

UE A211 : SYSTÈMES EMBARQUÉS I

Professeur : COSTA EMILE

---

**RAPPORT DE LABORATOIRE**

---

Etudiants : FINYA ALBAN  
KORKUT CANER

TP3 - Mise en oeuvre de la communication  
série (UART)

# Table des matières

<b>1</b>	<b>Objectif du TP</b>	<b>2</b>
<b>2</b>	<b>Introduction</b>	<b>3</b>
2.1	Notions prérequis . . . . .	3
2.2	Démarche suivie . . . . .	4
<b>3</b>	<b>Schéma de principe</b>	<b>6</b>
<b>4</b>	<b>Algorigramme</b>	<b>7</b>
<b>5</b>	<b>Code source</b>	<b>8</b>
<b>6</b>	<b>Analyse du code source</b>	<b>12</b>
<b>7</b>	<b>Conclusion</b>	<b>14</b>
<b>A</b>	<b>ANNEXE - Datasheet PIC18F45K22</b>	<b>15</b>
<b>B</b>	<b>Programme Python</b>	<b>16</b>
<b>C</b>	<b>Bibliographie</b>	<b>17</b>

# 1 Objectif du TP

Dans cette troisième séance de TP nous avons introduit la transmission/réception série appelée UART. Pour ce faire, il nous a été demandé de réaliser un mini-projet consistant à afficher des valeurs sur un écran LCD via le microcontrôleur *PIC18F45K22* et un Terminal RS232. Comme le montre la figure 1. La première ligne de l'écran LCD devra contenir les valeurs XXX (commandé par les boutons 1 et 2), YYY (commandé par les boutons 2 et 3), ZZZ (commandé par le potentiomètre) et OK ou NOK, en fonction du statut de la connexion entre l' $UART_{\mu C}$  et l' $UART_{PC}$ .

La seconde ligne quant à elle devra afficher des valeurs envoyées sous la forme [999;999] depuis un Terminal au  $\mu C$ .

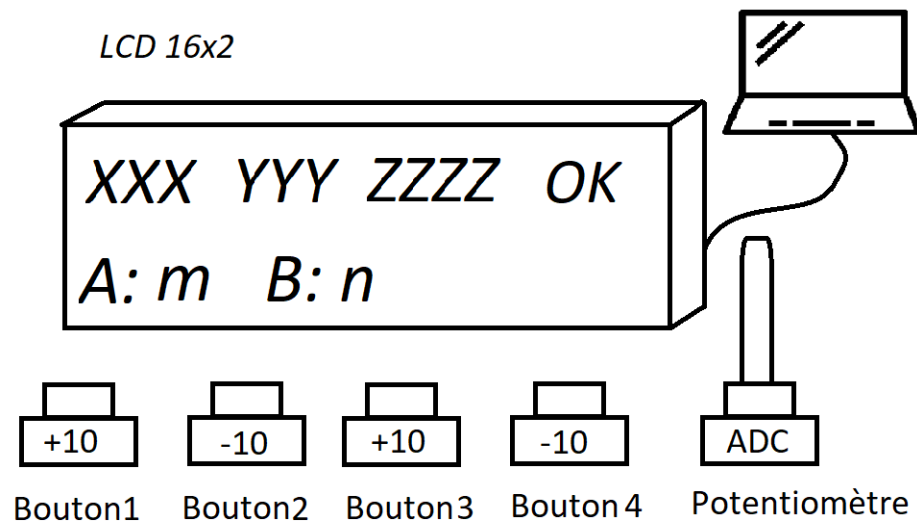


FIGURE 1 – Croquis du travail à réaliser

## 2 Introduction

Avant d'introduire le travail à réaliser, il nous semblait indispensable d'aborder quelques notions préliminaires afin de comprendre le mécanisme de la communication.

### 2.1 Notions prérequis

L'UART est un mode de communication asynchrone caractérisé par les éléments suivants :

- Transmission de données d'un équipement à un autre en P2P ;
- Stockage des données dans un buffer (mémoire tampon) d'une capacité de 64 octets ;
- Transmission asynchrone. La notion de *baudrate*<sup>1</sup> permet dès lors la synchronisation des données ;
- Niveaux de tension logiques de type TTL (0V à 5V).

