École d'ingénieurs
Télécom Physique Strasbourg

*Promotion 2023*
*Année 2021-2022*

# Réalisation d'un système d'acquisition série sur PIC


*Léo Jellimann*
*Télécom Physique Strasbourg*

UIMM PÔLE FORMATION Alsace LA FABRIQUE DE L'AVENIR

itii ALSACE

# Introduction

*Au cours de cette deuxième année au sein de Télécom Physique Strasbourg, nous étudiants, avons eu la chance d'avoir un cours sur les microcontrôleurs. Grâce à ce cours, nous avons pu comprendre comment fonctionnent ces petits composants électroniques qui sont présents partout dans nos quotidiens.*

*Une fois l'étude du microcontrôleur effectuée en classe, nous avons pu nous attaquer au langage de programmation permettant au composant de réaliser diverses actions, l'assembleur.*

*C'est donc grâce à ce projet de réalisation d'un système d'acquisition d'une grandeur analogique avec le port série d'un PIC, que nous avons pu mettre en pratique nos connaissances.*

*Pour cela, nous avons utilisé un microcontrôleur PIC 16F877 et le logiciel de développement MPLABX IDE.*

## Table des matières

# Prise en main de l'environnement de programmation

On rappelle que l'objectif principal de ce projet est de réaliser un système d'acquisition série sur microcontrôleur.

Avant de commencer directement dans le projet, il a fallu prendre en main l'environnement de développement MPLABX IDE, ainsi que la carte comportant le microcontrôleur PIC16F877.

Pour cela, j'ai donc implémenté le programme livré par M. Anstotz, afin d'observer ce qu'il faisait et de comprendre le code associé. Effectivement, ce programme convertissait la tension entre 0 et 5 Volts entrant dans la carte, sous 4 bits allumés sur des LEDS. Une tension de 0V en entrée allumait 0 des 4 LEDS, alors que 5V eux allumait les 4 LEDS. Pour modifier cette tension entrante, il fallait tourner un potentiomètre qui ajustait en temps réel la valeur de la tension affichée sur les LEDS.

Une fois cette première partie réalisée, j'ai pu commencer le projet avec notamment la lecture de la tension analogique et l'affichage d'un caractère ascii sur le port série du PIC.

# Lecture de la tension analogique

Effectivement, la première partie de ce projet étant de réussir à lire la tension d'entrée de cette carte. Le programme tutoriel donné par M. Anstotz a beaucoup aidé pour cette tâche. Il faut remarquer dans ce programme l'utilisation du registre ADCON0 qui contrôle les opérations de conversion analogique vers numérique. Ainsi, j'ai sélectionné le convertisseur analogique/numérique cadencé à « Fosc/8 ». Cela, tout en activant le bit ADON, qui autorise l'utilisation du convertisseur analogique/numérique.

```
;configuration du convertisseur analogique/numérique
banksel ADCON0
movlw       B'01000001'          ;Utilisation de Fosc/8 et autorisation de l'utilisation du convertisseur analogique/numérique
movwf       ADCON0

banksel ADCON1
movlw       B'00001110'          ;Configuration du convertisseur pour avoir 7 ports numériques et les tensions de référence Vref+ = VDD et Vref- = VSS
movwf       ADCON1
```

Aussi, dans ce tutoriel, le registre ADRESH permet de récupérer le résultat de cette conversion analogique/numérique. Dans ce programme d'exemple, l'affichage de ce résultat se fait sur les LEDS de la carte.

Il faut donc utiliser ADRESH pour récupérer les valeurs de tensions que l'on souhaitera afficher par la suite sur le terminal du PC grâce au port série.

Maintenant que le fonctionnement de lecture d'une tension analogique a été compris, je vais mettre en place un système permettant d'envoyer un caractère ascii sur le port série du PIC.

# Envoi d'un caractère ascii sur le port série du PIC

Avant de pouvoir envoyer un caractère sur le port série du PIC, il faut vérifier que le port série fonctionne. Pour cela, j'ai branché un câble USB/RS232 partant de l'ordinateur et j'ai court-circuité les pins d'envoi et de réception du câble. Ainsi, avec le logiciel PuTTY ouvert en mode série et connecté au port COM correspondant, il faut que lorsque je presse une touche du clavier, cette dernière apparaisse dans le terminal PuTTY sur l'ordinateur.

Une fois cela réussi, il me faut configurer la communication série en assembleur dans le programme.

Afin de configurer la communication série USART, il faut initialiser trois registres différents : TXSTA (Transmit Status and Constol Register), RCSTA (Receive Status and Control Register) et SPBRG (Baud Rate Generator Register).

Les registres TXSTA et RCSTA sont utilisés afin d'autoriser la transmission des données en USART. Cela en mode asynchrone, à haute vitesse et en contrôlant l'état du buffer d'échange des données (soit plein, soit vide).

```
banksel TXSTA             ;configuration Transmit Status And Control Register
movlw       B'00100110'   ;autorisation de la transmission TXEN, vitesse des bauds haute en asynchrone, bit TMRT à 1 pour vider le registre TSR
movwf       TXSTA         ;ajout de la configuration
banksel RCSTA             ;configuration de la reception des status et le controle des registres pour USART
movlw       B'10000000'   ;autorisation du bit du port série SPEN
movwf       RCSTA
```

Le registre SPBRG quant à lui, il est initialisé afin de configurer une certaine fréquence d'envoi des données en USART. On peut donc réaliser un calcul afin de connaître la valeur à initialiser pour SPBRG.

Baud Rate = FOSC/(16(X+1))

Or on a Baud Rate = 9600 bauds, qui est donné dans le sujet et une fréquence d'oscillation de 4MHz.

