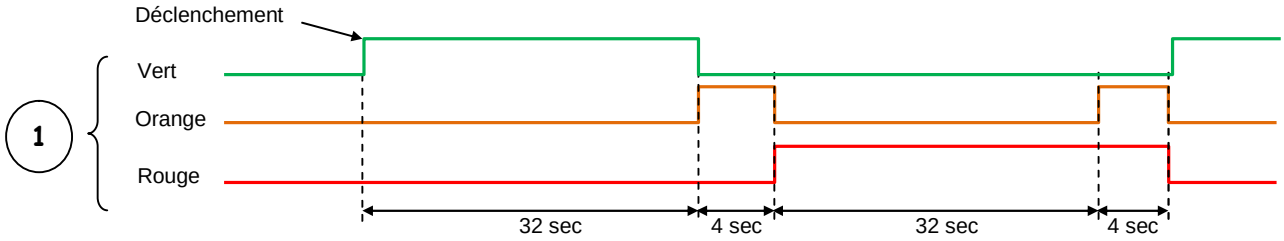


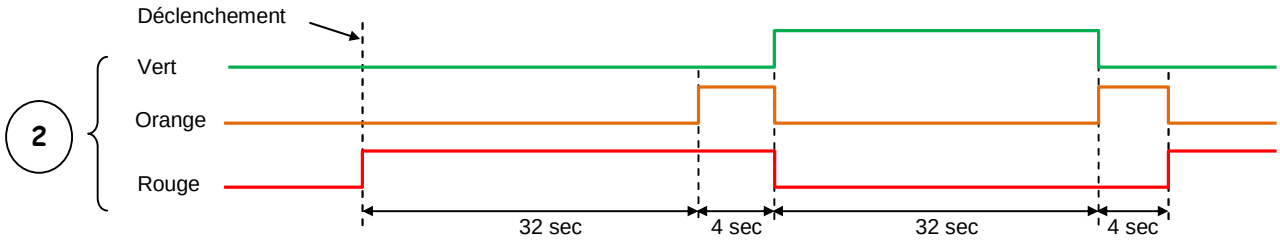

## II- Gestion de deux feux tricolores de circulation (carrefour)

Réaliser un programme en MikroC qui permet de piloter deux feux tricolores de carrefour. Vérifier le bon fonctionnement du programme réalisé en utilisant Proteus ISIS.



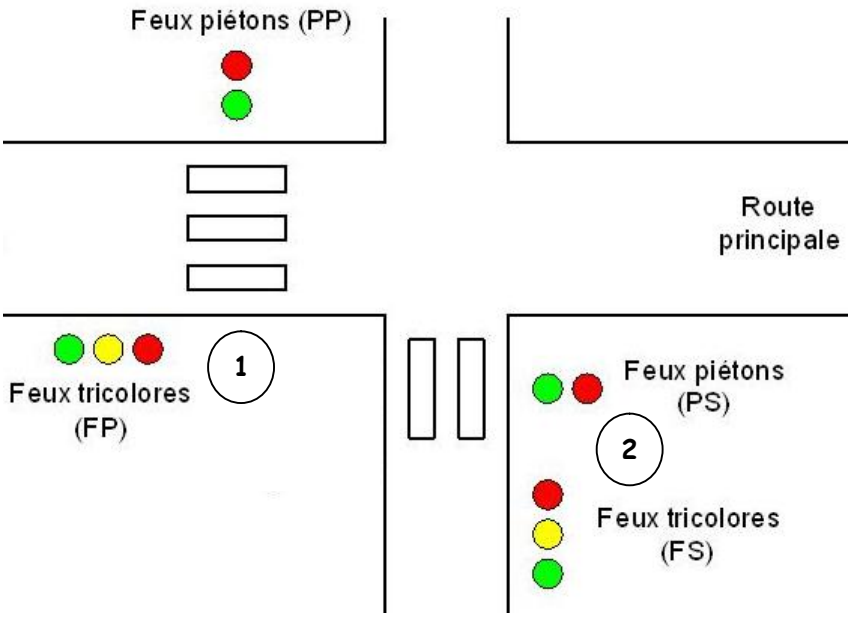

Le chronogramme de fonctionnement est le suivant :



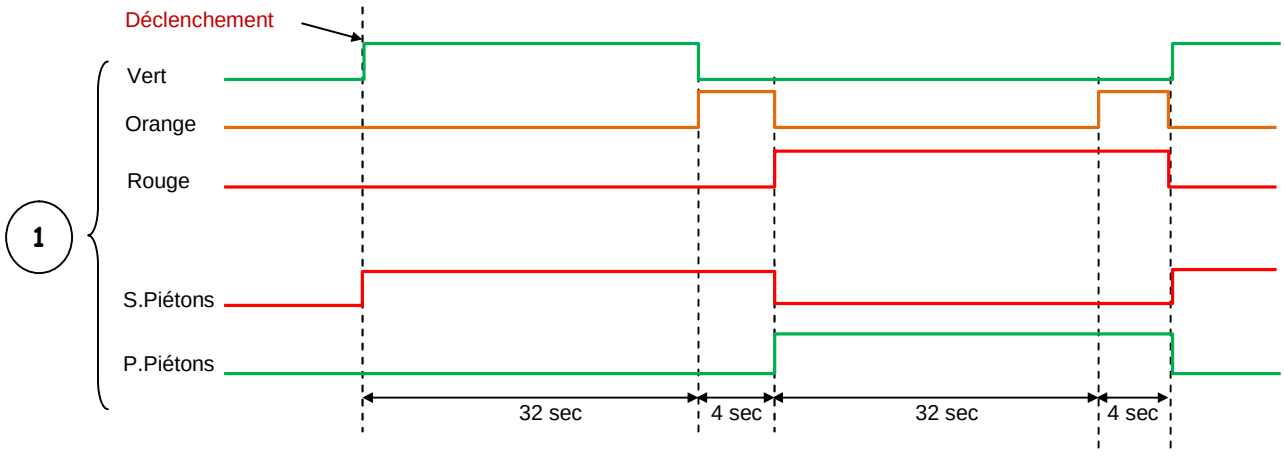


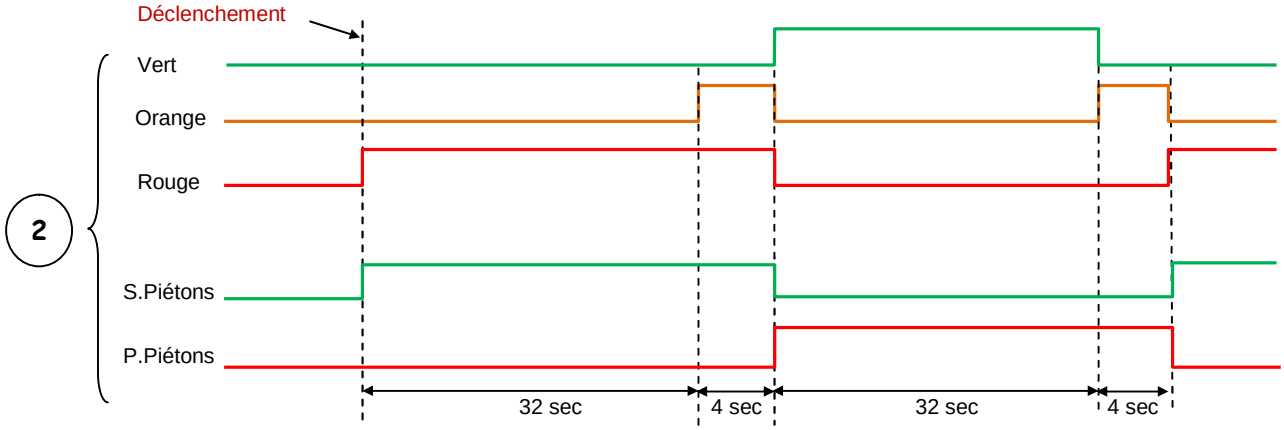

### III- Gestion de deux feux tricolores de circulation et deux passages piétons

Réaliser un programme en MikroC qui permet de piloter un carrefour avec passage piétons. Vérifier le bon fonctionnement du programme réalisé en utilisant Proteus ISIS.




Le chronogramme de fonctionnement est le suivant :



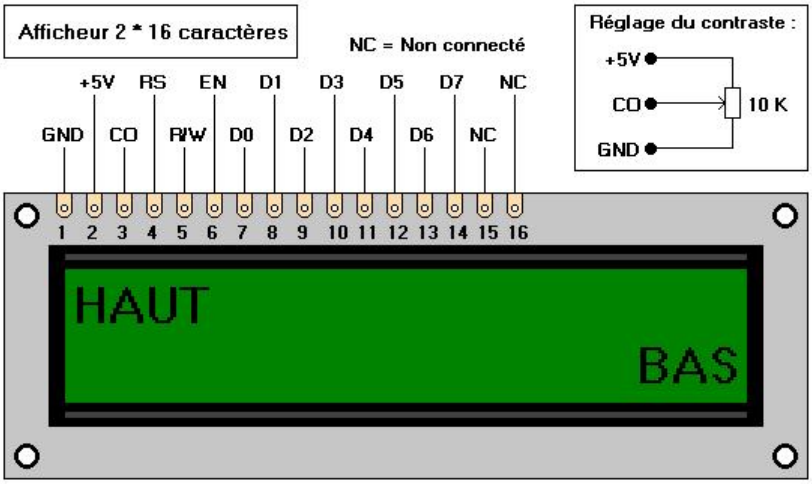
# TP 3, PROGRAMMATION DU PIC 16F877A

## GESTION DE L'AFFICHAGAGE LCD 2X16 ET DU CLAVIER MATRICIEL

**Objectif** : A l'aide des microcontrôleurs, il est possible de réaliser des montages avec une véritable gestion de l'information et des protocoles pour la communication de données (entrées/sorties). Il devient possible d'utiliser, pour l'interface utilisateur, des périphériques "évolués" comme des claviers matriciels ou encore des afficheurs à cristaux liquides (LCD). Nous nous intéressons dans ce 3ème TP à la gestion du clavier matriciel et de l'afficheur LCD 2x16.