Les données à transmettre existent sous forme parallèle (octet) et sont transmises sous forme série (LSB en premier). L'opération inverse est réalisée à la réception. La constitution des trames est la suivante :

- 1 bit de départ ;
- 7 à 8 bits de données ;
- 1 bit de parité optionnel ;
- 1 ou 2 bits d'arrêt.

A la sortie d'un UART, les bits sont généralement représentés par des tensions de niveau logique. Ces bits peuvent devenir *RS-232*, *RS-422*, *RS-485*, ou peut-être d'autres spécifications définies par les propriétaires du matériel considéré. Dans le cadre du laboratoire, nous avons utilisé la norme standard RS232 (niveaux logiques  $\pm 15V$ ). Nous n'entrerons pas dans les détails de ce standard. Notons simplement qu'il est présent sur la plupart des ordinateurs depuis les années 80 et qu'il est communément appelé *port série*.

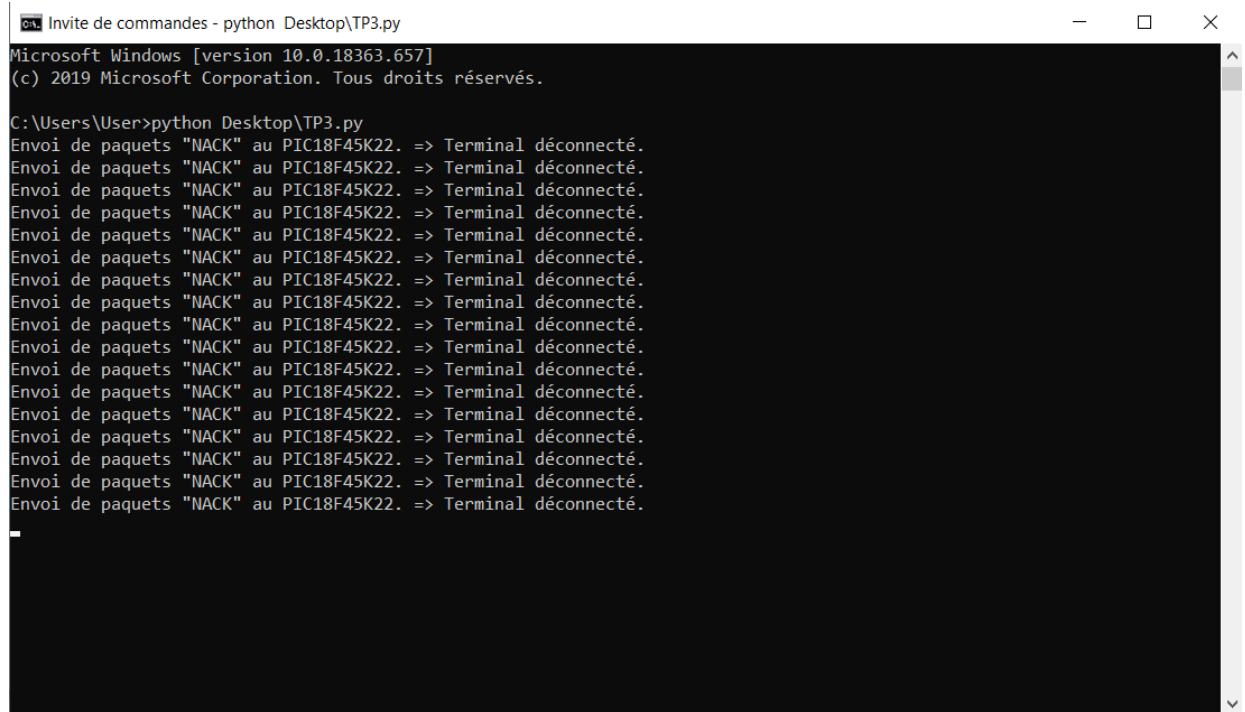
---

1. Nombre de *symboles* par seconde.

## 2.2 Démarche suivie

Afin de pouvoir faire communiquer la simulation Proteus avec un Terminal RS232 tournant sur une même machine, il nous a fallu créer 2 ports série virtuels et les relier entre eux. N'ayant pas réussi à utiliser le *polling* ou les *interrupt* pour vérifier l'état de la connexion entre le PC et le  $\mu C$ , nous avons rédigé un script **Python** afin de vérifier l'état de la communication. Le principe de ce programme est le suivant :