On a donc : X = (4000000/9600 - 16) /16 = 25.
Il faut donc configurer le nombre 25 dans les 16 bits du registre SPBRG afin de communiquer à 9600 bauds.

```
banksel SPBRG                       ;configuration de la fréquence d'envoi des données au port série
movlw       B'0000000000011001'     ;SPBRG = 25 pour une fréquence à 4MHz à 9600 bauds en asynchrone
movwf       SPBRG
```

Une fois la configuration de l'USART effectuée, il reste à envoyer un caractère sur le terminal.

Pour cela, l'instruction "movlw" doit être utilisée. Je mets le caractère souhaité dans le registre W. Je vérifie ensuite que le buffer USART possède bien une donnée. Une fois la donnée présente, j'envoie le caractère sur le port série en utilisant "movwf" avec le registre de transmission des données USART "TXREG" (voir prochain code).

Une fois la donnée envoyée, je vérifie que la communication a eu lieu grâce au bit TRMT du registre TXSTA. Ce bit étant accessible uniquement en lecture, il va s'initialiser à 1 lorsque la donnée a été envoyée avec TXREG et que le registre de transmission est donc vide.

Voici ci-dessous la partie de code qui vient d'être expliquée, permettant d'envoyer le caractère "a" dans le terminal, grâce à la liaison série.

```
envoie
    banksel PIR1                ;démarrage de la communication USART
    movlw 'a'
    verifmsg
    btfss       PIR1, TXIF      ;verification si la communication est libre. Le flag du bit TXIF est mis à 0 en chargeant TXREG
    goto        verifmsg
    banksel     TXREG
    movwf       TXREG           ;envoi du message dans le registre TXREG

    verif
        banksel     TXSTA
        btfsc       TXSTA, TRMT     ;verification si la communication a eu lieu
        goto        verif           ;on boucle tant que TRMT est à 1. TRMT = 1 --> TSR = vide. Quand TMRT = 0 alors communication effectuée.
        return
```

Maintenant que je sais envoyer un caractère dans le terminal, il faut être capable d'envoyer la tension lue en entrée de la carte en continue.

## Transcodage de la tension de binaire à ascii

Avant de pouvoir envoyer la tension lue en entrée en fonction du positionnement du positionneur sur le port série du pic, il faut être capable de lire cette tension. Pour réaliser cela, je fais appel au tutoriel, en réutilisant le registre ADRESH.

Je vais donc stocker ce nombre binaire de 8bits dans une variable appelée "valBinaire".

```
conversionVal
        banksel     ADRESH                  ;Selection du registre
        movfw       ADRESH                  ;Envoie dans le registre W la valeur de la conversion Ana/Num de 0 à 5V
        movwf       valBinaire
```

L'algorithme que je vais mettre en place va à chaque lecture, initialiser la partie entière et décimale de la tension lue à 0 pour avoir 0,0V.

```
movlw       D'0'                    ;On initialise l'entier et le décimal à 0 pour réécrire la nouvelle valeur lue
movwf       valDecDecimale
movwf       valDecEntiere
goto        conversionUnit
```

Puis, je vais mettre la valeur décimale 51 dans le registre W, que je vais traiter tout au long de l'algorithme afin d'obtenir la tension souhaitée entre 0 et 5V.

En effet, grâce à la valeur binaire renvoyée par le convertisseur analogique/numérique, je vais pouvoir effectuer une soustraction sur 51 afin d'obtenir une unité et non pas une dizaine comme chiffre entier.
On aura donc valBinaire – 51 (00110011 en binaire) qui donnera notre unité de tension qui sera stockée dans le registre W.

Afin d'obtenir cette unité, je vais tester si un overflow est présent lors de la soustraction. Tant que la carry est à 1, j'incrémente la valeur de l'unité à chaque appel de la fonction pour passer de 0,0V à x,0V.

```
conversionUnit
    movlw       D'51'                   ;51 est la valeur pour l'unité du V 0,1,2,3,4 ou 5
    subwf       valBinaire,0            ;Calcul de W (=51) - valBinaire. On stocke le résultat dans le registre W
    btfss       STATUS,C                ;Permet de vérifier si la carry d'overflow=1. Si C = 1 alors on skip l'instruction suivante, si C = 0 alors on continue normalement
                ;C se trouve dans le registre STATUS
    goto        conversionDec           ;Permet d'obtenir la valeur décimale
    incf        valDecEntiere
    movwf       valBinaire      ;
    goto        conversionUnit
```

Une fois l'unité récupérée et la carry à 0, il me faut récupérer le dixième de la tension lue avec ADRESH. Comme j'ai choisi de soustraire la décimal prérentrée par la valeur lue avec ADRESH, j'ai choisi 5 et non 51 pour les dixièmes de volts allants de 0 à 9. Le même processus est réalisé pour obtenir les unités en soustrayant la valBinaire obtenue, par 5.

Il faut incrémenter les décimales tant qu'elles ne sont pas supérieures à 9. Ici, il faut être capable de gérer le passage à l'unité suivante. En effet, dans la table ascii, l'incrémentation suivant le 9 vaut ":". Or dans ce cas, je souhaite avoir un 0 après le 9. C'est pour cela que j'effectue un test sous forme de soustraction à nouveau pour savoir si le 9 a été atteint ou non. Ainsi, si l'opération vaut zéro (9-9=0), je n'incrémente plus les décimales et je redonne la main à l'algorithme qui gère les unités pour passer à l'unité suivante si "valBinaire" a changé.

```
conversionDec
    movlw       D'5'                    ;5 est la valeur pour 0,1V. On soustrait 5 pour avoir les décimales
    subwf       valBinaire,0        ;Calcul de W (=5) - valBinaire. On stocke le résultat dans le registre W
    btfss       STATUS,C                ;Permet de vérifier si la carry d'overflow=1. Si C = 1 alors on skip l'instruction suivante, si C = 0 alors on continue normalement
    continue
    goto        conversionFin

    movwf       valBinaire
    movlw       D'9'
    subwf       valDecDecimale,0        ;voir si la partie decimal vaut 9 et lui soustraire 9 car en ascii le caractère après 9 est ":"
    btfss       STATUS,Z                ;skip if Z = 1 et donc que l'opération vaut 0
    incf        valDecDecimale
    goto        conversionUnit
```

