3 Gestion de la liaison série asynchrone

Description générale

- **Date du TP :** 6ième semaine ;
- **Durée du TP :** 3h (2semaines) ;
- **Matériels nécessaires** : Carte d'experimentation, microcontrôleur PIC 16f877, programmateur, des composants électroniques ;
- **Logiciels utilisés :** Complilateur C PCW, Simulateur ISIS, IC Writer ;
- **Public cible :** 4ième niveau d'études de technicien supérieur en informatique industrielle ;
- **Partie du cours en rapport avec le TP :** Classification architecturale des systèmes temps reel, les microcontrôleurs.

Objectifs

- Gestion et paramètrage de liaison série RS232 ;
- Savoir le protocole RS232 ;
- Gestion d'interruption ;
- Se familiariser avec les outils de programmation et de développement ;
- Approfondir les acquis en programmation C des microcontrôleurs PIC ;
- Maitrise des architectures des microcontrôleurs Microchip PIC16.

Pré requis

Notions générales sur :
- L'électronique ;
- Algorithmique ;
- Programmation en C ;
- Les microcontrôleurs.

3.1 Introduction

Dès lors que votre application doit communiquer avec des circuits ou des équipements externes elle doit être a même de respecter les protocoles de dialogue de ces derniers. Ces protocoles sont toutefois relativement peu nombreux et les PIC les mieux dotés en ressources internes, qui sont aussi rappelons-le ceux qui sont les plus intéressants pour la programmation en C, disposent généralement des interfaces matérielles nécessaires qu'il ne reste donc plus qu'à mettre en oeuvre. Ce TP va donc s'intéresser à ces interfaces et plus précisément nous allons y étudier comment communiquer avec "le monde extérieur" via une liaison série asynchrone.

3.2 Mise en oeuvre de la liaison série asynchrone

La liaison série asynchrone, très souvent appelée RS 232, permet de relier votre application à tous les équipements informatiques qui en sont équipes. Cela peut être un PC qui exécute l'hyper terminal de Windows bien sûr, mais aussi de très nombreux autres équipements péri informatiques tels que : imprimantes spécialisées, lecteurs de cartes magnétiques ou à puces, modems, etc. L'exploitation d'une telle liaison a lieu en principe grâce à un UART, et tous les PIC du haut de la famille l6xxxx et de la famille 18xxxx en sont équipés. Pour ceux qui n'en possèdent pas, la majorité des compilateurs C est capable de générer un UART "par logiciel", c'est-à-dire en fait de transformer deux lignes de ports parallèles en entrée et sortie série asynchrone au moyen d'un programme adéquat. Les performances qu'il faut en attendre ne sont toutefois pas celles permises si l'on utilise un véritable UART matériel et le programme nécessaire occupe une place non négligeable en mémoire.
En résumé : si vous avez besoin d'une liaison série asynchrone performante, choisissez un PIC contenant un UART et faites appel à lui au moyen de la directive appropriée du compilateur. Ceci etant vu, l'utilisation d'une telle liaison est fort simple car, grâce à la directive #use RS232, le compilateur C de CCS (mais tous les bons compilateurs disposent d'une possibilité similaire) se charge de la programmation des registres internes de l'UART en fonction des paramètres de la liaison que vous avez spécifiés. Une fois cette directive fournie au préprocesseur du compilateur, les fonctions d'entrée/sortie standard du langage C sont automatiquement a même de faire appel à la liaison série ainsi définie sans autre intervention de votre part.
En voici un exemple très simple avec le schéma présenté figure 3.1. Comme vous pouvez le deviner a son analyse, il a pour simple fonction d'afficher sur les LED reliées au port B le code ASCII des caractères reçus sur sa liaison serie asynchrone. Et, pour etre sûr que vous ne faites pas de fautes de frappe, il renvoie systematiquement sur cette même liaison tous les caractères reçus. Malgré sa simplicité, va nous permettre d'étudier deux modes d'utilisation des entrées/sorties via une telle liaison. Celui qui vient immédiatement à l'esprit est celui que l'on appelle le mode programmé, dans lequel le processeur attend l'arrivé d'un caractère pour déclencher une action. Comme vous pouvez le constater à l'examen du listing 3, c'est le plus facile à écrire.
Ce programme n'appelle quasiment pas de commentaire puisqu'il utilise les fonctions

