GESTION DES COMMUNICATIONS ASYNCHRONES SUR PIC18 EN C18

Objectifs :

Comprendre le fonctionnement d'une communication asynchrone NRZ (No Return to Zero)

Etre capable de définir les valeurs des registres d'un PIC18 associés aux communications asynchrones dans un contexte donné.

Mettre en œuvre une bibliothèque de gestion des communications.

Configuration ISIS :

Frequence du PIC : clic-droit sur le PIC puis « edit properties »
entrez la fréquence de l'oscillateur dans « Processor Clock Frequency »

Vitesse de communication du terminal : clic-droit sue le terminal puis «edit properties » , compléter « Baud Rate ».

Conditions : 4h à 8h en classe

Communications_séries_asynchrones_P18fxx20.pdf,

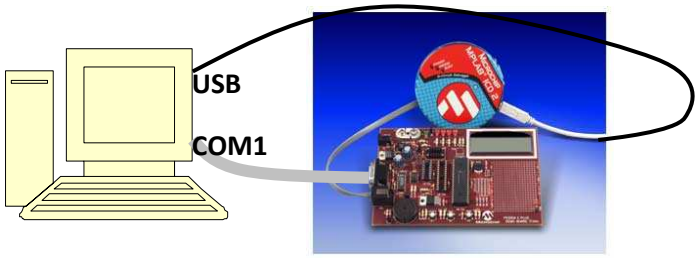
Data-Sheet PIC18F4620, KIT PICDEM2+ ou simulateur ISIS

(carte PICDEM2+ with PIC18F4620.dsn)

Schéma carte PICDEM2+ équipée d'un microcontrôleur PIC18F4620

libusart.c (dossier c:\tpmcc18vx)

Etablir un compte rendu propre et réutilisable pour la projet technique.



1) A partir du schéma de la carte PICDEM2+, dessiner un schéma partiel depuis le connecteur du PC jusqu'au microcontrôleur, faisant bien apparaitre les lignes de transmissions (RX et TX)
2) Indiquer le rôle du circuit MAX232 ?
   Quelle est la longueur maxi d'une communication RS232 à 19200 Bauds, 9600 Bauds, 4800 Bauds (sources Wikipedia)?
3) Rechercher (si besoin) sur Internet la signification de RX et de TX, expliquer pourquoi la broche 3 du DB9 du PICDEM2 est connecté à « Transmit data from PC » et la broche 2 du DB9 TX à « Receive data to PC »

4) Etablir une connexion entre une maquette PICDEM2+ et un PC par RS232.
   Si nécessaire, réaliser un câble de communication de 1 mètre, prise DB9, connexions TX,RX et GND (3 fils et bouclages utiles). (Le schéma de la connexion pourra être trouvé sur le document joint)

Essayer le programme « **echo »** qui réalise un « echo+1 » entre le PC et le PIC. (Dernière page)

Analyser ce programme, pour cela répondre aux questions :

5) Quelle est la fréquence de l'horloge du PIC.
6) Que veut dire 9600 BAUDS, 8 bits, 1 STOP
7) indiquer le rôle de TXSTA et justifier sa valeur à 0x20
8) indiquer le rôle de RCSTA et justifier sa valeur à 0x90
9) Quel est le rôle du bit BRG16
10) Quel est le rôle du bit BRGH
11) indiquer le rôle de SPBRG et justifier sa valeur à 51
12) indiquer le rôle des bits RCIF, TMRT et TXIF.
13) Que fait la fonction putch ? la fonction data_recue ?
14) indiquer le rôle du registre RCREG
15) Compléter le programme « echo » de manière à utiliser l'horloge externe du PICDEM2+ (FOSC=4Mhz) et obtenir la même vitesse de communication.
16) Tester le programme, relever à l'oscilloscope en mode mono-coup synchronisé sur front descendant et en en correspondance la « trame » du caractère 'A' à l'entrée et à la sortie du convertisseur +12/-12 vers 0/+5 . Repérer sur cette trame, le bit de start, les 8 bits de données et le bit de stop
17) Modifier ce le programme de afin de configurer les communications en 1200,n,8,1 en utilisant l'oscillateur interne à32Mhz (et non plus l'oscillateur externe). Effectuer les tests et visualiser le résultat sur un oscilloscope comme précédemment en mettant en évidence la nouvelle durée d'un bit.