### I- L'afficheur LCD 2x16

La figure ci-dessous montre le brochage d'un afficheur LCD 2x16 :




Le tableau suivant donne une description rapide de la fonction de chaque broche :

En observant le brochage de l'afficheur, on constate qu'il faut un minimum de 6 sorties pour le commander. En effet, si on utilise l'adressage sur 4 bits et que l'on se prive de la lecture dans l'afficheur (ce n'est pas nécessaire, donc on relie R/W à la masse), il nous faut commander les six broches : EN, RS, D4, D5, D6, et D7.

| N° | Broche | Fonction |
|---|---|---|
| 1 | GND | Masse de l'alimentation |
| 2 | +5V | Alimentation 5V |
| 3 | CO | Réglage du contraste |
| 4 | RS | 0 : Instruction 1 : Donnée |
| 5 | R/W | 0 : Ecriture 1 : Lecture |
| 6 | EN | Validation |
| 7 | D0 | Données<br>Dans le cas de l'adressage en 4 Bits, seuls les bits D4...D7 sont utilisés. Les bits D0...D3 ne sont alors pas connectés. |
| 8 | D1 | |
| 9 | D2 | |
| 10 | D3 | |
| 11 | D4 | |
| 12 | D5 | |
| 13 | D6 | |
| 14 | D7 | |

Le compilateur MikroC utilise des fonctions (bibliothèques) pour simplifier aux utilisateurs l'exploitation des afficheurs LCD. On utilise souvent l'instruction "**sbit**" pour affecter chaque broche du LCD à une broche du PIC.

**Remarque** : Utiliser "**LCD Library**" à partir du help du MikroC.

| Association des broches du LCD au port b du pic | définition du sens |
|---|---|
| sbit LCD_RS at RB4_bit; | sbit LCD_RS_Direction at TRISB4_bit; |
| sbit LCD_EN at RB5_bit; | sbit LCD_EN_Direction at TRISB5_bit; |
| sbit LCD_D7 at RB3_bit; | sbit LCD_D7_Direction at TRISB3_bit; |
| sbit LCD_D6 at RB2_bit; | sbit LCD_D6_Direction at TRISB2_bit; |
| sbit LCD_D5 at RB1_bit; | sbit LCD_D5_Direction at TRISB1_bit; |
| sbit LCD_D4 at RB0_bit; | sbit LCD_D4_Direction at TRISB0_bit; |

Ainsi, d'autres fonctions peuvent être utilisées :

**Lcd_Out** : affiche un texte à la position spécifiée, exemple Lcd_Out (1, 1, "TP3")

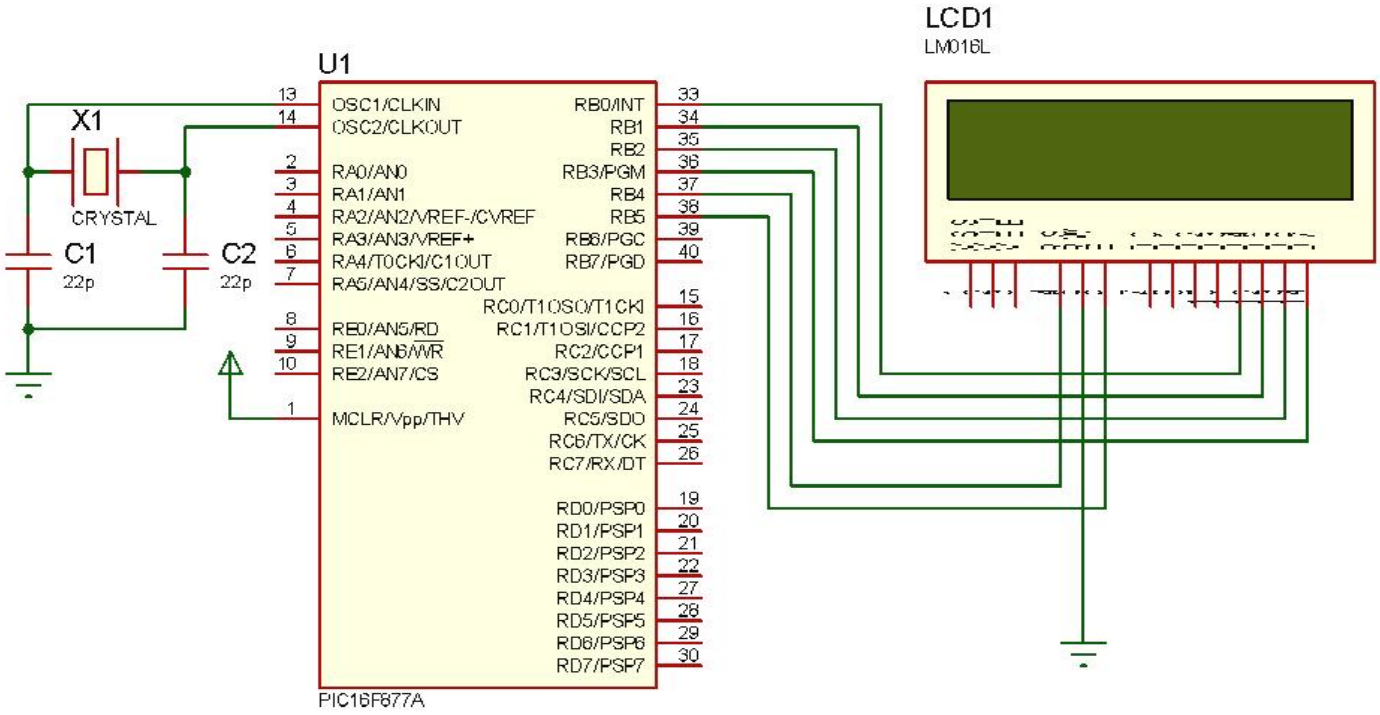
**Lcd_Out_Cp** : affiche un texte à la position courante, exemple Lcd_Out (1, 1, "ici")

**Lcd_Chr** : affiche un caractère à la position spécifiée, exemple Lcd_Chr (1, 3, "T")

**Lcd_Cmd** : envoi une commande au LCD, exemple Lcd_Cmd(_LCD_CLEAR) ;

### II- Gestion de l'affichage d'un message via le PIC 16F877A et l'afficheur LCD 2x16

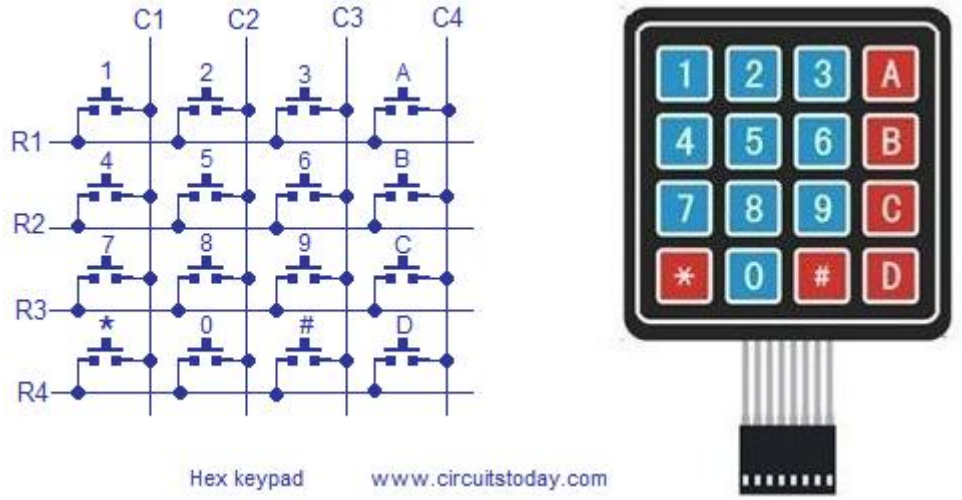
En utilisant le compilateur MikroC et Proteus ISIS, réaliser un programme qui permet d'afficher un message sur l'afficheur LCD 2x16. A titre d'exemple, réaliser le schéma suivant ci-dessous et la librairie LCD : MikroC → Help → Lcd Library.

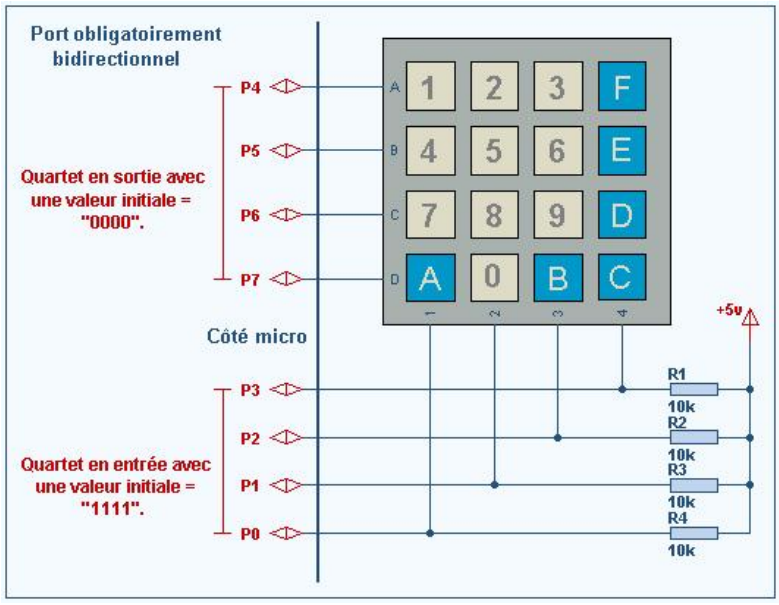


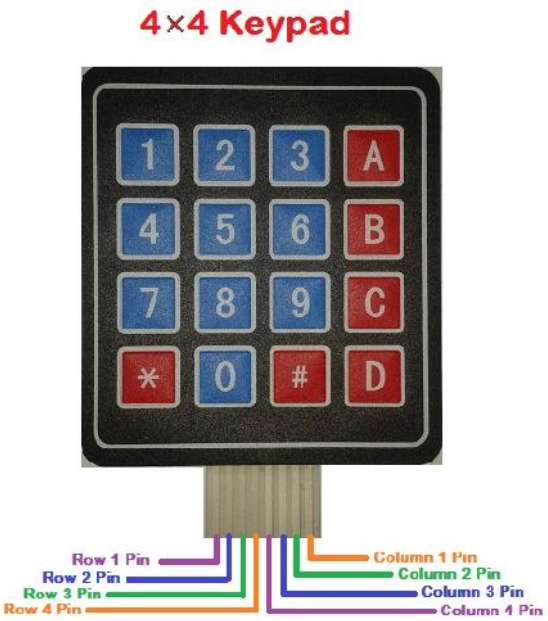

```
// les connections de l afficheur LCD 2x16
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;
 //les directions
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// fin des connections du module LCD
void main() {
Lcd_Init();    // initialisation du LCD
Lcd_Cmd(_LCD_CLEAR); // effacer l'écran

while(1)   {
Lcd_Out(1, 1, " MASTER1 TP SESTR ");
Lcd_Out(2, 1, " TP3 , 2019 ");
Lcd_Cmd(_LCD_CURSOR_OFF); // éliminer le curseur
delay_ms(1000);
Lcd_Out(1, 1, "Univ Constantine");
Lcd_Out(2, 1, "Merci bien ");
Lcd_Cmd(_LCD_CURSOR_OFF); // éliminer le curseur
delay_ms(1000);
//Lcd_Out_Cp("ICI");
//Lcd_Cmd(_LCD_CURSOR_OFF); // éliminer le curseur
}
}
```

### III- Gestion de l'affichage via le PIC 16F877A, l'afficheur LCD 2x16 et un clavier matriciel (keypad)

A titre d'exemple, un clavier matriciel 4x4 est composé de 4 lignes ( L1 L2 L3 L4) et de 4 colonnes ( C1 C2 C3). Lorsqu'une touche est enfoncée, une connexion entre une ligne et une colonne est établi.