*Les ports série ne peuvent pas être exploités par deux processus différents. Dans ce cas, si notre programme Python arrive à se connecter au  $\mu C$ , nous pouvons déduire que les 2 processus ne sont pas connectés.*



```
Microsoft Windows [version 10.0.18363.657]
(c) 2019 Microsoft Corporation. Tous droits réservés.

C:\Users\User>python Desktop\TP3.py
Envoi de paquets "NACK" au PIC18F45K22. => Terminal déconnecté.
Envoi de paquets "NACK" au PIC18F45K22. => Terminal déconnecté.
Envoi de paquets "NACK" au PIC18F45K22. => Terminal déconnecté.
Envoi de paquets "NACK" au PIC18F45K22. => Terminal déconnecté.
Envoi de paquets "NACK" au PIC18F45K22. => Terminal déconnecté.
Envoi de paquets "NACK" au PIC18F45K22. => Terminal déconnecté.
Envoi de paquets "NACK" au PIC18F45K22. => Terminal déconnecté.
Envoi de paquets "NACK" au PIC18F45K22. => Terminal déconnecté.
Envoi de paquets "NACK" au PIC18F45K22. => Terminal déconnecté.
Envoi de paquets "NACK" au PIC18F45K22. => Terminal déconnecté.
Envoi de paquets "NACK" au PIC18F45K22. => Terminal déconnecté.
Envoi de paquets "NACK" au PIC18F45K22. => Terminal déconnecté.
Envoi de paquets "NACK" au PIC18F45K22. => Terminal déconnecté.
Envoi de paquets "NACK" au PIC18F45K22. => Terminal déconnecté.
Envoi de paquets "NACK" au PIC18F45K22. => Terminal déconnecté.
Envoi de paquets "NACK" au PIC18F45K22. => Terminal déconnecté.
```

FIGURE 2 – Programme Python servant à interroger le  $\mu C$

En suivant cette logique, notre programme envoie un caractère *NACK* au  $\mu C$  qui a été programmé pour reconnaître ce caractère. Une fois ce signal reçu, on peut afficher le message *NOK* sur le LCD. Il est évident que ce programme devra être lancé après avoir connecté Proteus au Terminal.

Tout au long de notre travail, nous nous sommes souvent demandé si le format des caractères envoyés était correct. Pour y remédier, nous avons utilisé un logiciel appelé **RealTerm NO Port Scan** qui nous a permis de *sniffer* les ports lors des échanges.

Nous avons rédigé un fichier README.md afin de faciliter l'utilisation du programme.