Le programme précédent fonctionne bien lorsque le débit est suffisamment lent pour permettre au programme de traiter une donnée avant que la suivante n'arrive. Si le débit est trop important (envoie d'un fichier ou d'un groupe de données) il y a risque d'écrasement du registre RCREG et donc perte de donnée. Afin de palier ce problème on gère la réception des données en interruption. Lorsqu'une donnée arrive dans RCREG, une interruption est générée. Le sous-programme d'interruption récupère la donnée et la stocke dans un buffer tournant (dans la RAM). Les données peuvent ainsi être traitées plus tard.

La bibliothèque libusart.c réalise cela : fonctions de la bibliothèque :

`void initsci(void)` configuration BAUD et interruption (9600,n,8,1)

`char getsci(void)` retourne le plus ancien caractère reçu dans le buffer (si le buffer est vide, attend un caractère).

`char *getstsci(char *s, char finst`) lit une chaîne de caractère sur SCI se terminant par finst

`char carUSARTdispo(void)` Cette fonction retourne 1 (vrai) si un caractère est disponible dans le buffer de réception et 0 si non

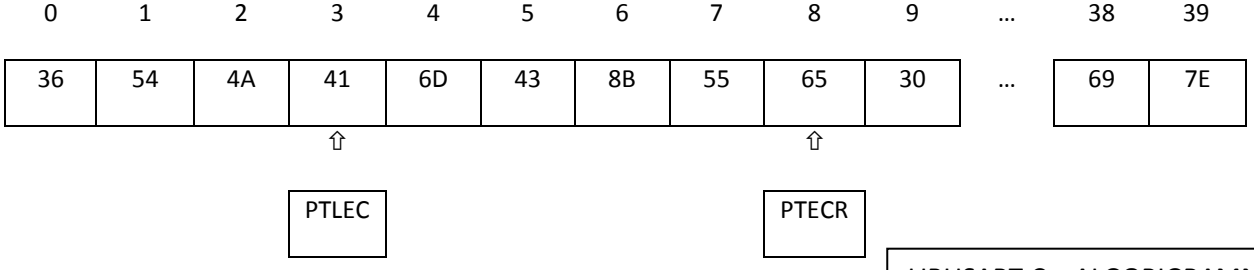
18) Compléter et essayer le programme tstlibUSART.c ci-dessous. 9600 Bauds, N, 8, 1 (penser à intégrer la bibliothèque pour l'horloge interne dans le projet)

```
#include <p18f4620.h>
#include <stdio.h>  // pour fprintf
#include "libusart.c"

unsigned char chaine[20] ;
void main(void)
{
init_horloge_interne() ;
initsci();  // ATTENTION CETTE FONCTION SERA ADAPTEE A L'HORLOGE INTERNE
fprintf (_H_USART, "Les communications sont ouvertes\n\r");
while(1)
    {
      fprintf(_H_USART,"\n\rTapez un caractere : ");
      fprintf(_H_USART,"%c",getsci()+1);
      fprintf(_H_USART,"\n\rTapez un message (finir par '*') : ");
      fprintf(_H_USART,"%s",getstsci(chaine,'*'));
    }
}
```

