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.

Conditions : 15h en classe

Data-Sheet PIC18F26K22, KIT PICDEM2+ ou simulateur ISIS

Schéma carte PICDEM2+ équipée d'un microcontrôleur PIC18F26K22

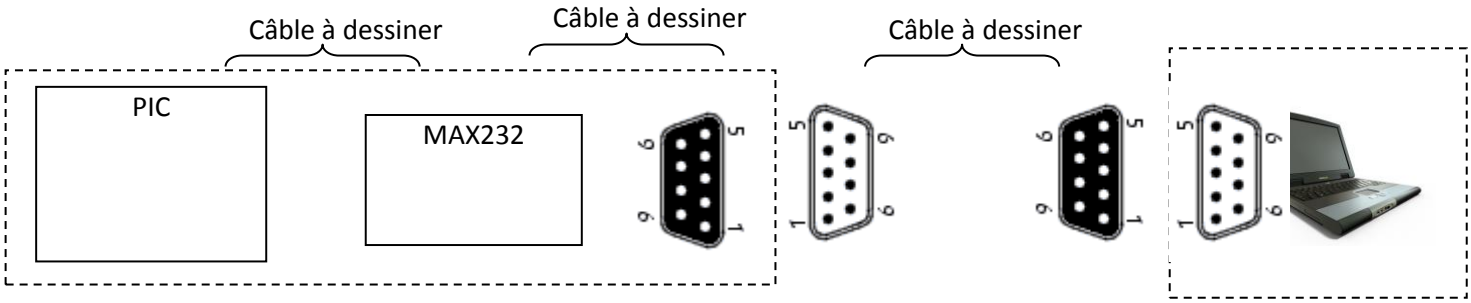
libusart.c

Emulateur de terminal « terminal.exe »

Noms :

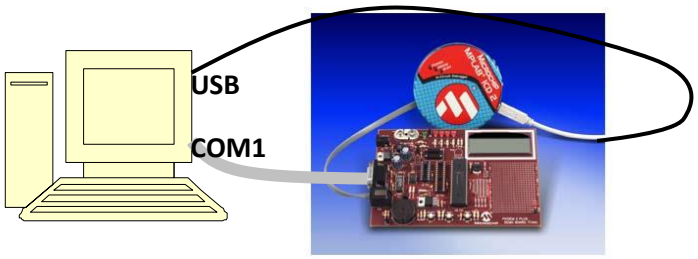
Etablir un compte rendu propre et réutilisable pour l'épreuve de projet technique.

1) A partir du schéma de la carte PICDEM2+ et du brochage d'un connecteur DB9 sur PC (à rechercher sur Internet), compléter le schéma partiel ci-dessous depuis le connecteur du PC jusqu'au microcontrôleur, faisant bien apparaître la masse ainsi que les lignes de transmissions (RX et TX) ainsi que les noms et numéros des broches utilisées (DB9 femelles en noir, DB9 mâles en blanc).



2) Indiquer le rôle du circuit MAX232 ? (effectuer une recherche sur le site MAXIM-IC)

3) Configurer le projet MPLAB proposé pour la carte PICDEM2+ avec debugger ICD et le programme echo.c. A l'aide du câble réalisé, connecter le KIT à un PC équipé du logiciel « terminal.exe » configuré en 9600 bauds, données sur 8 bits, pas de test de parité, un bit de stop.
Tester le programme ainsi que les communications full-duplex.



4) Relever à l'oscilloscope (prélever les signaux sur le connecteur de la carte) en mode mono-coup synchronisé sur front descendant et en correspondance la « trame » du caractère ASCII 'A' à l'entrée et à la sortie du convertisseur +12/-12 vers 0/+5 (voies A et B). Repérer sur cette trame, le bit de start, les 8 bits de données et le bit de stop, mesurer précisément la vitesse de communication...
- Relever maintenant sur la voie A le signal sur TX du PIC et sur la voie B le signal sur RX du PIC.
Imprimer les oscillogrammes obtenus sur le compte rendu.