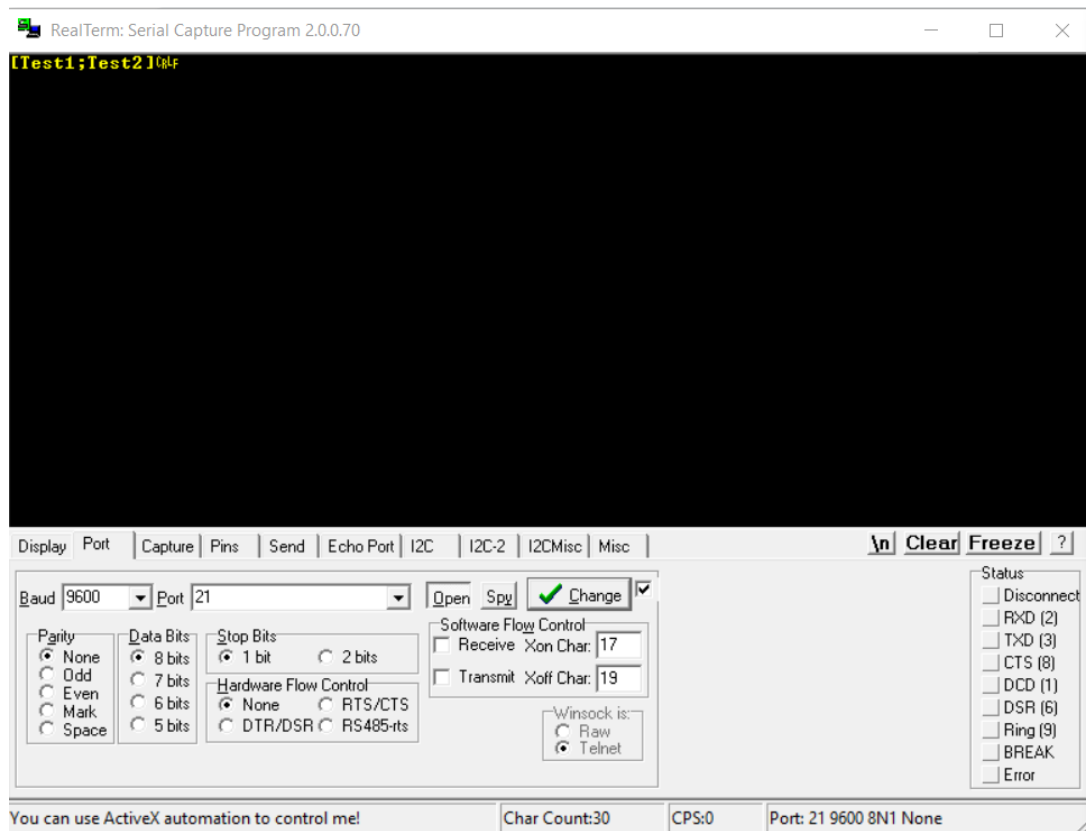


FIGURE 3 – RealTerm NO Port Scan

### 3 Schéma de principe

Le schéma de principe ainsi que la simulation ont été réalisés sur Proteus. Les composants utilisés sont les suivants :

- PIC18F45K22;
- 4 Résistances  $330\Omega$ ;
- 1 Potentiomètre  $1k\Omega$ ;
- 1 LCD 16x2;
- 4 Boutons poussoir;
- 6 Alimentations 5V;
- 1 Masse;
- 1 COMPIM (RS232).

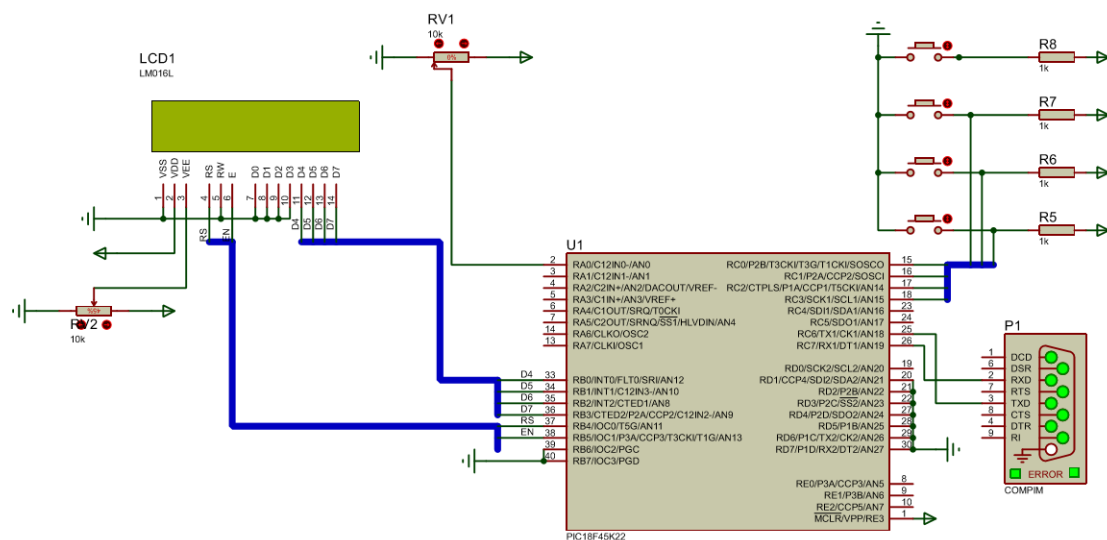


FIGURE 4 – Schéma de principe sur Proteus

Il est indispensable de bien configurer le composant appelé *COMPIM* afin de garantir la communication.

## 4 Algorithme

Toujours en partant du travail précédent, nous avons restructuré notre algorithme afin qu'il réponde au programme à coder. Afin de ne pas surcharger le rapport, nous ne présenterons ici que les nouvelles fonctions créées.