Grâce à ces deux parties de code, je peux maintenant avoir une valeur décimale de la tension lue en entrée. Le soucis étant qu'une liaison série de va pas afficher une valeur décimale en lui envoyant une donnée de type décimale. En effet, en prenant la table ascii, si j'envoie par exemple "64" en décimal, le terminal PuTTY va afficher un "@"

| Decimal | Hex | Char |
|---|---|---|
| 64 | 40 | @ |

Il faut donc pour cela convertir la valeur décimal en ASCII. A savoir, qu'en ASCII, les chiffres de 0 à 9 sont codés de 0x30 à 0x39 en hexadécimal. Or, les résultats sauvegardés dans mes variables "valDecEntiere" et "valDecDecimale" sont toujours des valeurs décimales entre 0 et 9. Ainsi, en ajoutant des valeurs entre 0 et 9 à 0x30, j'aurais toujours une valeur ASCII entre 0 et 9.

C'est pour cela qu'une fois la conversion de l'unité réalisée, je fais une addition entre valDecEntiere et 0x30. Puis, je fais de même entre valDecDecimale et 0x30.

```
    conversionFin
        movfw       valDecEntiere
        addlw       0x30                        ;Addition de la valeur dans W de W + 0x30 pour l'avoir en mode ascii
return
```

Grâce à tout ce processus, j'arrive à convertir la tension binaire lue en entrée de la carte en ASCII et je peux l'afficher sur le terminal PuTTY en fonction de la position du potentiomètre.

Effectivement cette tension est maintenant envoyée vers le terminal, mais un problème persiste. Je ne sais pas forcément si toutes les informations sont envoyées vers le terminal ou si seulement une partie est correctement transférée.

# Transmettre en continue la tension au terminal du PC

Afin de transmettre au terminal PuTTY toutes les informations que je souhaite réellement afficher grâce au port série, j'ai mis au point un algorithme permettant d'ordonner les données que je souhaite afficher. De plus, cet algorithme va permettre d'afficher toutes ces informations en continue sur le terminal.

Pour cela, je démarre tout d'abord la communication avec le convertisseur analogique/numérique en initialisant le registre ADCON0. Tant que toutes les données de la tension en binaire ne sont pas récupérées, je continue de récupérer les données.

```
recupValToPrint                         ;récupère la valeur de la tension et l'affiche sur le port série du PC
    banksel ADCON0                      ;On mesure la tension entre 0 et 5V
    bsf         ADCON0,GO               ;Go est mis à 1 dans le registre ADCON0 pour faire le convertisseur analogique numérique
    affichemesure
        btfsc       ADCON0,GO
        goto        affichemesure                   ;on boucle sur affichemesure tant qu'on a pas toutes les données de la tension
```

Une fois la tension récupérée, je convertis d'abord l'unité (grâce à la fonction "conversionVal" expliquée précédemment) que j'envoie ensuite sur le terminal (grâce à la fonction "envoie" expliquée précédemment).

```
call        conversionVal               ;conversion de binaire à ascii dans le cas où ADCON0 = 0
call        envoie                      ;affiche l'unité convertie en ascii
```

Une fois l'unité envoyée sur le terminal PuTTY grâce à l'USART, j'envoie le caractère "," pour séparer l'unité du dixième. Pour cela, j'utilise la même façon de faire que pour la deuxième partie où il fallait envoyer un simple caractère sur le terminal.

```
movlw       ','
call        envoie                      ;affiche la virgule
```

Il reste ensuite à convertir la partie décimale en ASCII en utilisant une nouvelle fois la fonction "conversionVal" puis à l'envoyer sur le terminal via le port série.

```
call        conversionVal
movfw       valDecDecimale
call        envoie                      ;affiche la décimale convertie en ascii
```

Une fois la valeur de la tension affichée, il reste plus qu'à envoyer un "V" pour l'unité de mesure et à s'assurer du saut de ligne.

```
movlw       'V'
call        envoie                      ;affiche l'unité Volt
movlw       '\n'                        ;retour à la ligne
call        envoie
movlw       '\r'                        ;mise du curseur en début de ligne
call        envoie

return
```

J'ai donc ajouté cette fonction au programme principal qui va appeler en continue cette partie du programme. Ainsi, il sera possible d'afficher la valeur de la tension lue en entrée, en continu sur un terminal à travers le port série du PIC.

```
main

    call recupValToPrint
```

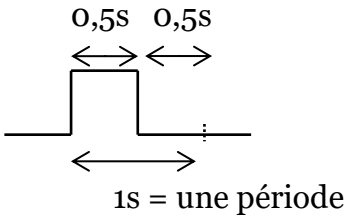
Le soucis reste que ces informations de tensions sont envoyées à très grande vitesse et qu'il serait intéressant de pouvoir les cadencer afin de réguler l'affichage de la tension sur le terminal.

# Cadencement de l'affichage avec un timer

Afin de pouvoir contrôler la vitesse d'affichage de la tension lue, il est nécessaire de mettre en place un timer. Le timer créé ici va permettre d'affichage la valeur de la tension une fois par seconde.

Afin d'utiliser un timer, il faut tout d'abord le configurer et l'initialiser dans notre code. Pour cela il faut trouver quel timer pourrait être utilisé pour compter une seconde entre chaque affichage.