Analyser le programme echo.c, pour cela répondre aux questions :

5) Quelle est la fréquence de l'horloge du PIC (FOSC). En déduire FCY=FOSC/4

6) indiquer le rôle de TXSTA et justifier sa valeur à 0x20

7) indiquer le rôle de RCSTA et justifier sa valeur à 0x90

8) Quel est le rôle du bit BRG16

9) Quel est le rôle du bit BRGH

10) indiquer le rôle de SPBRG et justifier sa valeur à 51

11) indiquer le rôle des bits RCIF, TMRT et TXIF.

12) Que fait la fonction putch

13) Que fait la fonction data_recue ?

14) indiquer le rôle du registre RCREG

15) Modifier le programme « echo.c » de afin de configurer les communications en 1200,n,8,1 en utilisant l'oscillateur interne à32Mhz .. Effectuer les tests et visualiser le résultat sur un oscilloscope comme précédemment en mettant en évidence la nouvelle durée d'un bit.

Transmission d'un message à l'aide de la fonction fprintf.

16) Le premier paramètre de fprintf indique la destination des données.
H_USART pour une sortie sur l'USART (TXREG)
H_USER pour une sortie sur la fonction _USER_PUTC (cas d'un afficheur LCD par exemple)

Analyser la programme fprintf_USART.c, le modifier de manière à renvoyer vers le PC l'état de la touche S2.
ex : « Touche S2 relâchée » ou « Touche S2 appuyée »
note : le caractère ASCII \n descend le curseur d'une ligne sur le terminal
le caractère ASCII \r retourne le curseur en début de ligne sur le terminal

GESTION DES COMMUNICATIONS EN INTERRUPTION

Le programme précédent fonctionne bien lorsque le débit est suffisamment lent (caractères tapé au clavier par exemple) 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 pallier 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

`int putstsci(unsigned char *s),` envoie la chaine s contenue en RAM
remplaçable par fprintf(_H_USART, « %s »,chaine) ;

`int putrstsci(rom unsigned char *s)` envoie la chaine s contenue en ROM
remplaçable par fprintf(_H_USART, « %S »,chaine) ;

`void putsci(unsigned char c) envoie` le caratère c
remplaçable par fprintf(_H_USART, « %c »,c) ;

17) Essayer le programme tstlibUSART.c qui met en œuvre la bibliothèque libusart.c (penser à intégrer la bibliothèque pour l'horloge interne dans le projet ainsi que la bibliothèque libusart.c)
Expliquer ce que fait ce programme