Le brochage dit "matrice carrée" (comme le montre la figure) nécessite l'utilisation d'un port parallèle, obligatoirement bidirectionnel, complet (8 bits de P0 à P7). Le poids faible de ce port sera réservé pour les colonnes et initialisé en entrée avec la valeur "1111" tandis que son poids fort est affecté aux lignes et initialisé en sortie avec la valeur "0000".

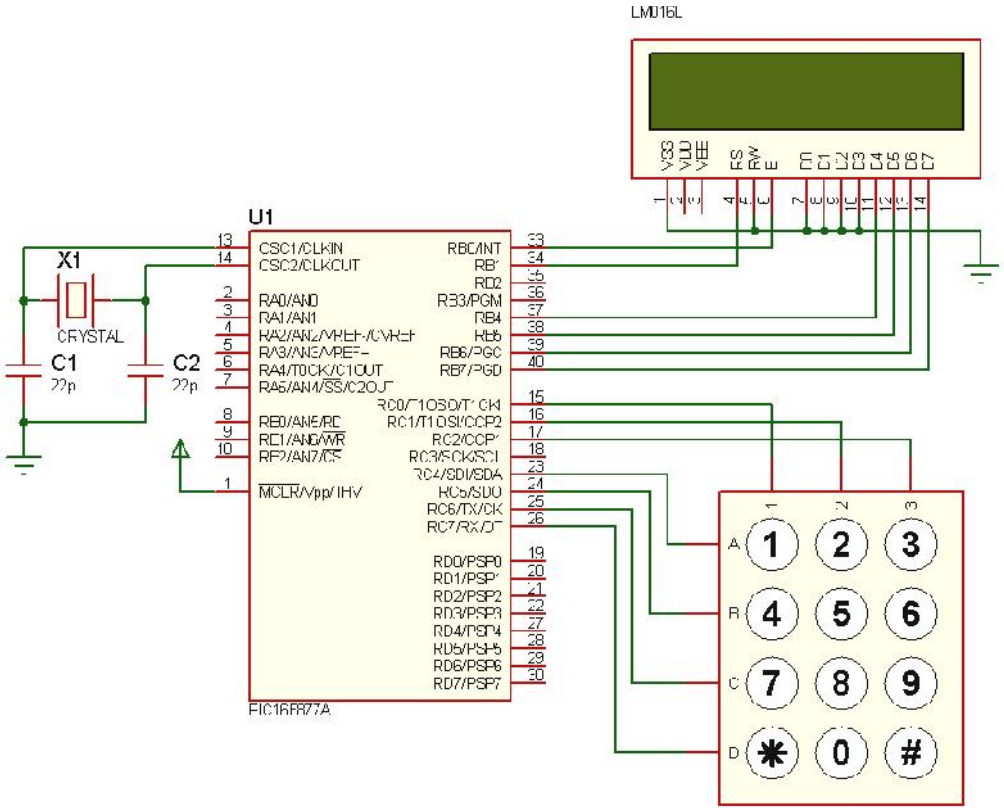
**Codage du clavier matriciel** : On sait déjà qu'une touche enfoncée provoque un court-circuit entre la ligne et la colonne correspondantes. Si ce court-circuit est matérialisé par un niveau logique 0 et tout le reste est à 1, on peut alors facilement, établir le tableau de la figure suivante :

| | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 | |
|---|---|---|---|---|---|---|---|---|---|
| **Touches** | **Lignes** | | | | **Colonnes** | | | | **Codes** |
| (Valeur ASCII) | **D** | **C** | **B** | **A** | **4** | **3** | **2** | **1** | **Hexa** |
| '1' => 49 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | **EE** |
| '2' => 50 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | **ED** |
| '3' => 51 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | **EB** |
| 'A' => 65 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | **E7** |
| '4' => 52 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | **DE** |
| '5' => 53 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | **DD** |
| '6' => 54 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | **DB** |
| 'B' => 66 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | **D7** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| '7' => 55 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | **BE** |
| '8' => 56 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | **BD** |
| '9' => 57 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | **BB** |
| 'C' => 67 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | **B7** |
| '*' => 42 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | **7E** |
| '0' => 48 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | **7D** |
| ' # ' => 35 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | **7B** |
| 'D' => 68 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | **77** |

**Décodage du clavier matriciel** : L'acquisition d'une touche se fera par scrutation (examen), en effectuant une lecture permanente de l'état du clavier. A l'appui d'une touche, le microcontrôleur se dépêche pour explorer judicieusement les codes, déjà établis théoriquement, dans le tableau précédent.

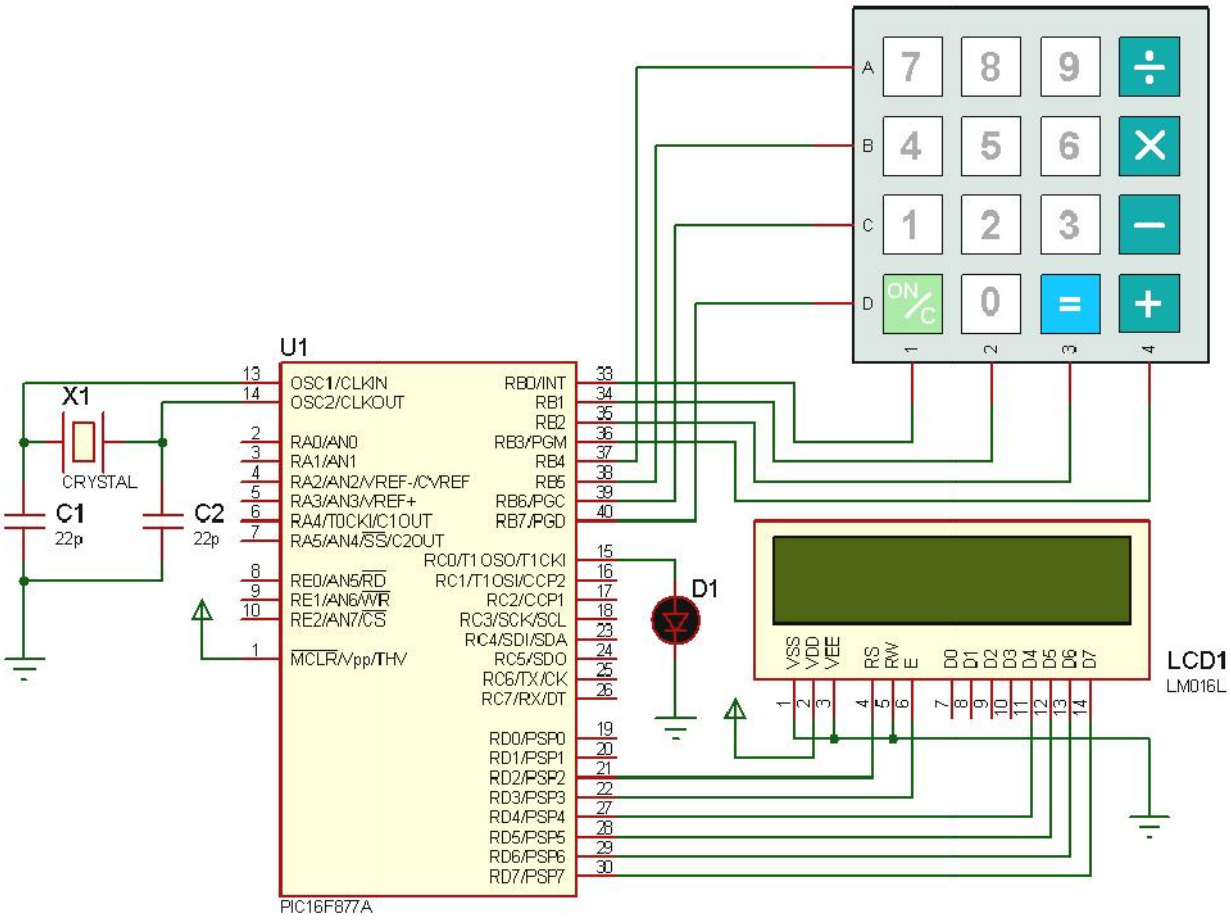
En utilisant le compilateur MikroC et Proteus ISIS, réaliser un programme qui permet d'afficher le la touche enfoncée sur l'afficheur LCD 2x16. A titre d'exemple, réaliser le schéma suivant :




```
// connexions du module Keypad
char keypadPort at PORTC;
// connexions du module LCD
sbit LCD_RS at RB1_bit;
sbit LCD_EN at RB0_bit;
sbit LCD_D4 at RB4_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D7 at RB7_bit;
sbit LCD_RS_Direction at TRISB1_bit;
sbit LCD_EN_Direction at TRISB0_bit;
sbit LCD_D4_Direction at TRISB4_bit;
sbit LCD_D5_Direction at TRISB5_bit;
sbit LCD_D6_Direction at TRISB6_bit;
sbit LCD_D7_Direction at TRISB7_bit;

void main() {
  unsigned short kp = 0;
Keypad_Init(); // Initialisation du Keypad
Lcd_Init(); // Initialisation du Lcd
Lcd_Cmd(_LCD_CLEAR); // effacement de l'écran
Lcd_Cmd(_LCD_CURSOR_OFF); // Cursor off
Lcd_Out(1, 1, "Clavier 3x4");
Lcd_Out(2, 1, "Touche :"); // écrire un message dans LCD
do
{
kp = 0; // Reset la variable
// attendre que le bouton soit pressé et relâché
do
kp = Keypad_Key_Click(); // sauvegarder le code
while (!kp);
// transformation du code en valeur ASCII
switch (kp)
{
case  1: kp = 49; break; // 1
case  2: kp = 50; break; // 2
case  3: kp = 51; break; // 3
//case  4: kp = 65; break; // A
case  5: kp = 52; break; // 4
case  6: kp = 53; break; // 5
case  7: kp = 54; break; // 6
//case  8: kp = 66; break; // B
case  9: kp = 55; break; // 7
case 10: kp = 56; break; // 8
case 11: kp = 57; break; // 9
//case 12: kp = 67; break; // C
case 13: kp = 42; break; // *
case 14: kp = 48; break; // 0
case 15: kp = 35; break; // #
//case 16: kp = 68; break; // D
}
Lcd_Chr(2, 10, kp); // affichage de la valeur ASCII sur LCD
} while (1);
}
```