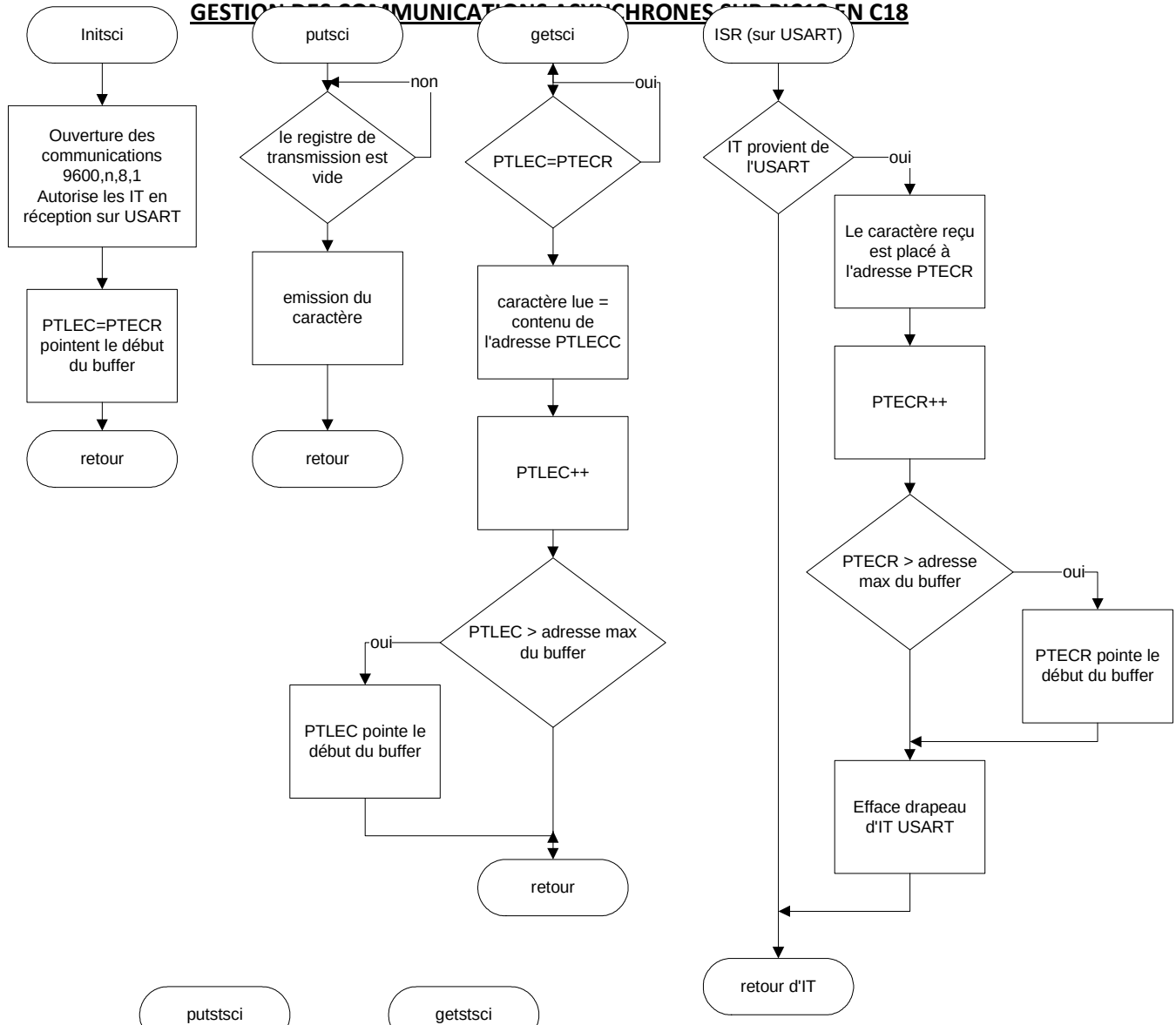
19) Indiquer le rôle des fonctions initsci, getsci et getstsci
20) Utiliser la fonction de transfert de fichier texte afin de transmettre un fichier d'une trentaine d'octets vers le PIC, vérifier la bonne réception de ce texte dans la mémoire du PIC.
21) Proposer un programme recopiant sur l'afficheur LCD du kit PICDEM2+ les messages envoyés depuis le PC (on pourra prendre l' '*' comme caractère de fin de message).
22) Réaliser un programme affichant sur le terminal du PC le contenu de la ROM du PIC, en commençant à l'adresse 0 par groupes de 16 lignes de 16 octets (utiliser un pointeur et les fonctions BTOA et ITOA de la bibliothèque stdlib)