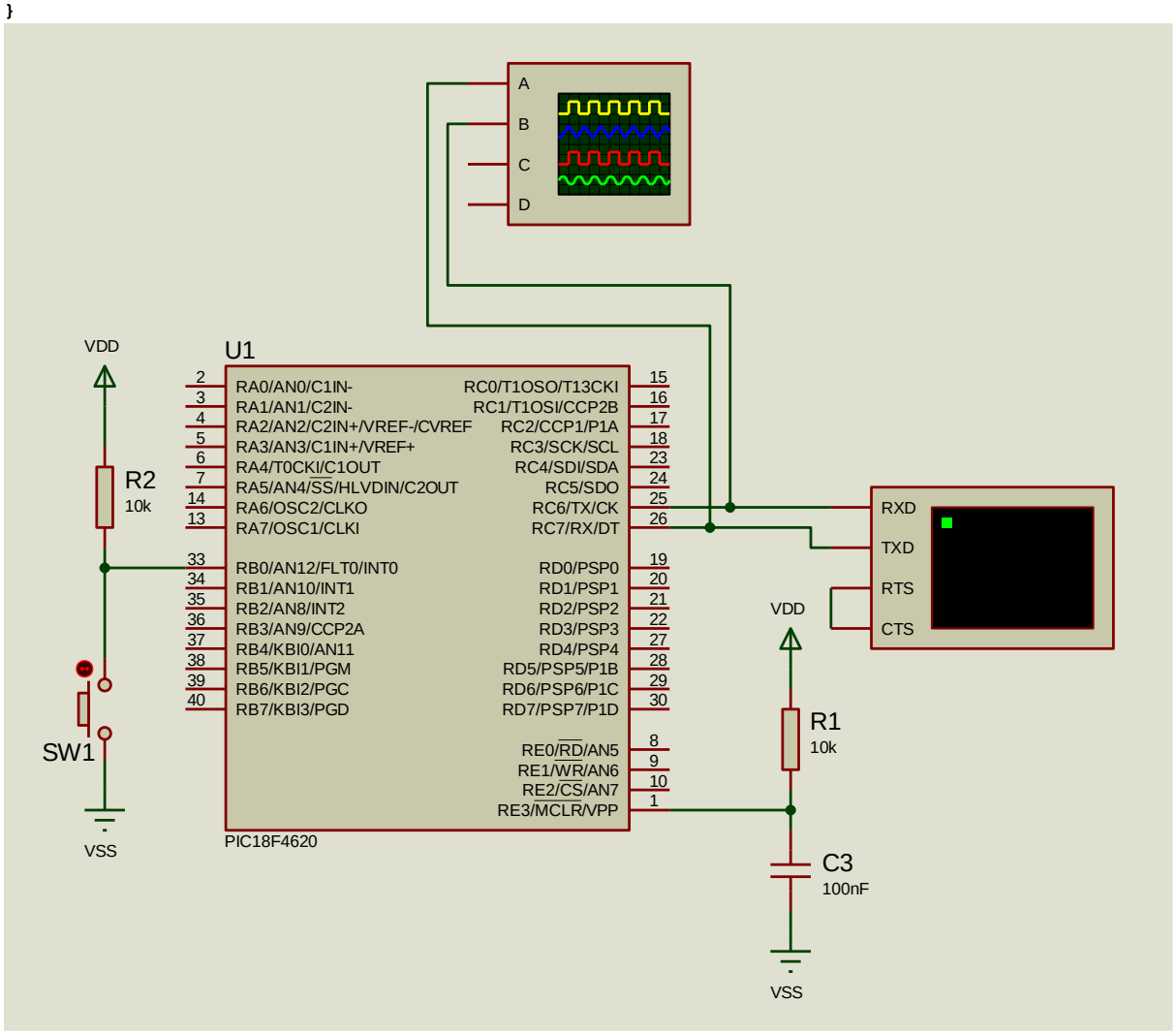
18) Relever les trames complètes de tstlibUSART.c à l'aide d'un analyseur logique

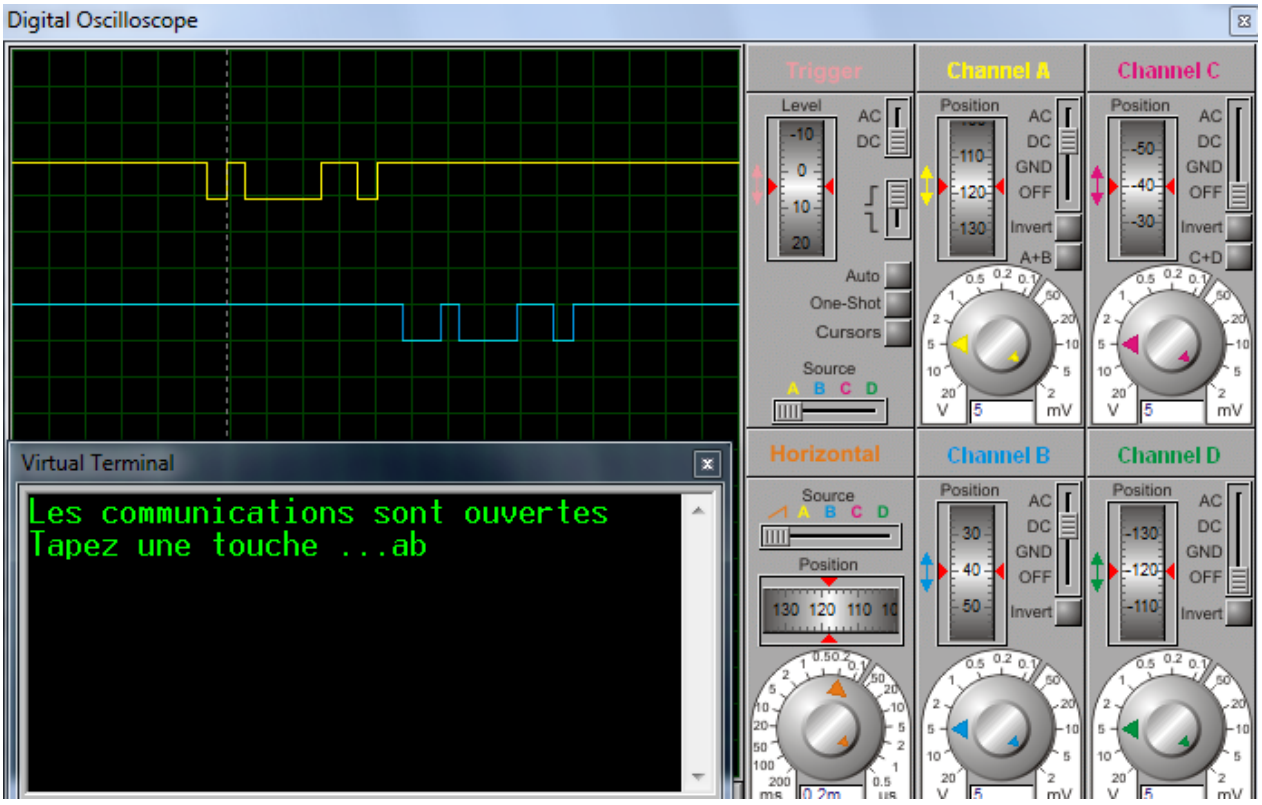
19) A partir du programme précédent, proposer un programme recopiant sur l'afficheur LCD du kit PICDEM2+ les messages envoyés depuis le PC (on pourra prendre l' '*' comme caractère de fin de message et utiliser MAESTRO pour construire la bibliothèque LCD).