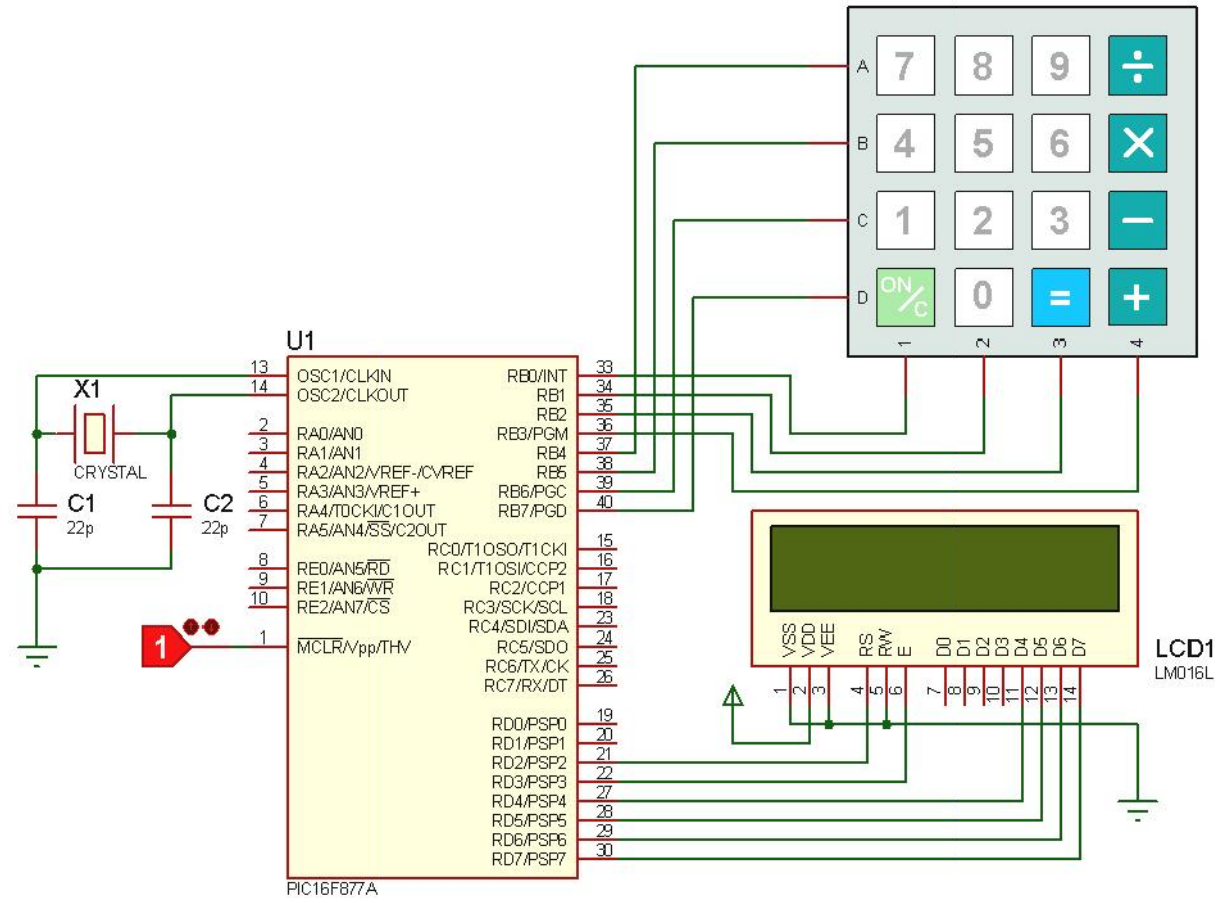
**Exercice 1**

Construire un programme en MikroC PRO qui permet la commande d'une serrure codée à l'aide de 3 chiffres ?




**Exercice 2**

Construire un programme en MikroC PRO qui permet de générer le fonctionnement d'une calculatrice simple ?



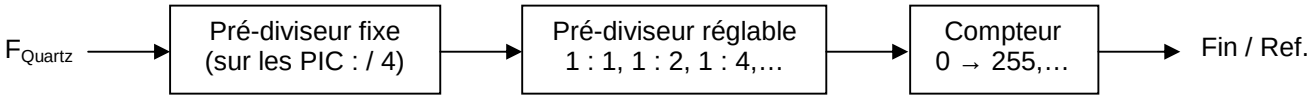
# TP 4, PROGRAMMATION DU PIC 16F877A

# TIMER2, PWM ET CONTROLE DE LA VITESSE D'UN MOTEUR CC

**Objectif** : L'objectif capital de ce 2ème TP est de réaliser un programme en MikroC qui permet de configurer le Timer2 du PIC 16F877A pour générer une modulation de largeur d'impulsion PWM (Pulse Width Modulation). Le signal PWM généré est utilisé pour contrôler la vitesse d'un moteur à courant continu (MCC). La vérification est faite via Proteus ISIS.

## I- PIC 16F877A et PWM mode

En bref, le PIC 16F877A contient 3 Timers internes : **Timer0**, **Timer2** sur 8 bits et **Timer1** sur 16 bits. Pour réaliser une temporisation, le principe de fonctionnement est basé sur l'utilisation d'un compteur qui s'incrémente à chaque front montant du signal qui lui est appliqué.



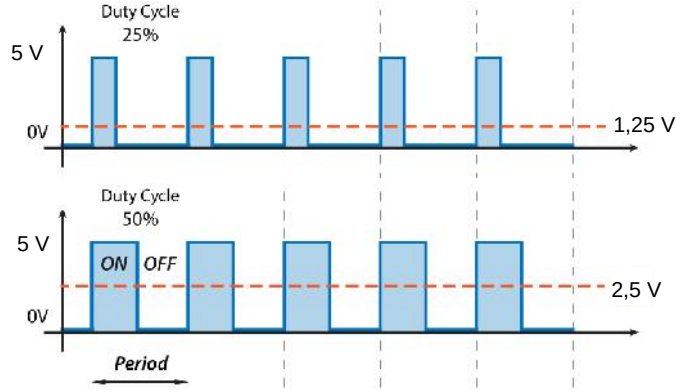

Un Timer doit pouvoir compter un temps défini par le programme (ex. 1 ms, 20 ms, 50 ms,…etc.). Deux paramètres peuvent être modifiés : la fréquence du signal appliqué au compteur et le nombre d'impulsions à compter.

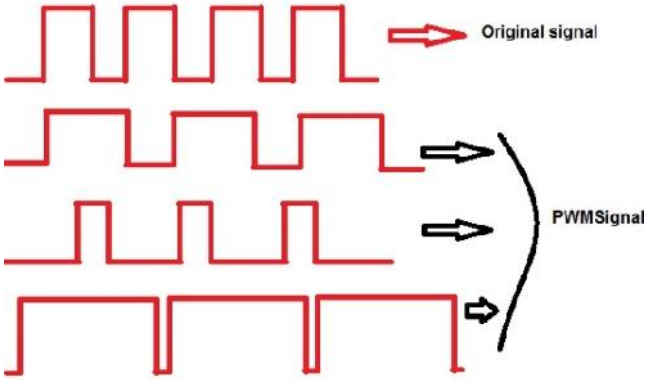
Les méthodes de configuration sont les suivantes :

- Modification de la fréquence du signal appliqué au compteur (pré-diviseur ou "prescalar" en anglais).
- Modification du nombre d'impulsions à compter (charger une valeur initiale pour changer le temps de comptage).

En effet, le Timer2 possède un pré-compteur fixe et un pré-compteur variable (1 : 1, 1 : 4 et 1 : 16).

PWM est un signal dont le rapport cyclique varie. Il est utilisé dans plusieurs applications (la commande des servomoteurs, la variation de la vitesse des moteurs à courant continu, la conversion numérique analogique,…etc.).



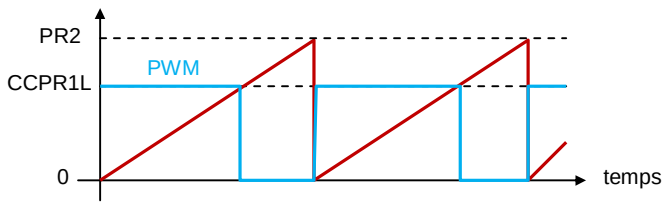

Le PIC 16F877A utilise le module CCP (Capture/Compare/PWM) en mode PWM. La broche RC2 doit être configurée en sortie. La valeur de la période est chargée dans le registre PR2 et la valeur du rapport cyclique dans le registre CCPR1L.

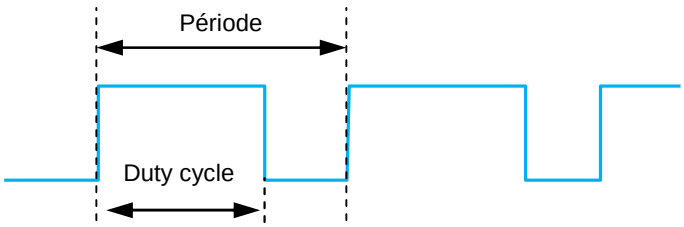
Le Timer2 (8 bits) compte l'horloge interne après une pré-division (prescaler) programmable par 1:1, 1:4 ou 1:16. Quand Timer2 atteint la valeur dans PR2, Timer2 passe à zéro et la broche PWM est mise à 1. Quand Timer2 atteint la valeur dans CCPR1L, la sortie PWM est mise à 0.