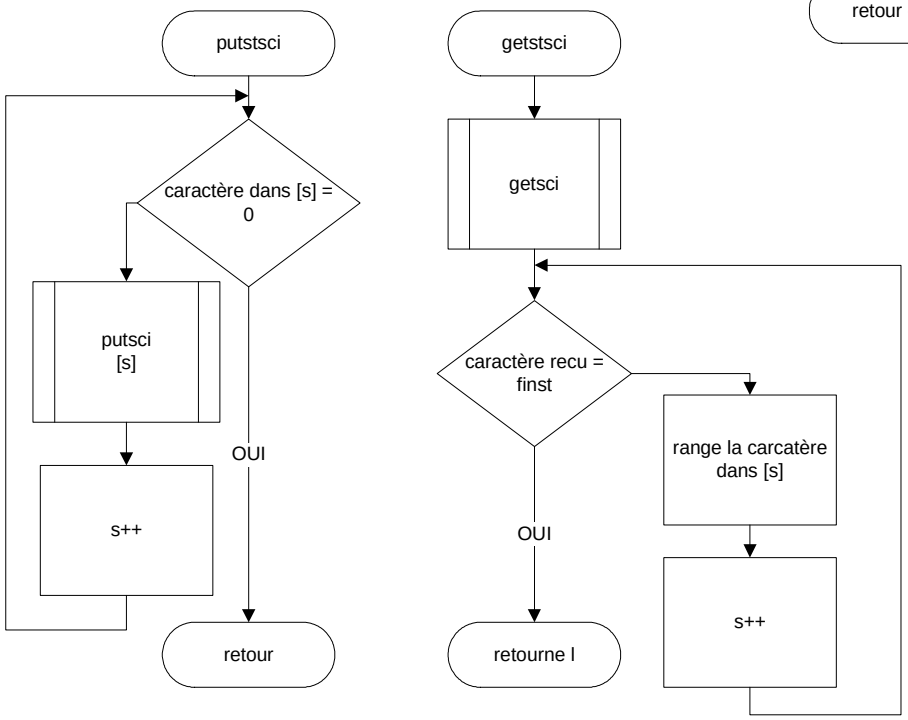
PRINCIPE LIBUSART

Tableau de réception : Les caractères reçus sont rangés dans un tableau (buffer) à l'adresse d'un pointeur PTECR qui est ensuite incrémenté. getsci retourne le caractère ponté par PTRECR puis PTRECR est incrémenté. Lorsque tous les caractères reçus ont été lus, PTERC=PTLEC. Si l'un des pointeurs dépasse l'adresse max du tableau il pointera à nouveau le début [PTLEC-1] représente le dernier caractère lu et [PTECR-1] le dernier caractère reçu.



LIBUSART.C – ALGORIGRAMME


Initsci
Ouverture des communications 9600,n,8,1 Autorise les IT en réception sur USART
PTLEC=PTECR pointent le début du buffer
retour
putsci
le registre de transmission est vide
non
emission du caractère
retour
getsci
PTLEC=PTECR
oui
caractère lue = contenu de l'adresse PTLECC
PTLEC++
PTLEC > adresse max du buffer
oui
PTLEC pointe le début du buffer
retour
ISR (sur USART)
IT provient de l'USART
oui
Le caractère reçu est placé à l'adresse PTECR
PTECR++
PTECR > adresse max du buffer
oui
PTECR pointe le début du buffer
Efface drapeau d'IT USART
retour d'IT