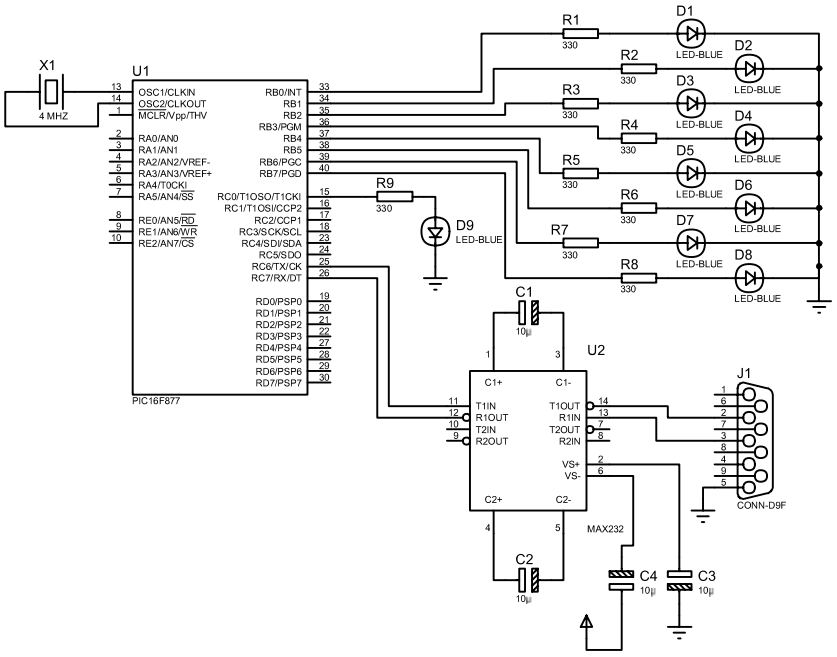

FIG. 3.1 – Montage de test pour les programmes de gestion de la liaison série asynchrone

```
//********************Listing 3*****************************//
#include  <16F877.h>
#use  delay(clock=4000000)
#fuses   NOWDT,XT,PUT,NOPROTECT
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7, bits =8)
void main()
{char car;
output_b(0);
while(true)
{
car = getchar();
output_b(car);
printf   ("\n\r Caractère reçu   :   %c",car);
}
}
//******************fin du Listing*************************//
```

standards getchar et printf du langage C pour traiter les caractères entrants et sortants via la liaison série. Cette dernière a été définie au moyen de la ligne
#use rs232<baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8) Qui se comprend d'elle-même dès lors que l'on sait ce qu'est une liaison série asynchrone. Hélas, ce mode de travail est loin d'etre le plus performant. En effet, dès que le processeur est entré dans la boucle d'attente d'un caractère, il ne fait plus rien d'autre tant que celui-ci n'est pas reçu. Ce n'est pas necessairement gênant pour certaines applications mais c'est rédhibitoire

pour d'autres. Une bien meilleure solution consiste à exploiter la possibilité qu'a l'UART de générer une interruption dès qu'il a reçu un caractère. Le processeur peut alors vaquer à ses occupations, la réception des caractères ayant lieu " automatiquement " au fur et à mesure de l'arrivee de ces derniers. La figure 3.1 va encore être utilisée à titre d'exemple mais, cette fois-ci, la LED connectee au port C0 va être utilisee. En effet, notre programme principal va avoir pour fonction de faire clignoter cette LED, tout en continuant bien sûr a afficher le code ASCII de tout caractère reçu, via sa liaison série, sur les LED reliées au port B. Le listing 4, nécessaire à un tel comportement reste cependant fort simple.

```
//*********************************Listing 4***************//
#include  <16F877.h>
#use  delay(clock=4000000)
#fuses   NOWDT,XT,PUT,NOPROTECT
#use  rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)
char  car;
#int_RDA
RDA_isr()
{
   car  =  getchar();
   output_b(car);
   printf   ("\n\r  Caractère  reçu   :   %c",car);
}
void main()
{
   output_b(0);
   enable_interrupts(INT_RDA);
   enable_interrupts(GLOBAL);
   while(true)
         {
         output_bit(PIN_C0,0);
         delay_ms(250);
         output_bit(PIN_C0,1);
         delay_ms(250);
         }
}
//***********************fin du listing*****************//
```

Comme vous pouvez le constater, c'est le programme de traitement de l'interruption de réception de caractère par l'UART qui se charge tout a la fois de son affichage sur les LED et de son renvoi sur la liaison série, tandis que le programme principal fait clignoter la LED reliée a C0 au moyen d'une boucle sans fin. il suffit donc de remplacer dans cet exemple, les quatre lignes correspondantes par celles que doit réellement accomplir le programme principal de votre application pour que le tour soit joué.

3.3 Travail demandé

1. Compiler le programme avec PCW par la suite vérifier son bon fonctionnement à l'aide de la simulation par ISIS (pour simuler la liaison RS232 utiliser le terminal virtuel).
2. expliquer puis commenter le programme présenté dans les deux listing.
3. selon vous quel est le meilleur type de listing qui sera appliquer dans un programme quelconque qui exploite la liaison série asynchrone.