$$\text{La période} = 4 \times (PR2 + 1) \times Tosc \times \text{Valeur du pré-diviseur}$$

$$\text{La durée} = CCPR1L \times Tosc \times \text{Valeur du pré-diviseur}$$

$$\text{Avec : } Tosc = \frac{1}{\text{Fréquence horloge (Quartz)}}$$



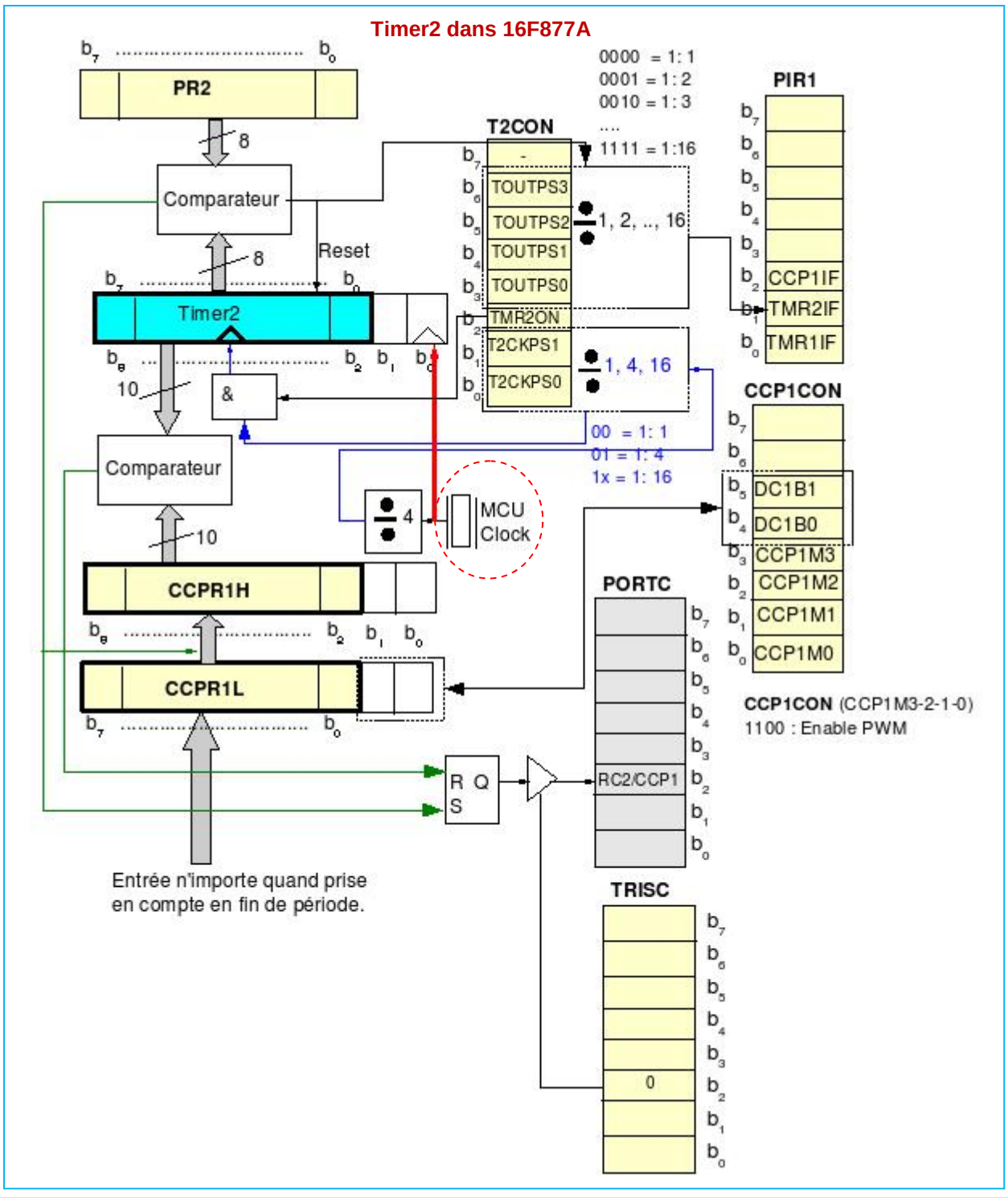
En effet, pour générer le signal PWM, on peut utiliser ce principe qui agit directement sur les registres. Par conséquent, on peut utiliser les fonctions fournis par le compilateur MikroC PRO :

- Pwm_init(freq) : pour le choix de la fréquence en Hz.
- Pwm_Start() : pour le démarrage du module PWM.
- Pwm_Set_Duty(durée) : pour changer le rapport cyclique (de 0 pour 0% à 255 pour 100%).
- Pwm_Stop() : pour l'arrêt du module PWM.

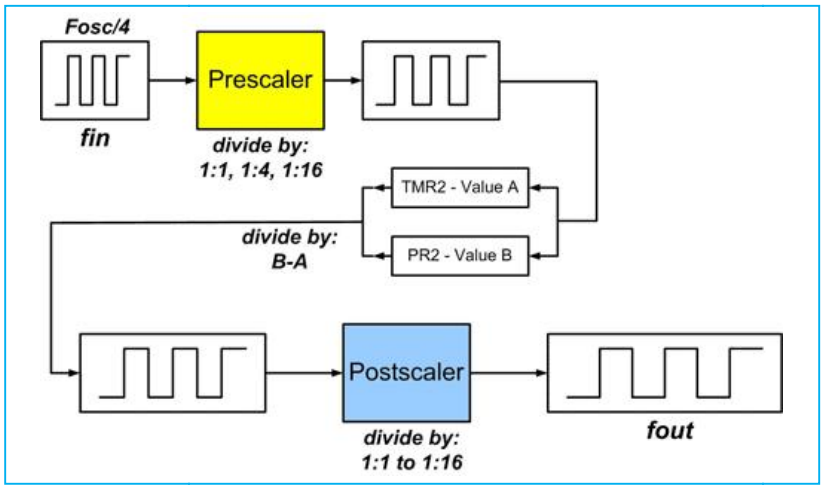
## II- Configuration du Timer2 et PWM (PIC 16F877A) : Configuration du CCP module

La configuration du Timer2 dans le PIC 16F877A est faite via les registres suivants : T2CON (TMR2), CCP1CON, PR2, CCPR1L. Ceci est réalisé à l'aide du comptage des cycles d'horloge (après la pré-division programmable).

La configuration du CCP module permet de générer un signal PWM de différentes formes ou la génération d'une temporisation bien déterminée ce qui permettra la commande d'un moteur CC, d'un servomoteur, …etc.



La configuration du "post-scaler" sera étudiée dans un prochain TP.




### III- Exemple descriptif de Configuration du Timer2

Ecrire un programme en MikroC qui permet de configurer le Timer2 pour générer d'un signal PWM de période maximale avec un rapport cyclique 50% (duty cycle). Visualiser le signal générer via Proteus ISIS. Vérifier les résultats obtenus ? Refaire la même chose pour les cas des rapports cycliques suivants : 50 %, 25% et 12,5% ?

La période $= 4 \times (PR2 + 1) \times Tosc \times$ Valeur du pré-diviseur

Timer2 : registre 8 bits → période maximale pour PR2 = 256 – 1 = 255 (8 bits : comptage de 00000000 à 11111111).

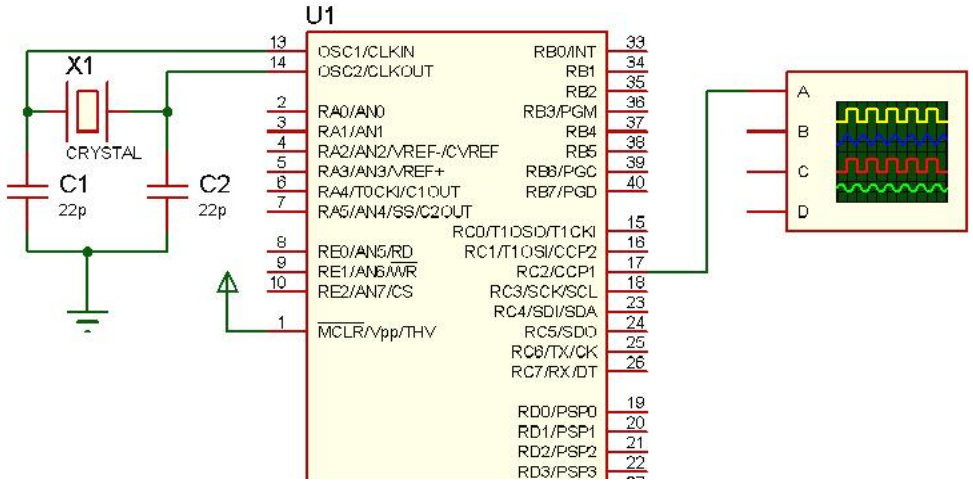
Soit, par exemple, un pré-diviseur **1 : 4** et un Oscillateur **8 MHz**. On obtient donc : CCPR1L = 256 / 2 = 128.

$T_{PWM}$ = (PR2 +1) x (Tosc x 4 x $Timer2_{Prescaler}$) = (255 + 1) x ($1/8.10^6$) x 4 x 4 = 0,512 ms (soit f = 1,95 KHz).

```
void main() {
  unsigned char dc;  // dc : valeur pour duty cycle (rapport cyclique)
  TRISC=0;     // Port C en mode sortie
  PORTC=0;     // Clear port C
   // Configuration du CCP module à f_out  et  PWM Sortie
  PR2=255;    // Période Max pour Timer2 (compteur 8 bits)
  T2CON=0b00000101;  // 1 : TMR2 ON 00 -> 1:1  01 -> 1:4  1x -> 1:16
  CCP1CON=0b00001100;  // 1100 : Enable PWM
  dc=128;
  while(1)
  {
    CCPR1L=dc;
  }
}
```

**T2CON = 0b00000101 ;**
- 1 : 4
- ON

**Remarque** : il est possible de configurer et d'utiliser CCP2CON, CCPR2L et RC1 (PIC broche 16) de la même manière.



**Travail demandé** :

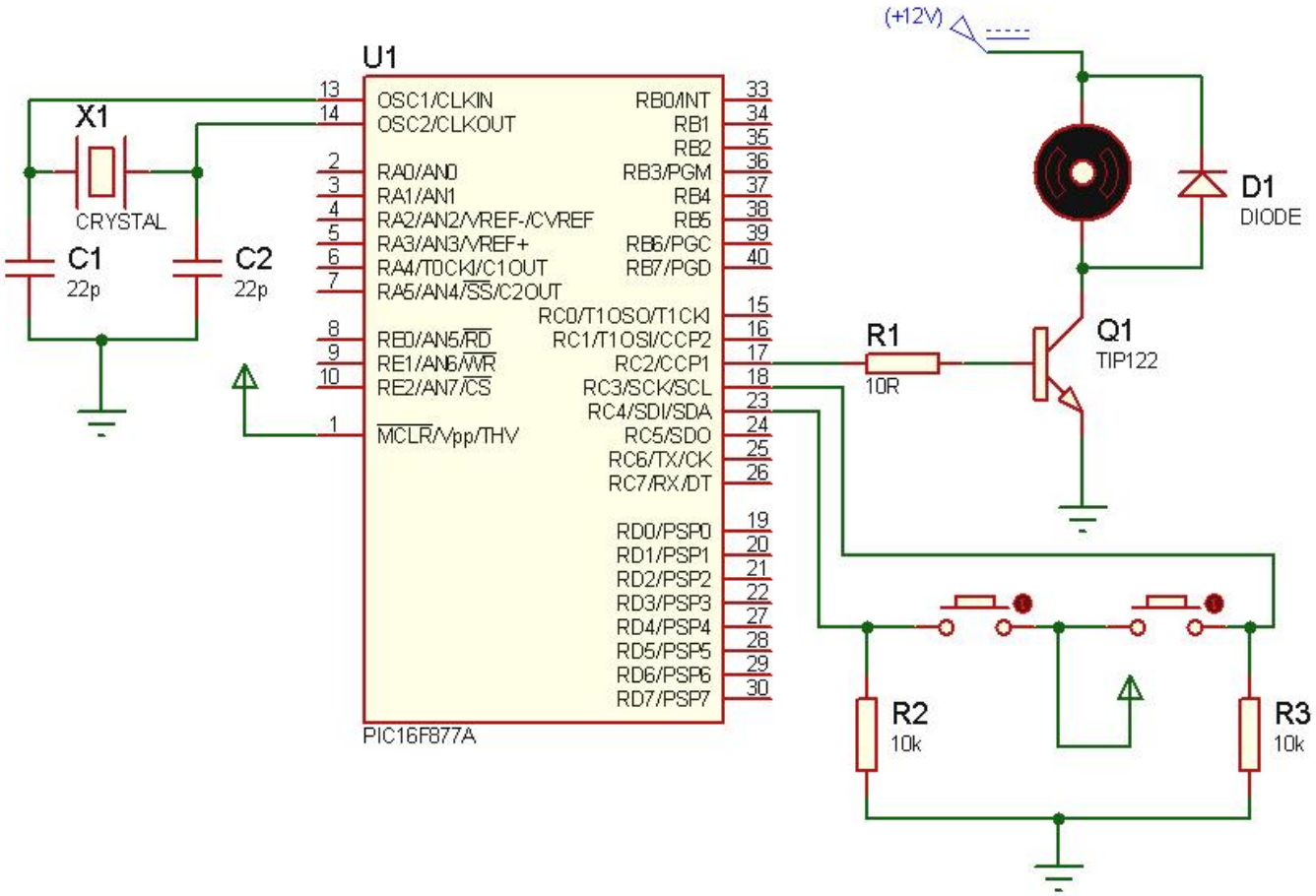
**1)** Construire un programme en MikroC PRO qui permet la génération d'un signal PWM carré de fréquence 1 KHz.