En effet, le PIC16F877 possède deux timers qui comptent sur 8 bits et un timer qui compte sur 16 bits. On sait qu'un timer est une horloge avec comme période un front montant et un front descendant. Dans le cas d'un cadencement à une seconde, il faut non pas initialiser le compteur à une seconde, mais à 500ms. Ainsi, le front montant et le front descendant généreront la seconde de cadencement.




1s = une période

Comme le prédiviseur le plus grand utilisable avec le PIC vaut 8, il n'est pas possible d'utiliser un timer qui compte sur 8bits. En effet, 500000/8 = 62500. On doit donc incrémenter le compteur 62500 fois afin de compter 500000µs.

Un timer sur 8 bits peut s'incrémenter 256 fois alors qu'un timer sur 16 bits peut s'incrémenter 65536 fois au maximum.

On sait maintenant qu'il faudra incrémenter 62500 fois le compteur et que seul le timer1 sur 16bits permettra de réaliser cela.

Il faut donc configurer le registre T1CON en mettant les bits T1CKPS1 et T1CKPS0 à 1 afin de sélectionner le prédiviseur 8.

```
;configuration du timer 1 pour compter 1sec
banksel T1CON
movlw   B'00110000'         ;selection du prescaler 1:8 pour pouvoir compter assez longtemps pour 1sec
movwf T1CON
```

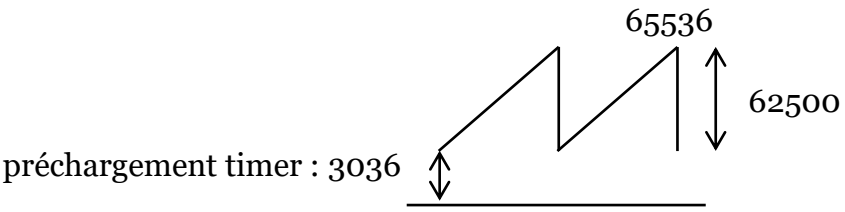
Une fois le programme lancé et la première valeur de tension affichée, il va falloir appeler le compteur. Pour cela, j'ai créé une variable "compteur" que j'initialise à "2" pour réaliser deux fois la boucle d'incrémentation du compteur (2x500000µs = 1sec).

```
main
    envoyer
        call recupValToPrint

        ;utilisation du timer pour cadencer l'affichage à 1sec.
        movlw   .2                  ;comme j'initialise le compteur à 500ms, il faut compter deux fois pour avoir 1sec
        movwf   compteur
        call    timer
```

Tout d'abord, il faut vérifier que le timer ne compte pas lors de son initialisation. Pour cela, il suffit de mettre le timer1 à l'arrêt en mettant le bit TMR1ON à 0 dans le registre T1CON, ainsi que de supprimer le flag du registre PIR1. Cela s'effectue en mettant le bit de TMR1IF à 0.

```
timer
    banksel T1CON
    bcf     T1CON, TMR1ON   ; met en arrêt le timer 1

    ;initialisation à 3036 du timer1
    banksel     PIR1
    bcf     PIR1, TMR1IF    ;supprime le flag du timer1
```

Il reste maintenant à initialiser le compteur pour qu'il compte 500µs et donc incrémente exactement 62500 fois. On sait aussi que le timer pourrait compter 65536 fois si on ne lui donne aucune valeur de démarrage.
65536 – 62500 = 3036.
Il faut donc faire en sorte de précharger le timer à une valeur de 3036, afin qu'un tour de boucle puisse faire compter le compteur 500µs.




Pour réaliser cela, il est possible d'initialiser les deux octets TMR1H et TMR1L dans le registre PIR1. Or 3036 se code sous 16 bits de la façon suivante :

| TMR1H | TMR1L |
|---|---|
| 00001011 | 11011100 |

```
;préchargement du timer à 3036
movlw   B'11011100'
movwf   TMR1L
movlw   B'00001011'
movwf   TMR1H
```

Une fois le pré chargement du timer effectué pour compter 500µs, il suffit de démarrer le timer1 en mettant le bit TMR1ON à 1 dans le registre T1CON.

```
banksel T1CON
bsf     T1CON, TMR1ON       ;démarrage le timer1
```

Une fois le timer démarré, il faut vérifier que la boucle de comptage s'effectue bien deux fois. Pour cela, dès que le timer déclenche un flag pour annoncer qu'il est au bout de son comptage, il faut décrémenter la valeur du compteur (initialement 2), puis rappeler le timer tant que "compteur" ne vaut pas 0. Si la variable "compteur" a été décrémentée deux fois, alors le cadencement d'une seconde a été effectué et on peut redemander la valeur de la tension actuelle en rappelant la fonction "envoyer".

```
gestiontimermode
    call lectureMode
    btfss statbtn, 0                ;si statbtn = 0 alors on repasse en manuel
    goto entreemanu

    banksel PIR1
    btfss PIR1, TMR1IF              ;vérification du flag d'overflow si le timer a fini d'incrémenter 62500 fois.
    goto gestiontimermode           ;si le timer n'a pas finit, il boucle dans cet état.
    decfsz compteur, F              ;si il a fini, je décrémente compteur pour réaliser un deuxième décompte de 500µs
    goto attfintimer                ;va rappeler le timer pour compter une deuxième fois
    goto envoyer                    ;retourne à l'état initial d'affichage de la tension sur le terminal
    attfintimer
    call timer
    goto gestiontimermode
```

Le timer réalisé dans cette partie permet d'afficher sur le terminal, la valeur de la tension lue une fois par seconde. Il reste maintenant à choisir si l'on souhaite que l'affichage sur le terminal se fasse automatiquement ou à la demande de l'utilisateur.