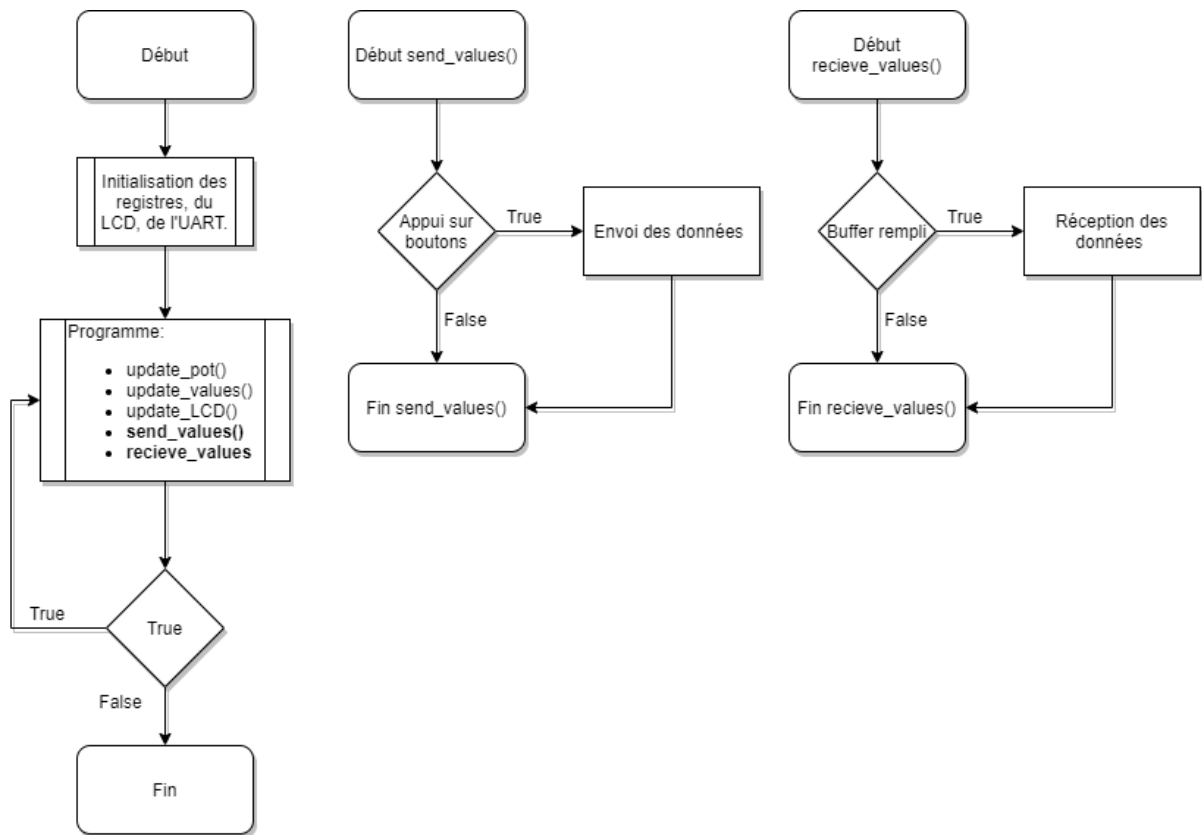


FIGURE 5 – Algorithme du programme principal



## 5 Code source

```
1 // Directives au preprocesseur
2 #define SOT 2
3 #define EOT 3
4 #define LF 10
5 #define CR 13
6
7 // Connexions du module LCD
8 sbit LCD_RS at RB4_bit;
9 sbit LCD_EN at RB5_bit;
10 sbit LCD_D4 at RB0_bit;
11 sbit LCD_D5 at RB1_bit;
12 sbit LCD_D6 at RB2_bit;
13 sbit LCD_D7 at RB3_bit;
14
15 sbit LCD_RS_Direction at TRISB4_bit;
16 sbit LCD_EN_Direction at TRISB5_bit;
17 sbit LCD_D4_Direction at TRISB0_bit;
18 sbit LCD_D5_Direction at TRISB1_bit;
19 sbit LCD_D6_Direction at TRISB2_bit;
20 sbit LCD_D7_Direction at TRISB3_bit;
21 // Fin des connexions du module LCD
22
23 // Declaration des variables globales
24 int val_1 = 0;
25 int val_2 = 0;
26
27 bit ready_to_send;
28
29 unsigned int pot_val;
30 unsigned int old_pot_val;
31
32 char content_line_1[16];
33 char content_line_2[16];
34 char buffer[12];
35 char recieved_data[12];
36
37 // Envoi des valeurs
38 void send_values() {
39     if (ready_to_send) {
40         UART1_Write(SOT);
41         UART1_Write_Text(content_line_1);
```

```

42     UART1_Write(EOT);
43     UART1_Write(LF);
44     UART1_Write(CR);
45     ready_to_send = 0;
46 }
47 }
48 // Reception des valeurs
49 void recieve_values() {
50     if (UART1_Data_Ready()) {
51         UART1_Read_Text(recieved_data, "]", 10);
52         strncpy(buffer, recieved_data+2, 1);
53
54         if (recieved_data[0] == 0x15) {
55             // Cas o le message provient du programme Python:
56             Lcd_Out(1, 14, "NOK");
57             Delay_ms(500);
58         }
59         else {
60             // Cas o le message provient du Terminal RS232:
61             strncpy(buffer, recieved_data+2, 3);
62             Lcd_Out(2, 3, buffer);
63             strncpy(buffer, recieved_data+5, 3);
64             Lcd_Out(2, 9, buffer);
65             Lcd_Out(1, 14, " OK");
66         }
67         memset(buffer, '\0', sizeof(buffer));
68     }
69 }
70
71 // Lecture de l'etat des boutons
72 void update_values() {
73     if (Button(&PORTC, 0, 1, 0) && val_1 + 10 <= 255) {
74         val_1 += 10;
75         ready_to_send = 1;
76         delay_ms(300);
77     }
78
79     if (Button(&PORTC, 1, 1, 0) && val_1 - 10 >= 0) {
80         val_1 -= 10;
81         ready_to_send = 1;
82         delay_ms(300);
83     }
84 }