**2)** Construire un programme en MikroC PRO qui permet la génération d'un signal PWM de fréquence 2 KHz, dont le rapport cyclique est de 75%.

**3)** Ecrire un programme en MikroC PRO qui permet la génération d'un signal PWM de fréquence 2 KHz, dont le rapport cyclique varie continuellement entre 10% et 90% (pour ralentir la variation rapide, utiliser un palier de temps 100 ms).

**4)** Quelle est la temporisation maximale ($T_{PWM\_MAX}$) qu'on peut générer avec Timer2 ?

**5)** Ecrire un programme en MikroC PRO qui permet la génération d'un signal PWM de fréquence 2 KHz, dont le rapport cyclique est modifié à l'aide de 2 boutons (augmentation et diminution). Vérifier le fonctionnement avec Proteus ISIS ?



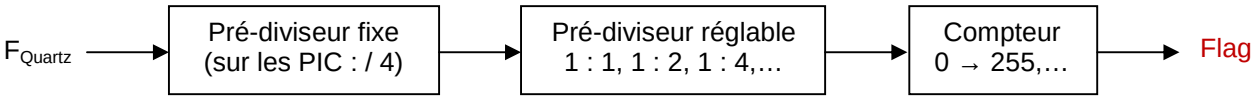
# TP 3, PROGRAMMATION DU PIC 16F877A

# CONFIGURATION DES TIMERS POUR LA GENERATION D'UNE TEMPORISATION

**Objectif** : L'objectif primordial de ce 5ème TP est de construire des programmes simples en MikroC qui permettent de configurer un tel Timer du PIC 16F877A pour générer une temporisation bien déterminée (1 ms, 20 ms,…etc.). La vérification des résultats des programmes réalisés est faite via Proteus ISIS.

## I- La génération d'une temporisation via le Timer2 du PIC 16F877A

Comme nous l'avons mentionné avant, le PIC 16F877A contient 3 Timers internes : Timer0, Timer2 sur 8 bits et Timer1 sur 16 bits. Pour réaliser une temporisation, le principe de fonctionnement est basé sur l'utilisation d'un compteur qui s'incrémente à chaque front montant du signal qui lui est appliqué. Lorsque le compteur dépasse la valeur maximale qu'il peut contenir (par exemple 256 pour un compteur sur 8 bits), un drapeau (flag en anglais) se lève. Ce dernier a pour but d'indiquer au programme que le compteur a débordé (fini de compter).



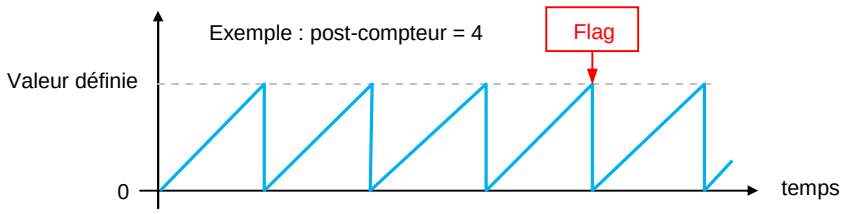

Un Timer doit pouvoir compter un temps défini par le programme (ex. 1 ms, 20 ms, 50 ms,…etc.). Deux paramètres peuvent être modifiés : la fréquence du signal appliqué au compteur et le nombre d'impulsions à compter.

Les méthodes de configuration sont les suivantes :

- Modification de la fréquence du signal appliqué au compteur (pré-diviseur ou "prescalar" en anglais).
- Modification du nombre d'impulsions à compter (charger une valeur initiale pour changer le temps de comptage).

Comme nous l'avons vu dans le TP précédent, le Timer2 possède un pré-compteur fixe et un pré-compteur variable (1 : 1, 1 : 4 et 1 : 16). La durée de comptage maximale est : **$T_{Max}$ = (PR2 +1) x Tosc x 4 x $Timer2_{Prescaler}$ = 2,048 ms**.

Dans le cas particulier du Timer2, le flag ne se lève pas systématiquement à chaque fin de comptage. Il existe un "post-compteur" ou "postscalar" qui peut prendre une valeur entière entre 1 et 16 (Flag TMR2IF_bit & TMR2 registres associés de contrôle).




La temporisation maximale du Timer2 est donc : **$T_{Max}$ = Post-compteur x (PR2 +1) x (Tosc x 4 x $Timer2_{Prescaler}$)**.

Puisque : 16 x 2,048 ms = 32,768 ms,

La formule permettant de calculer la durée de la temporisation : T = Post-scalar x (PR2 +1) x (Tosc x 4 x $Timer2_{Prescaler}$).

Par exemple, pour avoir une **temporisation** de **10 ms** (période de 20 ms à 50%), on fait la configuration suivante :

```
void tempo_timer2 ( void ); // tempo avec le timer2
void init ( void ); // initialisation des périphériques
void main ( void )
{
init(); // initialisation des périphériques
while ( 1 ) // boucle infinie
{
PORTB.F1 = !PORTB.F1; // on change l'état de RB1
tempo_timer2(); // tempo de 10 ms
}
}
void tempo_timer2 ( void ) // tempo avec le timer2
{
TMR2IF_bit = 0; // initialisation du drapeau lié au timer2
while ( TMR2IF_bit == 0 ); // attente de la levée du drapeau
}
```

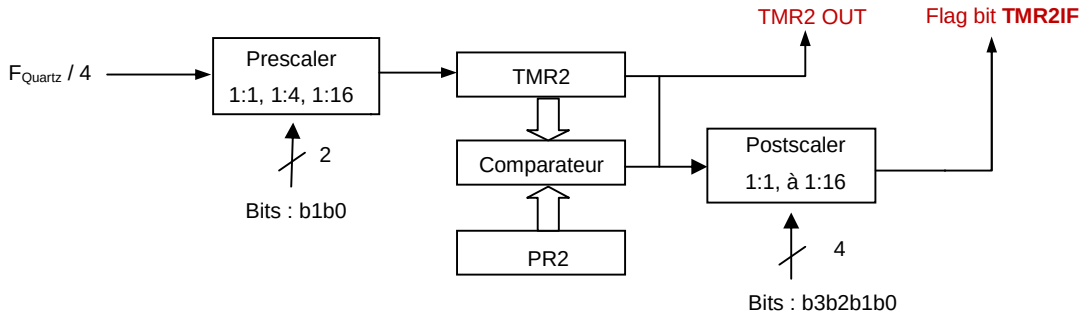
```
void init ( void ) // initialisation des périphériques
{
TRISB.F1 = 0; // RB1 configuré en sortie
PR2 = 95; // Valeur définie de fin de comptage   96-1 = 95
T2CON = 0x66; // configuration du timer2  0x66 (0b01100110)
}
```

**T2CON = 0b01100110** : en binaire ⇔ **T2CON = 0x66** : en hexadécimal.

Oscillateur = **8 MHz**, Pré-compteur fixe : **4**, Prescaler : **16** (10), Activation du Timer2 : bit (**1**) , Post-scaler : **13** (**1100**).

Calcul de la valeur de PR2 (en utilisant un rapport cyclique de 50%) : **PR2 = 95**

$$T_H = T_B = 10 \times 10^{-3} = \frac{1}{8 \times 10^6} \times 4 \times 16 \times 13 \times (PR2 + 1)$$




**REGISTER 7-1: T2CON: TIMER2 CONTROL REGISTER**

| U-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| — | TOUTPS3 | TOUTPS2 | TOUTPS1 | TOUTPS0 | TMR2ON | T2CKPS1 | T2CKPS0 |
| bit 7 | | | | | | | bit 0 |

| Legend: | | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as '0' | |
| -n = Value at POR | '1' = Bit is set | '0' = Bit is cleared | x = Bit is unknown |

bit 7 **Unimplemented:** Read as '0'

bit 6-3 **TOUTPS<3:0>: Timer2 Output Postscaler Select bits**

0000 = 1:1 Postscaler
0001 = 1:2 Postscaler
0010 = 1:3 Postscaler
0011 = 1:4 Postscaler
0100 = 1:5 Postscaler
0101 = 1:6 Postscaler
0110 = 1:7 Postscaler
0111 = 1:8 Postscaler
1000 = 1:9 Postscaler
1001 = 1:10 Postscaler
1010 = 1:11 Postscaler
1011 = 1:12 Postscaler
1100 = 1:13 Postscaler
1101 = 1:14 Postscaler
1110 = 1:15 Postscaler
1111 = 1:16 Postscaler

bit 2 **TMR2ON: Timer2 On bit**

1 = Timer2 is on
0 = Timer2 is off

bit 1-0 **T2CKPS<1:0>: Timer2 Clock Prescale Select bits**

00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

La configuration des registres du Timer2 pour avoir une temporisation de **5 ms** :
**T2CON = 0b01100110** : en binaire ⇔ **T2CON = 0x66** : en hexadécimal.
Quartz = **8 MHz**, Pré-compteur fixe : **4**, Pré-compteur (prescaler) : **16** (**10**)
Activation Timer2 : bit (**1**) , Post-compteur : **13** (**1100**), **PR2 = 47** en décimal
Le calcul de la valeur de PR2 se fait facilement, en utilisant un rapport cyclique de 50% ($T = T_H + T_B$) : La période

$$T = T_H + T_B = 10 \times 10^{-3} = \frac{1}{8 \times 10^6} \times 4 \times 16 \times 13 \times (PR2 + 1)$$

**Travail demandé** :

Soit à réaliser un programme en MikroC qui permet la configuration du Timer2 (PIC 16F877A) pour la génération d'un signal carré de période **5 ms** (une temporisation de 2,5 ms). Sachant que l'oscillateur utilisé est de **8 MHz**.