# Envoi des caractères du clavier au terminal

Afin de pouvoir afficher les tensions sur le terminal en fonction de la demande de l'utilisateur, il faut pouvoir lire les entrées au clavier de l'utilisateur. Pour cela, il faut utiliser le registre RCREG et lire l'état du bit RCIF du registre PIR1. En effet, RCREG est le registre de réception des données envoyées par USART. Si l'utilisateur rentre un caractère au clavier, ce dernier sera stocké dans le registre de 8 bits RCREG.

Quant au bit RCIF, il permet de connaître l'état de RCREG. Effectivement, lorsque le buffer de RCREG est lu et vidé, RCIF = 0. Sinon, lorsqu'un caractère a été rentré au clavier, ce buffer est plein et RCIF = 1.

Ainsi, pour envoyer un caractère sur le terminal, il faut tout d'abord vérifier que le bit RCIF soit bien à 0 et boucler sur la lecture de ce bit tant qu'il ne vaut pas 1.

```
lectureMode
    banksel PIR1
    btfss PIR1, RCIF          ;si le buffer qui recoit les informations USART (RCREG) est plein, alors je passe la prochaine instruction et je traite la demande
    return                    ;cas où aucun caractère n'a été rentré. On boucle jusqu'à obtenir un caractère
```

Une fois le caractère détecté, il suffit de charger le registre W avec caractère entré au clavier, qui présent dans le buffer RCREG. À la suite de cela, l'appel à la fonction "envoie" qui permet d'envoyer un caractère sur le port série va envoyer correctement le caractère rentré par l'utilisateur sur le terminal.

```
banksel RCREG               ;RCIF reset si RCREG est lu et vide
movfw RCREG                 ;Envoie dans le registre W le carctère entré au clavier
call envoie
```

L'envoi d'un caractère depuis une entrée sur le clavier est maintenant possible. Il reste encore à détecter si la touche entrée correspond à une touche qui permettrait de sélectionner le mode "automatique" ou le mode "à la demande".

# Gestion du mode manuel ou automatique

Enfin, comme il a été demandé dans le cahier des charges, l'objectif de ce projet est de pouvoir afficher la tension en entrée de la carte PICDEM2 soit automatiquement, soit à la demande de l'utilisateur.

Pour cela, il faut faire en sorte que lorsque l'utilisateur rentre le caractère "a" au clavier, l'affichage de la tension s'affiche automatiquement avec le cadencement d'une seconde réalisé auparavant. Une autre situation est possible, c'est celle du mode "à la demande". Ce mode doit être appelé lorsque l'utilisateur appui sur la touche "r" de son clavier.

Ainsi, une fois dans le mode "à la demande", à chaque appui de l'utilisateur sur la douche "d", la tension en entrée de la carte s'affichera sur le terminal. Et ce, jusqu'au prochain appui de l'utilisateur sur le caractère "a" pour repasser en mode "automatique".

J'ai donc décidé d'initialiser le mode "automatique" au lancement du programme. Les valeurs de la tension s'afficheront automatiquement toutes les secondes sur le terminal. On peut voir ci-dessous que la sélection du mode se fera en fonction de la valeur de la variable "statbtn" que j'ai initialisée au début du code.

```
bsf statbtn, 0              ;initialisation du mode automatique
```

Lorsque "statbtn" = 1, le mode est automatique. Si "statbtn" vaut 0, alors on est dans le mode "à la demande".

La partie la plus compliquée ici était de réarranger tout le code déjà effectué afin d'intégrer les différents modes d'utilisation. De plus, il fallait pouvoir détecter si le caractère entré correspondait à la sélection d'un mode ou non.

Dans la partie principale du programme, je suis donc parti du principe que le mode sélectionné par défaut était le mode "automatique", tout en testant qu'il n'y ait pas eu de modification de l'utilisateur entre temps.

```
main

    btfss statbtn, 0                    ;je test si le mode manuel a été activé ou non
    goto entreemanu
```

Puis, j'ai dû réaliser une séquence expliquant ce que le mode "automatique" devait afficher sur le terminal. Dans le cas où aucun caractère n'est rentré au clavier, l'affichage de la valeur sur le terminal se fait normalement, comme j'ai pu l'expliquer jusqu'à présent.

```
entreeauto
    envoyer
        call selectionModeA         ;affichage sur le terminal de "A : " pour dire que le programme est en automatique
        call recupValToPrint        ;affichage de la valeur de la tension

        ;utilisation du timer pour cadencer l'affichage à 1sec.
        movlw   .2                      ;comme j'initialise le compteur à 500ms, il faut compter deux fois pour avoir 1sec
        movwf   compteur
        call    timer
```

L'unique différence est que le programme est en mode "automatique" et qu'il faut pouvoir justifier cela à l'utilisateur. J'ai donc réalisé une fonction "selectionModeA" qui va afficher "A : " sur le terminal avant d'afficher la valeur de la tension lue.

```
selectionModeA                          ;Affichage du mode automatique sur le terminal
    movlw 'A'                           ;permet de connaitre le mode de fonctionnement actuel sur le terminal
    call envoie                         ;Envoi du caractère A sur le terminal
    movlw ':'
    call envoie
    movlw ' '
    call envoie
    call recupValToPrint                ;affiche la valeur de la tension lue
    return
```

Il est maintenant possible de voir au terminal que le mode d'affichage de la tension est automatique, mais la gestion de l'entrée des caractères par l'utilisateur n'est pas encore effectuée. Pour cela, j'ai créé une fonction appelée "lectureMode". Je vais premièrement l'appeler lorsque je suis dans le mode "automatique".

```
gestiontimermode
    call lectureMode
    btfss statbtn, 0                ;si statbtn = 0 alors on repasse en manuel
    goto entreemanu
```

Cette fonction va comporter la réception d'un caractère comme cela a été expliqué dans la partie précédente "Envoi des caractères du clavier au terminal" avec l'utilisation de RCREG pour détecter si un caractère est envoyé par le clavier ou non. La modification qui est effectuée ici, est la comparaison de ce caractère envoyé avec l'un des caractères permettant la sélection d'un mode.