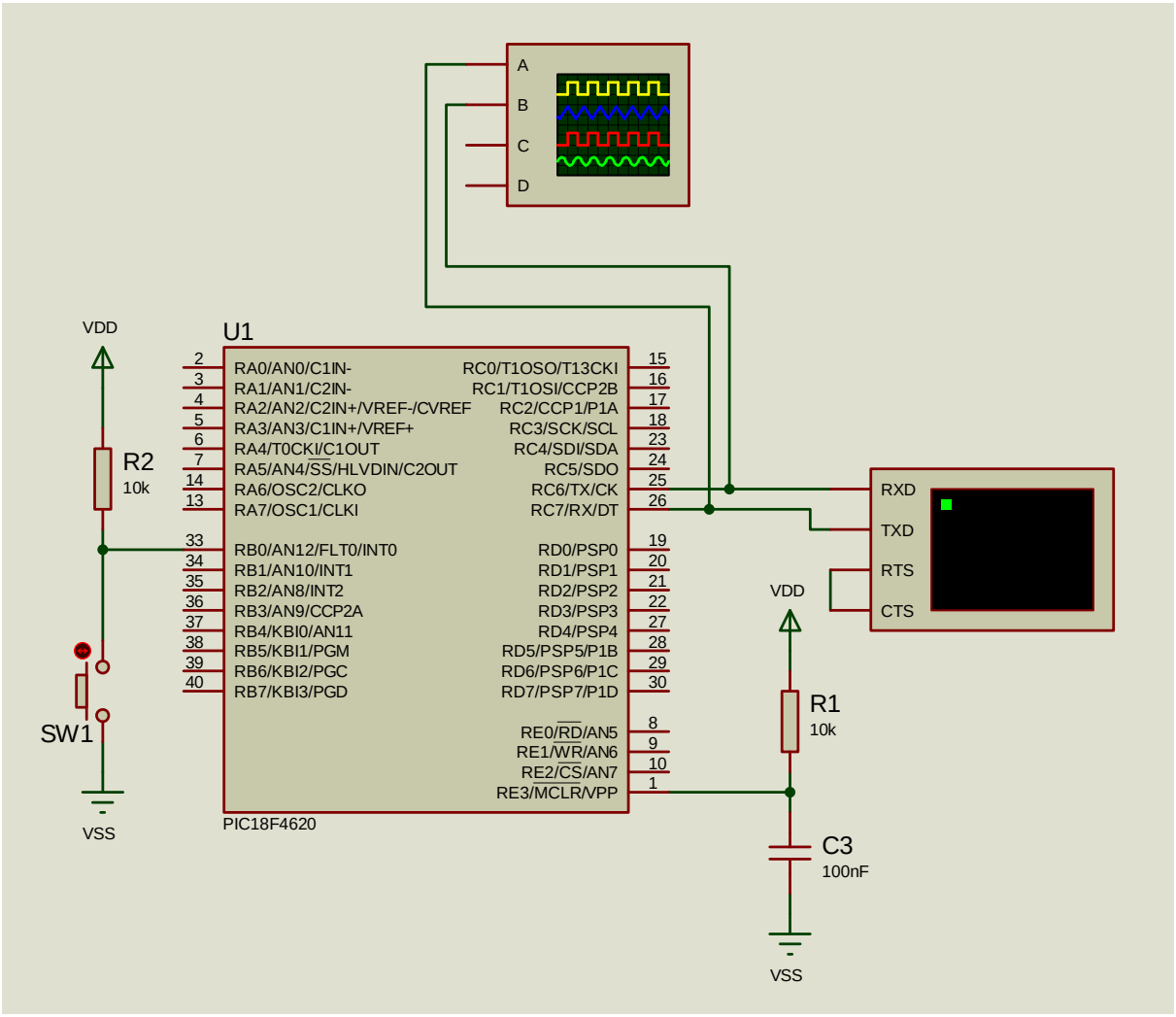

putstsci
caractère dans [s] = 0
putsci [s]
s++
OUI
retour
getstsci
getsci
caractère recu = finst
range la carcatère dans [s]
s++
OUI
retourne l