```

```

85     if (Button(&PORTC, 2, 1, 0) && val_2 + 10 <= 255) {
86         val_2 += 10;
87         ready_to_send = 1;
88         delay_ms(300);
89     }
90
91     if (Button(&PORTC, 3, 1, 0) && val_2 - 10 >= 0) {
92         val_2 -= 10;
93         ready_to_send = 1;
94         delay_ms(300);
95     }
96 }
97
98
99 // Lecture de l'etat du potentiometre
100 void update_pot(){
101     old_pot_val = pot_val;
102     pot_val = ADC_Read(0);
103     if (old_pot_val != pot_val){
104         ready_to_send = 1;
105     }
106 }
107
108 // Met a jour l'ecran du LCD
109 void update_LCD(){
110     sprintf(content_line_1, "%03u %03u %04u", val_1, val_2, pot_val);
111     sprintf(content_line_2, "A:    B:");
112     Lcd_Out(1, 1, content_line_1);
113     Lcd_Out(2, 1, content_line_2);
114     //Lcd_Chr_CP(buffer);
115 }
116
117 // Affichage du message initial sur le Terminal
118 void terminal_init_message(){
119     UART1_Write_Text("Connect au PIC18F45K22");
120     UART1_Write(LF);
121     UART1_Write(CR);
122     UART1_Write_Text("Bienvenue mon programme...");
123     UART1_Write(LF);
124     UART1_Write(CR);
125     UART1_Write_Text("Format des donnees envoyer: [X;Y]");
126     UART1_Write(LF);
127     UART1_Write(CR);

```

```

128     UART1_Write_Text("O  X et Y sont des entiers compris entre 0 et
        999.");
129     UART1_Write(LF);
130     UART1_Write(CR);
131 }
132
133 // Affichage du message initial sur le LCD
134 void LCD_init_message() {
135     Lcd_Cmd(_LCD_CLEAR);
136     Lcd_Cmd(_LCD_CURSOR_OFF);
137     Lcd_Out(1,1,"Initialisation...");
138     Delay_ms(1000);
139     Lcd_Cmd(_LCD_CLEAR);
140 }
141 // Initialisation des routines uniques
142 void init() {
143     ANSELA = 0b00000001;
144     ANSELB = 0;
145     ANSEL_D = 0;
146     ANSELC = 0;
147
148     ready_to_send = 0;
149
150     // Desactive les comparateurs
151     C1ON_bit = 0;
152     C2ON_bit = 0;
153
154     // Initialise les entrees et sorties
155     TRISC = 0b10001111;
156
157     TRISA = 0b00000001;
158
159     /*Initialise les objets lies
160     au bibliotheques utilisees*/
161     ADC_Init();
162
163     UART1_Init(9600);
164     terminal_init_message();
165
166     Lcd_Init();
167     LCD_init_message();
168
169 }

```

```

170
171 // Fonction principale :
172 void main() {
173     init();
174
175     // Programme :
176     for (;;) {
177         update_pot();
178         update_values();
179         update_LCD();
180         send_values();
181         recieve_values();
182     }
183 }

```

## 6 Analyse du code source

Nous avons déjà analysé une partie de ce programme lors du rapport précédent. Nous nous focaliserons ici sur les nouveaux éléments présents à savoir la bibliothèque associée à l'UART.