La comparaison commence avec le mode automatique. L'algorithme utilisé ici va utiliser la soustraction entre la valeur binaire de RCREG entrée au clavier et 'a'. Ainsi, si ces deux octets sont égaux, le résultat de cette soustraction vaut 0.

```
lectureauto
    movlw 'a'
    subwf RCREG, 0          ;soustraire la valeur entrée par a
    btfss STATUS, Z         ;skip si RCREG contient "a" et donc que l'opération = 0. Sinon on va tester le mode "à la demande"
```

Grâce à l'instruction "BTFSS STATUS, Z", on détecte que le caractère entré est 'a' si le résultat de l'opération vaut 0. Dans ce cas, on réinitialise la variable de sélection du mode (statbtn) en "automatique" et on continue d'envoyer la valeur de la tension une fois par seconde.

Si le résultat de cette soustraction est différent de 0, alors Z vaut 1 et il faut effectuer l'instruction suivante. L'instruction suivante va appeler le mode "à la demande" afin de regarder si le caractère entré est le 'r'.

```
    btfss STATUS, Z         ;skip si RCREG contient "a" et donc que l'opération = 0. Sinon on va tester le mode "à la demande"
    goto lecturedemande
    bsf statbtn, 0          ;mise en mode "automatique" en mettant le bit 0 à 1
return

lecturedemande
    movlw 'r'
    subwf RCREG, 0          ;soustraire la valeur entrée par r
    btfss STATUS, Z         ;skip si RCREG contient "r" et donc que l'opération = 0. Sinon on regarde si l'utilisateur a rentré un caractère
    go to demandeenvoyee    ;attente du caractère "d" entrée par l'utilisateur pour afficher la tension
    bcf statbtn, 0          ;mise en mode "à la demande" en mettant le bit 0 à 0
return
```

Ainsi, on peut voir que j'effectue également une soustraction entre le caractère entré et le caractère 'r' afin de savoir si le mode "à la demande" a été demandé par l'utilisateur ou non. Tant que l'utilisateur n'entre ni le caractère 'a', ni le caractère 'r', le programme va boucler entre les deux fonctions "lectureauto" et "lecturedemande" à chaque fois que l'utilisateur va entrer un caractère.

Une fois que l'utilisateur aura rentré le caractère 'r' et que la soustraction entre RCREG et 'r' vaudra 0, on remarque que "statbtn" est initialisé dans le mode "à la demande" et que le programme rentre dans un nouvel état.

En effet, une fois dans le mode "à la demande" sélectionné, il faut savoir quand est-ce que l'utilisateur souhaite afficher la tension à l'écran. Pour cela il faut tout d'abord vérifier qu'à chaque fois que l'on attend la valeur entrée par l'utilisateur avec le clavier, on est toujours en mode "à la demande".

```
demandeenvoyee
    btfsc statbtn, 0        ;test si le mode est devenu "automatique". Si toujours "à la demande" alors on skip le return
    return
```

Une fois ce test effectué, il suffit de récidiver la détection du caractère entré par l'utilisateur comme pour les deux fois précédentes.

```
movlw 'd'
subwf RCREG,0           ;soustraire la valeur entrée par d
btfss STATUS, Z         ;skip si RCREG contient "d" et donc que l'opération = 0. Sinon on va tester le mode "à la demande"
return          ;retourne dans le mode automatique si soustraction != 0
call selectionModeM
```

Ainsi, si le caractère entré n'est pas 'd', on retourne dans le processus du mode "automatique".

Dans le cas où le caractère 'd' est renseigné, alors on appelle la fonction "selectionModeM".

```
selectionModeM                  ;Affichage du mode manuel sur le terminal
    movlw 'M'                   ;permet de connaitre le mode de fonctionnement actuel sur le terminal
    call envoie                 ;Envoi du caractère M sur le terminal
    movlw ':'
    call envoie                 ;Envoi du caractère ":" sur le terminal
    movlw ' '
    call envoie                 ;Envoi du caractère " " sur le terminal
    call recupValToPrint        ;affiche la valeur de la tension lue
    return
```

Cette fonction va pouvoir justifier sur le terminal que la tension est affichée dans le mode manuel en écrivant "M : " avant d'afficher la tension lue.

Ainsi tant que la valeur de "statbtn" vaudra 0, la partie principale du programme voudra afficher une valeur à la demande de l'utilisateur. Et ce jusqu'à ce que l'utilisateur entre à nouveau un 'a' sur le clavier.

```
    entreemanu
        call lectureMode
        btfss statbtn, 0
        goto entreemanu             ;reste dans le mode manuel tant que la variable statbtn est en mode manu '0'
        goto entreeauto


end
```

# Conclusion

L'objectif de ce projet qui était de réaliser un système d'acquisition série pour PC a été réussi. En effet, on peut remarquer que petit à petit, en assemblant des parties de code, on peut réaliser un projet intéressant en assembleur. Il est donc préférable pour des projets de ce type de correctement séparer les algorithmes à réaliser, pour finalement atteindre l'objectif final.

Grâce à cela, j'ai pu mettre en place une communication série USART avec le microcontrôleur, afin d'afficher sur le terminal PuTTY de l'ordinateur la valeur de la tension lue en entrée de la carte.

Puis, j'ai également découvert comment convertir la valeur de la tension initialement donnée sur 8bits, en ascii. Cela, en passant par le domaine décimal.