**1)** Quelle est la durée de comptage maximale (temporisation maximale) du Timer2 du PIC utilisé ?

**2)** Donner la configuration nécessaire des registres du Timer2 puis construire le programme nécessaire ?

**3)** Visualiser le signal généré et mesurer la période T et la fréquence f en utilisant Proteus ISIS ?

## II- La génération d'une temporisation via le Timer0 du PIC 16F877A

Contrairement au Timer2, le Timer0 a un pré-diviseur qui peut prendre une valeur de : 1, 2, 4, 8, 16, 32, 64, 128, 256. Ainsi, la valeur initiale peut prendre n'importe quelle valeur entière comprise entre 0 et 255.

Les registres associés au Timer0 sont :

- TMR0 : c'est le registre de comptage. C'est dans ce registre que nous allons rentrer la valeur de départ de notre compteur.
- INTCON : seuls les bits 7, 6, 5 et 2 sont utiles pour le Timer0. En effet, le bit 2 est appelé TOIF correspond au flag permettant de tester la fin du comptage.
- OPTION_REG : Registre de configuration du Timer0 (T0CON : Timer0 Control Register).

| /RBPU | INTEDG | T0CS | T0SE | PSA | PS2 | PS1 | PS0 |
|---|---|---|---|---|---|---|---|
| X (1) | X (0) | 0 | X (0) | 0 | 1 | 0 | 1 |

Prédiviseur 64

**111** = 1 : 256
**110** = 1 : 128
**101** = 1 : 64
**100** = 1 : 32
**011** = 1 : 16
**010** = 1 : 8
**001** = 1 : 4
**000** = 1 : 2

| Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| TMR0 | Timer0 Module Register | | | | | | | |
| INTCON | GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | RBIF |
| OPTION_REG | $\overline{\text{RBPU}}$ | INTEDG | T0CS | T0SE | PSA | PS2 | PS1 | PS0 |
| TRISA | TRISA7 | TRISA6 | TRISA5 | TRISA4 | TRISA3 | TRISA2 | TRISA1 | TRISA0 |

A titre d'exemple, pour avoir une **temporisation** de **10 ms** (période de 20 ms à 50%), on fait la configuration suivante :

$$T_H = T_B = 10\times10^{-3} = \frac{1}{8\times10^{6}} \times 4 \times 128 \times Y \qquad \text{Avec : } Y = \text{valeur à compter} \Rightarrow Y = 256 - \text{valeur initiale}$$

valeur à compter $Y = 156$ (à compter 156 de 256) Donc : valeur initiale TMR0 = 100

Le registre de configuration du Timer0 est : OPTION_REG = 1000 0110 ⇔ **0x86** en hexadécimal

```
void tempo_timer0 ( void ); // tempo de 10ms avec le timer0
void init ( void ); // initialisation des périphériques
void main ( void )
{
init(); // initialisation des périphériques
while ( 1 ) // boucle infinie
{
PORTB.F1 = !PORTB.F1; // on change l'état de RB1
tempo_timer0(); // tempo de 10 ms
}
}
void tempo_timer0 ( void ) // tempo de 10ms avec le timer0
{
TMR0 = 100; // initialisation du registre de comptage
T0IF_bit = 0; // initialisation du drapeau lié au timer0
while ( T0IF_bit == 0 ); // attente de la levée du drapeau
}
void init ( void ) // initialisation des périphériques
{
TRISB.F1 = 0; // RB1 configuré en sortie
OPTION_REG = 0x86; // configuration du timer0 : pré-diviseur à 128
}
```

**Travail demandé** :

Soit à réaliser un programme en MikroC qui permet la configuration du Timer0 (PIC 16F877A) pour la génération d'un **signal carré de période 50 ms** (une temporisation de 25 ms). Sachant que l'oscillateur utilisé est de **8 MHz**.

**1)** Donner la configuration nécessaire des registres du Timer0 puis construire le programme nécessaire ?

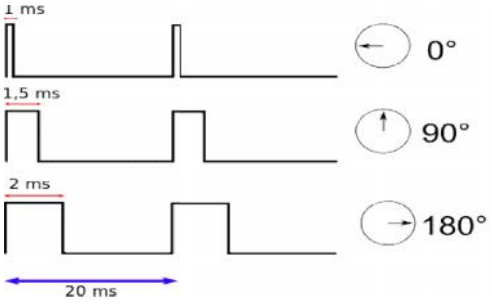
**2)** Visualiser le signal généré et mesurer la période T et la fréquence f en utilisant Proteus ISIS ?

**3)** Calculer le temps maximum (temporisation maximale) du Timer0 ; pour un oscillateur 8 MHz ?

### III- PIC 16F877A et Servomoteur : Réalisation d'une temporisation de 20 ms

Pour que le servomoteur reste à une position donnée, il faut transmettre toutes les 20 ms (soit à une fréquence de 50 Hz) une impulsion d'une longueur comprise entre 1 et 2 ms.

- Une impulsion de 1 ms correspond à un angle de 0°.
- Une impulsion de 2 ms correspond à un angle de 180°.
- En envoyant une impulsion d'une longueur intermédiaire, on obtient des angles différents (90° avec une impulsion de 1.5 ms par exemple).




Pour un signal PWM de 20 ms :

**1)** Quelles sont les registres du Timer2 qu'il est nécessaire de configurer ?

**2)** Calculer les valeurs du prescaler et du compteur, nécessaires pour obtenir une temporisation de 20 ms ?

**3)** Donner la configuration des registres du Timer2 ? Vérifier avec Proteus ISIS ?

*A suivre . . .*

# TP 6, PROGRAMMATION DU PIC 16F877A
# CONVERTISSEUR ANALOGIQUE NUMERIQUE CAN

**Objectif** : L'objectif essentiel de ce 6ème TP est de programmer, en MikroC, le module CAN (Convertisseur Analogique Numérique) du PIC16F877A. Le signal d'entrée à convertir peut être la variation d'une tension dérivée d'un capteur ; qui sera simulé généralement par un montage à potentiomètre. La vérification des résultats est réalisée via Proteus ISIS.

## I- La Configuration du module AD du PIC 16F877A

La conversion Analogique Digital (A/D) consiste à lire une tension sur les entrées analogiques du PIC et la convertir en nombre entier situé entre 0 et 1023 (10 bits) ; les valeurs 0 et 1023 correspondent aux tensions de référence $V_{Ref+}$ et $V_{Ref-}$.
Le module AD du PIC16f877A demande une configuration particulière :

- Configuration de la broche Analogique ; en liant un potentiomètre à l'entrée du PORT A, (à titre d'exemple : TRISA.F0 = 1, (RA0/AN0 )).
- Configuration des registres : ADCON0 et ADCON1 du PIC16f877A.

Le résultat de la conversion (10 bits) est stocké dans les registres ADRESH et ADRESL, qu'on ne va pas utiliser car MikroC PRO a une fonction **Adc_Read(0)** qui nous retourne directement le résultat.

### a) Registre ADCON0 : 8 bits b7…b0

- Les bits 7 et 6 permettent de choisir la fréquence qui va cadencer le convertisseur analogique.

  00 : Fosc / 2
  01 : Fosc / 8
  10 : Fosc / 32
  11 : Fosc
- Les bits 5, 4 et 3 permettent de sélectionner l'entrée analogique à convertir.

  000 : RA0 / AN0
  001 : RA1 / AN1
  010 : RA2 / AN2
  011 : RA3 / AN3
  100 : RA5 / AN4
  101 : RE0 / AN5
  000 : RE1 / AN6
  000 : RE2 / AN7
- Le bit 2 permet de lancer la conversion et indique la fin de la conversion : 1 → conversion lancée, 0 → conversion terminée.
- Le bit 0 permet de mettre en route la conversion : 0 → convertisseur ON et 1 → convertisseur OFF.

ADCON0 = 0 ⇔ 0b00000000

### b) Registre ADCON1 : 8 bits b7…b0

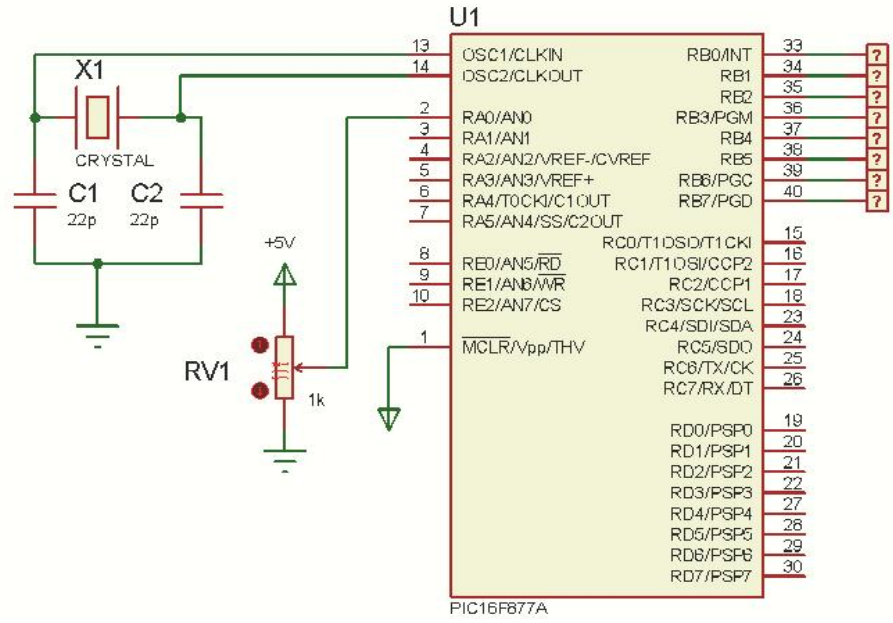
Le bit 7 permet de sélectionner les registres dans lesquels seront stockés les résultats de la conversion (1 → 10 bits dans ADRESH et ADRESL avec ajustement à droite ; 0 → 10 bits dans ADRESH et ADRESL avec ajustement à gauche).
Les bits 3 à 0 permettent de configurer les broches du PIC comme étant des entrées Analogiques ou Digitales (à titre d'exemple : 1110 pour DDDDDDDA ⇔ RE2 RE1 RE0 RA5 RA3 RA2 RA1 RA0 ).
ADCON1 = 0x8E ⇔ 0b10001110 (uniquement RA0 : analogique).
La procédure pour configurer le convertisseur Analogique Digital correctement est la suivante :

- La sélection de la fréquence d'horloge et la mise en route du convertisseur : utilisation du registre ADCON0,
- La configuration des broches du PIC (Analogique / Digital) : utilisation du registre ADCON1,
- Attente de l'acquisition de la grandeur analogique (qq 10 ms),
- Lancement de la conversion et récupération des résultats : utilisation du registre ADCON0.