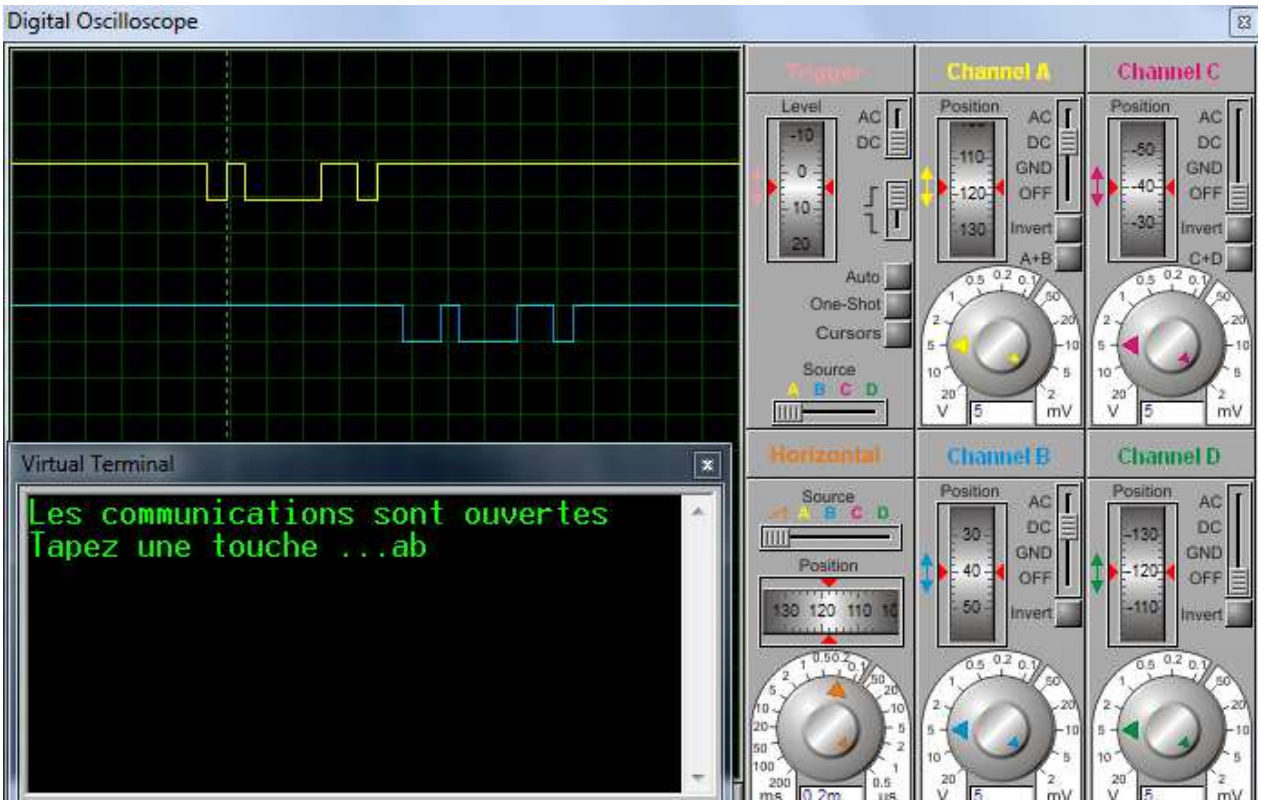
PROGRAMME echo.c

```
#include <p18f4620.h>
#include “init_horloge_interne.c”
rom char mess[]="\nLes communications sont ouvertes\nTapez une touche
...\n\n";

// indique qu'un caractère est dans RCREG de l'USART
char data_recue(void)
{
     if (PIR1bits.RCIF)      /* char recu en reception*/
     {
      PIR1bits.RCIF=0;  // efface drapeau
      return (1);            // indique qu'un nouveau caractère est dans
RCREG
     }
     else return (0);        // pas de nouveau caractère reçu
}

// envoie un caractère sur USART
void putch(unsigned char c)        //putch est défini sur le port série
{
     while(!TXSTAbits.TRMT);             // pas de transmission en cours ?
    TXREG=c;                          /* envoie un caractère */
    while(!PIR1bits.TXIF);
}

// envoie une chaine en ROM
void putchaine(rom char* chaine)
{
     while (*chaine) putch(*chaine++);
}

void main(void)
{
     init_horloge_interne();
     TXSTA = 0b00100000;
     RCSTA = 0b10010000;
     PIE1bits.TXIE=0;   // IT en emission désactivée
     PIE1bits.RCIE=0;   // IT en reception déactivée
     BAUDCONbits.BRG16=0;
     TXSTAbits.BRGH=0;
     SPBRG = 51;
     putchaine(mess);   // intro
     while(1)     // echo +1 ex: si ‘a’ est transmis ‘b’ est envoyé en echo
         {
             if (data_recue()) putch(RCREG+1);
      }
}
```