Ensuite, la volonté de cadencer l'affichage de la tension sur le terminal permet de prendre en main et de comprendre comment initialiser et compter avec les timer. En effet, grâce aux timer, on peut réaliser des temporisations sur les microcontrôleurs.

Enfin, ce projet m'a également permis de découvrir comment lire et comparer des données envoyées par l'utilisateur avec le clavier. C'est grâce à cela que j'ai pu mettre en place une sélection de mode d'affichage de la tension sur le terminal, à la bonne volonté de l'utilisateur.

# Annexe

```
;Réalisation d'un système d'acquisiton série pour PC
;Programme réalisé par Léo Jellimann
;FIP 2A promotion 2023


#include "p16f877.inc"

__CONFIG _FOSC_XT & _WDTE_OFF & _PWRTE_OFF & _CP_OFF & _BOREN_OFF & _LVP_OFF & _CPD_OFF & _WRT_ON

ORG 0x000

    CBLOCK 0x20                         ;Initialisation des variables
            valBinaire : 1
            valDecEntiere : 1
            valDecDecimale : 1
            compteur : 1
            statbtn : 1
    ENDC


    banksel TXSTA               ;configuration Transmit Status And Control Register
    movlw         B'00100110'   ;autorisation de la transmission TXEN, vitesse des bauds haute en asynchrone, bit TMRT à 1 pour vider le registre TSR
    movwf         TXSTA         ;ajout de la configuration

    banksel SPBRG                           ;configuration de la fréquence d'envoi des données au port série
    movlw         B'0000000000011001'       ;SPBRG = 25 pour une fréquence à 4MHz à 9600 bauds en asynchrone
    movwf         SPBRG

    banksel RCSTA               ;configuration de la reception des status et le controle des registres pour USART
    movlw         B'10000000'   ;autorisation du bit du port série SPEN
    movwf         RCSTA

    ;configuration du convertisseur analogique/numérique
    banksel ADCON0
    movlw         B'01000001'           ;Utilisation de Fosc/8 et autorisation de l'utilisation du convertisseur analogique/numérique
    movwf         ADCON0

    banksel ADCON1
    movlw         B'00001110'           ;Configuration du convertisseur pour avoir 7 ports numériques et les tensions de référence Vref+ = VDD et Vref- = VSS
    movwf         ADCON1


    ;configuration du timer 1 pour compter 1sec
    banksel T1CON
    movlw   B'00110000'         ;selection du prescaler 1:8 pour pouvoir compter assez longtemps pour 1sec
    movwf T1CON

    banksel PIR1
    bcf     PIR1, TMR1IF


    bsf statbtn, 0              ;initialisation du mode automatique
    call envoie                 ;on veut démarrer la communication USART
    goto          main


;fonctions

envoie
    banksel PIR1                ;démarrage de la communication USART
    verifmsg
    btfss         PIR1, TXIF    ;verification si la communication est libre. Le flag du bit TXIF est mis à 0 en chargeant TXREG
    goto          verifmsg

    banksel      TXREG
    movwf        TXREG          ;envoi du message dans le registre TXREG

    verif
        banksel      TXSTA
        btfsc        TXSTA, TRMT    ;verification si la communication a eu lieu
        goto         verif          ;on boucle tant que TRMT est à 1. TRMT = 1 --> TSR = vide. Quand TMRT = 0 alors communication effectuée.
        return


conversionVal

        banksel       ADRESH                    ;Selection du registre
        movfw         ADRESH                    ;Envoie dans le registre W la valeur de la conversion Ana/Num de 0 à 5V
        movwf         valBinaire
        movlw         D'0'                      ;On initialise l'entier et le décimal à 0 pour réécrire la nouvelle valeur lue
        movwf         valDecDecimale
        movwf         valDecEntiere
        goto          conversionUnit

    conversionUnit
        movlw        D'51'                      ;51 est la valeur pour l'unité du V 0,1,2,3,4 ou 5
        subwf        valBinaire,0               ;Calcul de W (=51) - valBinaire. On stocke le résultat dans le registre W
        btfss        STATUS,C                   ;Permet de vérifier si la carry d'overflow=1. Si C = 1 alors on skip l'instruction suivante, si C = 0 alors on continue normalement
                        ;C se trouve dans le registre STATUS
        goto         conversionDec              ;Permet d'obtenir la valeur décimale
        incf         valDecEntiere
        movwf        valBinaire      ;
        goto         conversionUnit

    conversionDec
        movlw        D'5'                        ;5 est la valeur pour 0,1V. On soustrait 5 pour avoir les décimales
        subwf        valBinaire,0           ;Calcul de W (=5) - valBinaire. On stocke le résultat dans le registre W
        btfss        STATUS,C               ;Permet de vérifier si la carry d'overflow=1. Si C = 1 alors on skip l'instruction suivante, si C = 0 alors on continue normalement
        continue
        goto         conversionFin

        movwf        valBinaire
        movlw        D'9'
        subwf        valDecDecimale,0               ;voir si la partie decimal vaut 9 et lui soustraire 9 car en ascii le caractère après 9 est ":"
        btfss        STATUS,Z                       ;skip si Z = 1 et donc que l'opération vaut 0
        incf         valDecDecimale
        goto         conversionUnit

    conversionFin
        movfw        valDecEntiere
        addlw        0x30                       ;Addition de la valeur dans W de W + 0x30 pour l'avoir en mode ascii
return

timer
    banksel T1CON
    bcf     T1CON, TMR1ON   ; met en arrêt le timer 1

    ;initialisation à 3036 du timer1
    banksel     PIR1
    bcf     PIR1, TMR1IF    ;supprime le flag du timer1
```

```
121
122     ;préchargement du timer à 3036
123     movlw   B'11011100'
124     movwf   TMR1L
125     movlw   B'00001011'
126     movwf   TMR1H
127
128     banksel T1CON
129     bsf     T1CON, TMR1ON       ;démarrage le timer1
130
131
132 recupValToPrint                            ;récupère la valeur de la tension et l'affiche sur le port série du PC