Par rapport au dernier travail réalisé, deux fonctions clefs sont venues s'ajouter au programme principal : `void send_values()` et `void recieve_values()`.

— `void send_values()` :

```

37 // Envoi des valeurs
38 void send_values() {
39     if (ready_to_send) {
40         UART1_Write(SOT);
41         UART1_Write_Text(content_line_1);
42         UART1_Write(EOT);
43         UART1_Write(LF);
44         UART1_Write(CR);
45         ready_to_send = 0;
46     }
47 }

```

Cette fonction effectue sa routine si le flag `ready_to_send` est mis à 1 par les fonctions `void update_pot()` et `void update_values()`. Après cette mise à 1 a lieu l'encapsulation des données. Enfin, le flag est remis à 0 afin de ne pas répéter l'opération. Nous pouvons à présent analyser la fonction de réception

de données.

— `void` `recieve_values()` :

```
48 // Reception des valeurs
49 void recieve_values() {
50     if (UART1_Data_Ready()) {
51         UART1_Read_Text(recieved_data, "", 10);
52         strncpy(buffer, recieved_data+2, 1);
53
54         if (recieved_data[0] == 0x15) {
55             // Cas o le message provient du programme Python:
56             Lcd_Out(1, 14, "NOK");
57             Delay_ms(500);
58         }
59         else {
60             // Cas o le message provient du Terminal RS232:
61             strncpy(buffer, recieved_data+2, 3);
62             Lcd_Out(2, 3, buffer);
63             strncpy(buffer, recieved_data+5, 3);
64             Lcd_Out(2, 9, buffer);
65             Lcd_Out(1, 14, " OK");
66         }
67         memset(buffer, '\0', sizeof(buffer));
68     }
69 }
```

Dans ce cas, il s'agit de n'exécuter la routine que si le registre de réception de données de l'UART1 est rempli. Cette vérification se fait par la fonction `UART1_Data_Ready()`. Elle est nécessaire car la fonction `UART1_Read_Text()` est bloquante. Ensuite, en fonction du contenu du buffer, nous pouvons afficher le message *OK* / *NOK*. Dans le cas où on écrit *OK* sur l'écran LCD, nous désencapsulons les données envoyées par le Terminal afin de l'afficher à l'endroit prévu. Il est primordial de connaître le format des données reçues afin de réaliser cette étape.

Notons qu'une variable intermédiaire a été créée afin de ne pas traiter directement les données reçues. Cette variable est réinitialisée<sup>2</sup> à chaque boucle.

---

2. La fonction `memset(buffer, '', sizeof(buffer))` chaque caractère du buffer par un *nullByte*

## 7 Conclusion

Après avoir mis en place une communication UART dans différents environnements ( $\mu C$ , *PC*, *Python*, *RealTerm*...), nous avons une meilleure connaissance de ce protocole. Nous sommes capables de recevoir, envoyer, formater, encapsuler tout type de données et ce dans n'importe quel d'environnement.

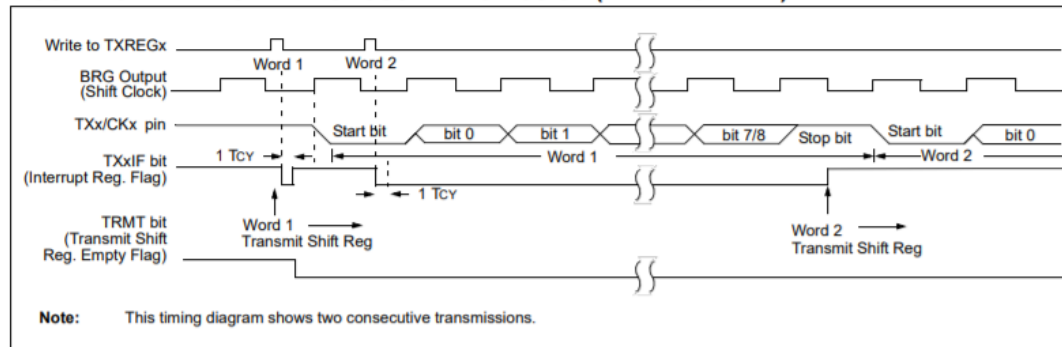
Nous retenons également l'utilisation de flags afin d'activer une action liée à un *événement*. N'ayant pas eu l'occasion d'y remédier pour cette séance, cette méthode sera appliquée pour vérifier les flancs descendants lors d'appuis sur les boutons pour la prochaine séance.