## II- Exemple de Configuration ADC

N.c

```
unsigned long int value;
void main()
 {
  PORTB = 0;
  TRISB = 0 ;
  TRISA =0X01 ;
  ADCON0 = 0 ;
  ADCON1 = 0x8E ;
  while(1)
   {
    Delay_us(20);
    value = Adc_Read(0);
    value = value/4 ;
    PORTB = value ;
   }
 }
```



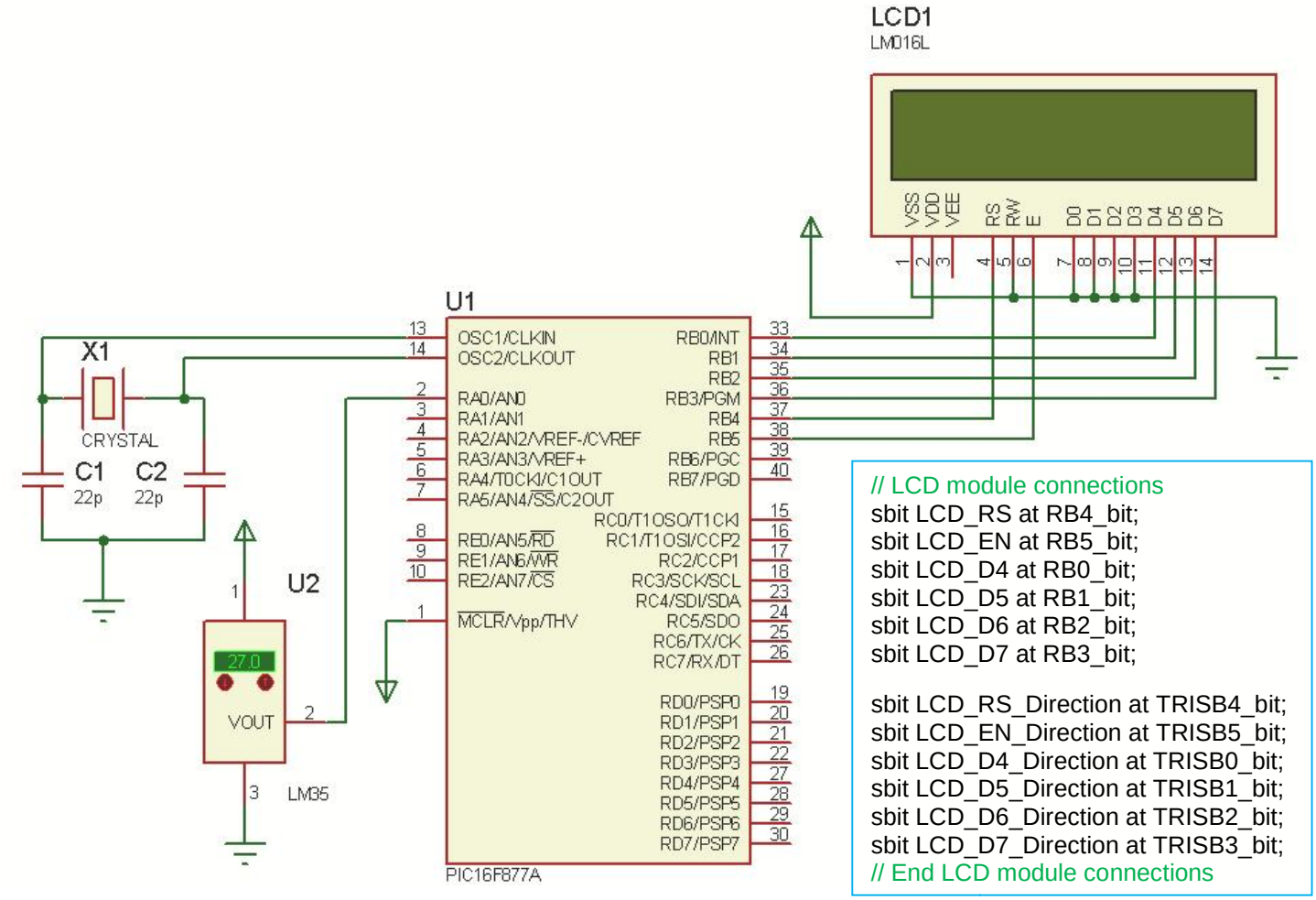

**Remarque** : dans cet exemple, "value" a été divisée par 4 pour avoir une sortie pleine échelle sur 8 bits.

**Travail demandé** :

Construire un programme en MikroC qui affiche la température issue d'un capteur LM35. Sachant que 5 V correspond à la valeur 1024 quand on a besoin d'afficher 5,00 V.

### Capteur de température LM35, Afficheur LCD 2x16 et PIC 16F877A

Construire un programme qui affiche la température issue d'un capteur LM35.

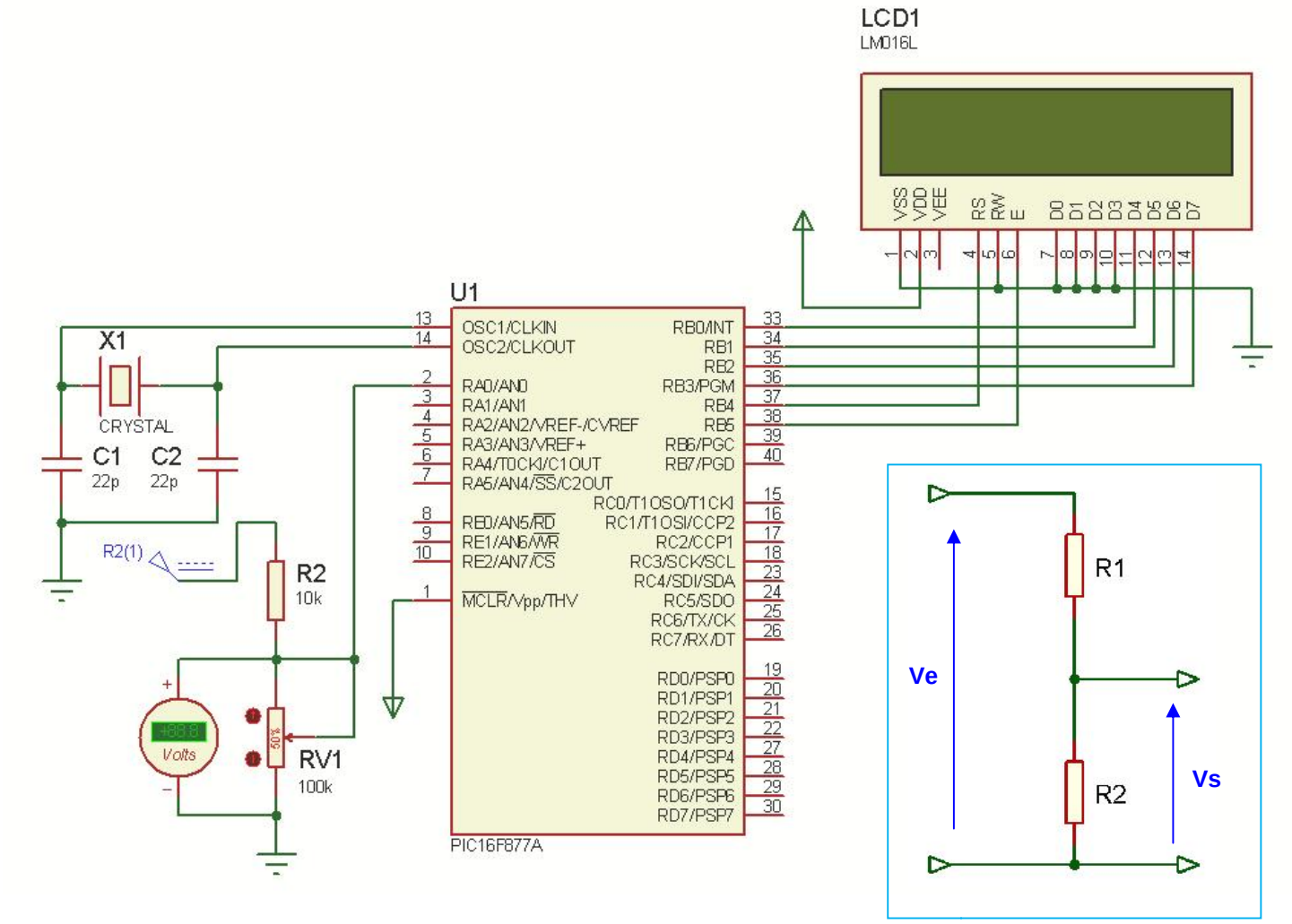



```
// LCD module connections
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;

sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// End LCD module connections
```

```
char display[16]="";

void Move_Delay() {  // Function used for text moving
Delay_ms(1000);    // change the moving speed here
}

void main() {
unsigned int result;
float volt,temp;

trisb=0;
trisa=0xff;
adcon1=0x80;
lcd_init();
lcd_cmd(_lcd_clear);
lcd_cmd(_LCD_CURSOR_OFF);
lcd_out(2,1,"UNIV Constantine");
while(1)
{
result=adc_read(0);
volt=result*4.88;
temp=volt/10;

lcd_out(1,1,"Temp = ");

floattostr(temp,display);
lcd_out_cp(display);
lcd_chr(1,14,223); //print at pos(row=1,col=14) "°" = 223
lcd_out_cp("C");  // Celcius
delay_ms(1000);
//Lcd_Cmd(_LCD_CLEAR);
}
}
```

**Travail demandé : DC voltmètre numérique**

**1)** Construire un programme, en MikroC, qui affiche la valeur de la tension de sortie d'un circuit diviseur de tension constitué de deux résistances R1 et R2. Sachant que 5 V correspond à la valeur 1024 quand on a besoin d'afficher 5 V.




**2)** Reconstruire votre programme pour afficher la valeur de la tension de sortie via un autre circuit diviseur de tension constitué de 2 résistances R1 et R2 ; sachant que 50 V correspond à la valeur 1024 quand on a besoin d'afficher 50 V

Donner les valeurs de R1 et R2 de votre circuit ?