133     banksel ADCON0                          ;On mesure la tension entre 0 et 5V
134     bsf         ADCON0,GO                   ;Go est mis à 1 dans le registre ADCON0 pour faire le convertisseur analogique numérique
135     affichemesure
136         btfsc       ADCON0,GO
137         goto        affichemesure                   ;on boucle sur affichemesure tant qu'on a pas toutes les données de la tension
138         call        conversionVal               ;conversion de binaire à ascii dans le cas où ADCON0 = 0
139         call        envoie                      ;affiche l'unité convertie en ascii
140         movlw       ','
141         call        envoie                      ;affiche la virgule
142         call        conversionVal
143         movfw       valDecDecimale
144         call        envoie                      ;affiche la décimale convertie en ascii
145         movlw       'V'
146         call        envoie                      ;affiche l'unité Volt
147         movlw       '\n'                        ;retour à la ligne
148         call        envoie
149         movlw       '\r'                        ;mise du curseur en début de ligne
150         call        envoie
151
152         return
153
154 selectionModeA                         ;Affichage du mode automatique sur le terminal
155     movlw 'A'                           ;permet de connaitre le mode de fonctionnement actuel sur le terminal
156     call envoie                         ;Envoi du caractère A sur le terminal
157     movlw ':'
158     call envoie                         ;Envoi du caractère ":" sur le terminal
159     movlw ' '
160     call envoie                         ;Envoi du caractère " " sur le terminal
161     call recupValToPrint                ;affiche la valeur de la tension lue
162     return
163
164 selectionModeM                         ;Affichage du mode manuel sur le terminal
165     movlw 'M'                           ;permet de connaitre le mode de fonctionnement actuel sur le terminal
166     call envoie                         ;Envoi du caractère M sur le terminal
167     movlw ':'
168     call envoie                         ;Envoi du caractère ":" sur le terminal
169     movlw ' '
170     call envoie                         ;Envoi du caractère " " sur le terminal
171     call recupValToPrint                ;affiche la valeur de la tension lue
172     return
173
174 lectureMode
175     banksel PIR1
176     btfss PIR1, RCIF            ;si le buffer qui recoit les informations USART (RCREG) est plein, alors je passe la prochaine instruction et je traite la demande
177     return                      ;cas où aucun caractère n'a été rentré. On boucle jusqu'à obtenir un caractère
178
179     banksel RCREG               ;RCIF reset si RCREG est lu et vide
180
181     lectureauto
182         movlw 'a'
183         subwf RCREG, 0          ;soustraire la valeur entrée par a
184         btfss STATUS, Z         ;skip si RCREG contient "a" et donc que l'opération = 0. Sinon on va tester le mode "à la demande"
185         goto lecturedemande
186         bsf statbtn, 0          ;mise en mode "automatique" en mettant le bit 0 à 1
187     return
188
189     lecturedemande
190         movlw 'r'
191         subwf RCREG, 0          ;soustraire la valeur entrée par r
192         btfss STATUS, Z         ;skip si RCREG contient "r" et donc que l'opération = 0. Sinon on regarde si l'utilisateur a rentré un caractère
193         go to demandeenvoyee    ;attente du caractère "d" entrée par l'utilisateur pour afficher la tension
194         bcf statbtn, 0          ;mise en mode "à la demande" en mettant le bit 0 à 0
195     return
196
197     demandeenvoyee
198         btfsc statbtn, 0        ;test si le mode est devenu "automatique". Si toujours "à la demande" alors on skip le return
199         return
200
201         movlw 'd'
202         subwf RCREG,0           ;soustraire la valeur entrée par d
203         btfss STATUS, Z         ;skip si RCREG contient "d" et donc que l'opération = 0. Sinon on va tester le mode "à la demande"
204         return          ;retourne dans le mode automatique si soustraction != 0
205         call selectionModeM
206
207 main
208
209     btfss statbtn, 0                    ;je test si le mode manuel a été activé ou non
210     goto entreemanu
211
212     entreeauto
213         envoyer
214             call selectionModeA         ;affichage sur le terminal de "A : " pour dire que le programme est en automatique
215             call recupValToPrint        ;affichage de la valeur de la tension
216
217             ;utilisation du timer pour cadencer l'affichage à 1sec.
218             movlw   .2                  ;comme j'initialise le compteur à 500ms, il faut compter deux fois pour avoir 1sec
219             movwf   compteur
220             call    timer
221
222         gestiontimermode
223             call lectureMode
224             btfss statbtn, 0            ;si statbtn = 0 alors on repasse en manuel
225             goto entreemanu
226
227             banksel PIR1
228             btfss PIR1, TMR1IF          ;vérification du flag d'overflow si le timer a fini d'incrémenter 62500 fois.
229             goto gestiontimermode       ;si le timer n'a pas finit, il boucle dans cet état.
230             decfsz compteur, F          ;si il a fini, je décrémente compteur pour réaliser un deuxième décompte de 500µs
231             goto attfintimer            ;va rappeler le timer pour compter une deuxième fois
232             goto envoyer                ;retourne à l'état initial d'affichage de la tension sur le terminal
233             attfintimer
234             call timer
235             goto gestiontimermode
236
237     entreemanu
238         call lectureMode
239         btfss statbtn, 0
240         goto entreemanu                 ;reste dans le mode manuel tant que la variable statbtn est en mode manu '0'
241         goto entreeauto
242
243 end
```