Nous sommes tout de même déçus de ne pas avoir pu établir l'état de la communication en utilisant le *polling* ou les *interrupt*. Nous espérons avoir une idée plus claire lors du prochain TP consacré aux interruptions système. Nous avons tout de même découvert la bibliothèque *pySerial* qui nous a permis de mettre en évidence le fait qu'un port série ne pouvait être utilisé que par un seul processus et de "profiter" de ce fait.

## A ANNEXE - Datasheet PIC18F45K22

### PIC18(L)F2X/4XK22

**FIGURE 16-4: ASYNCHRONOUS TRANSMISSION (BACK-TO-BACK)**



### PIC18(L)F2X/4XK22

**FIGURE 16-5: ASYNCHRONOUS RECEPTION**

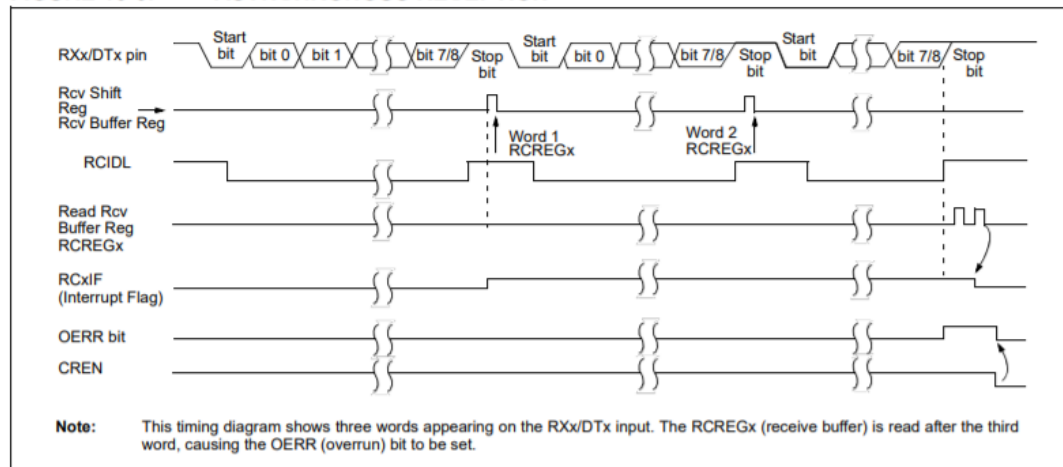


FIGURE 6 – Module UART provenant du datasheet



## B Programme Python

```
1 """
2 Auteur: KORKUT Caner – Ephec ISAT
3 Veuillez connecter le Terminal RS232 au microcontrôleur avant de
   lancer le programme.
4 Utilisation: ex: "python3 TP3.py 21"
5 """
6
7 import serial
8 import time
9 import sys
10
11 myCom = serial.Serial()
12 portNumber = sys.argv[1]
13 myCom.port = 'COM'+str(portNumber)
14
15
16 packet = bytearray()
17 packet.append(0x15) # "NAK" caractère de non acquis attendu par le
   PIC
18 packet.append(0x5D) # "]" caractère de fin attendu par le PIC
19 #packet = b'\x15\x5D' # => Ceci aurait pu substituer les 3 lignes
   précédentes
20
21
22 #Programme:
23 while True:
24
25     try: # On essaye de se connecter au même port que le Terminal.
26         myCom.open()
27
28     except serial.SerialException: # Port occupé: probablement utilisé
   par le Terminal
29         print("Microcontrôleur connecté au Terminal...\n")
30
31     else: # Si on y arrive:
32         myCom.write(packet)
33         myCom.close()
34         print('Envoi de paquets "NACK" au PIC18F45K22. => Terminal
   d connecté.')
35         time.sleep(0.5)
```

## C Bibliographie

- <https://bit.ly/38cjQPF>
- <https://bit.ly/3cotq5t>
- <https://bit.ly/3ck0yJG>
- <https://bit.ly/38hUC2d>
- <https://bit.ly/2TcJnDX>
- <https://bit.ly/38evwBw>
- <https://bit.ly/2TH3r0b>
- <https://bit.ly/3aicNq3>