20) 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)

```
// C.Dupaty 10/2014
// demo UART avec MPLAX PROTEUS8 sur P18F26K22
// configuration de l'UART : 9600,n,8,1
// P18F26K22 : oscillateur interne 16MHz la division par défaut donne 1MHz
// test de printf
// test de reception d'un caractere
// test d'emission d'un caractere

#include <xc.h>     // pour registres uC
#include <stdio.h>  // pour printf

// Horloge interne et autres configurations
#pragma config FOSC = INTIO67, FCMEN = OFF, PLLCFG=OFF                    // CONFIG1H
#pragma config IESO=OFF,PWRTEN=OFF,BOREN=OFF,WDTEN=OFF,CCP2MX=PORTC1      // CONFIG2H
#pragma config PBADEN=OFF,T3CMX=PORTC0,P2BMX=PORTC0,CCP3MX=PORTB5  // CONFIG3H
#pragma config MCLRE=EXTMCLR,STVREN=OFF,LVP=ON,XINST=OFF,DEBUG=OFF // CONFIG4H
#pragma config CP0 = OFF, CP1 = OFF, CP2 = OFF, CP3 = OFF               // CONFIG5L
#pragma config CPB = OFF, CPD = OFF                                     // CONFIG5H
#pragma config WRT0 = OFF, WRT1 = OFF, WRT2 = OFF, WRT3 = OFF           // CONFIG6L
#pragma config WRTB = OFF, WRTC = OFF, WRTD = OFF                       // CONFIG6H
#pragma config EBTR0 = OFF, EBTR1 = OFF, EBTR2 = OFF, EBTR3 = OFF       // CONFIG7L
#pragma config EBTRB = OFF

// 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(!PIR1bits.TXIF);           // pas de transmission en cours ?
    TXREG=c;                         /* envoie un caractère */
}

void initUART(void)
{
    ANSELC=0;
    TRISCbits.TRISC7 = 1; // RX - Set Recieve pin for tristate
    TRISCbits.TRISC6 = 0; // TX - Set as Output
    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;
}

void main(void)
{
    OSCCON = 0b01100000;  // IRCF=110 donc sortie 8MHz SCS=00 donc oscillateur primaire
    OSCTUNE = 0b01000000; // PLL active, l'oscillateur sera finalement 32MHz (compatibilite
18F2620)
                          // FOSC = 32MHz et FCY=8MHz , TCY=125ns
    initUART();

    printf("Les communications sont ouvertes\n\rTapez une touche\n\r");   // intro
       while(1)
           {
              if (data_recue()) printf("Vous avez tape : %c \n\r",RCREG);
       }
```

}


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
Horizontal
Channel B
Channel D
Virtual Terminal
Les communications sont ouvertes
Tapez une touche ...ab