Init_horloge_interne.c

```
// initialisation horloge PIC

#if defined (__18F4620)|| defined (__18F2620) || defined (__18F4520)
#pragma config OSC = INTIO67 //pour INTRC-OSC2 as RA6, OSC1 as RA7
#pragma config WDT = OFF     //pour watch dog timer disable
#pragma config LVP = OFF     //pour low voltage program disable
#pragma config PBADEN=OFF     // PRB est numerique
#elif defined(__18F452) || defined (__18F252)
#pragma config OSC = XT
#pragma config WDT = OFF     //pour watch dog timer disable
#pragma config LVP = OFF     //pour low voltage program disable
#endif

void init_horloge_interne(void)
{
#if defined(__18F4620)|| defined (__18F2620)         // config Q interne et
freq 8MHz
     OSCCONbits.IRCF2=1;
     OSCCONbits.IRCF1=1;
     OSCCONbits.IRCF0=1;
     OSCTUNEbits.PLLEN=1;     // PLL  (x4)
     OSCCONbits.SCS1=0;       // osc interne
     OSCCONbits.SCS0=0;
     ADCON1=0x0F;             // active tout le PORTB en numerique
#endif
}
```


A
B
C
D
VDD
U1
R2
10k
SW1
VSS
RA0/AN0/C1IN-
RA1/AN1/C2IN-
RA2/AN2/C2IN+/VREF-/CVREF
RA3/AN3/C1IN+/VREF+
RA4/T0CKI/C1OUT
RA5/AN4/SS/HLVDIN/C2OUT
RA6/OSC2/CLKO
RA7/OSC1/CLKI
RB0/AN12/FLT0/INT0
RB1/AN10/INT1
RB2/AN8/INT2
RB3/AN9/CCP2A
RB4/KBI0/AN11
RB5/KBI1/PGM
RB6/KBI2/PGC
RB7/KBI3/PGD
RC0/T1OSO/T13CKI
RC1/T1OSI/CCP2B
RC2/CCP1/P1A
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RD0/PSP0
RD1/PSP1
RD2/PSP2
RD3/PSP3
RD4/PSP4
RD5/PSP5/P1B
RD6/PSP6/P1C
RD7/PSP7/P1D
RE0/RD/AN5
RE1/WR/AN6
RE2/CS/AN7
RE3/MCLR/VPP
PIC18F4620
RXD
TXD
RTS
CTS
VDD
R1
10k
C3
100nF
VSS


Digital Oscilloscope
Trigger
Channel A
Channel C
Level
Position
AC
DC
GND
OFF
Invert
A+B
C+D
Auto
One-Shot
Cursors
Source
Horizontal
Channel B
Channel D
Virtual Terminal
Les communications sont ouvertes
Tapez une touche ...ab