M2P SET/SECI

TP Mise en œuvre d'une communication série asynchrone

Le but de ce TP est la mise en œuvre d'une communication série asynchrone utilisant des automates et une synchronisation à base de temporisateurs matériels (Timers). Les graphes des automates émetteur et récepteur sont donnés en annexes.
Nous utiliserons un microcontrôleur de la famille 80C51 (plus précisément le 80C592). Des explications sur les interfaces utilisateur et l'environnement logiciel de conception KEIL, compilation et debug seront données en séance de TP.
Lors de la création d'un projet, celui-ci doit regrouper l'ensemble des sources, les librairies utilisées ainsi qu'une librairie fondamentale notée " STARTUPFYT.LIB" (spécifique pour le module microcontrôleur de PHYTEC). Cette dernière assure l'amorçage du programme principal (main()) après avoir initialisé toutes les variables globales ainsi que la pile.
Le circuit CPLD de la carte de TP interface les périphériques au microcontrôleur de sorte à voir ces périphériques dans le plan mémoire XDATA (zone mémoire externe) du microcontrôleur.
En langage C étendu pour les microcontrôleurs :

- Le mot clé ***xdata*** précédant la déclaration d'une variable indique que la variable sera implantée dans l'espace XDATA du microcontrôleur.
- La déclaration "***xdata int v _at_ 0xFD00;***" précise que la variable v est allouée dans l'espace xdata plus précisément à partir de l'adresse FD00h. la variable v occupera 2 octets car le ***int*** est codé sur 16 bits sur ce microcontrôleur.
- Un tableau de constante devra être précédé du mot clé ***code*** pour éviter d'allouer de la mémoire RAM pour une donnée qui ne changera jamais.

Pour vous faciliter l 'écriture de vos programmes sous Keil, une librairie libtp2.lib est à votre disposition. Cette librairie contient les fonctions detect_touche() et decode_touche(), ainsi que les routines pour le contrôle de l'afficheur LCD.
Pour utiliser cette librairie, il faut ajouter #include "libtp2.h" dans l'entête de votre programme et libtp2.lib (ou libtp2.obj) dans votre projet. Voir le fichier libtp2.h pour les définitions.
Remarque : pour afficher un nombre, il faut d'abord le convertir en une chaîne avec sprintf() et afficher la chaîne avec print_lcd(chaine).

Interfaçage aux périphériques et mise en ouvre des interruptions :

1. On souhaite réaliser un chenillard. Le principe de la programmation repose sur une table de motifs (tableau de constantes) à faire apparaître sur les LEDs du "barre-graph" de la maquette de TP de manière à donner l'illusion d'un mouvement. On utilisera les 10 LEDs du barre-graph (codé sur un mot de 10 bits).
   Ecrire un programme permettant de faire défiler des motifs graphiques sur le "barregraph". Afin de distinguer les différents motifs sur les LEDs chaque motif doit être maintenu pendant un temps d'attente. Pour cela faire appel à un sous-programme TEMPO qui utilisera une fonction bloquante puis une fonction non bloquante.

2. En gardant l'ossature du programme précédent, utilisez une touche du clavier pour commander le sens de rotation du chenillard. On fera défiler le contenu du tableau (des motifs) dans un sens ou dans l'autre sens si l'on appuie sur la touche du clavier. On commencera par redéfinir la fonction chenillard (automate MOTIF) en y incluant un paramètre permettant de commander le sens du défilement des motifs. On définira également, un automate pour la gestion d'une touche du clavier. Concernant ce dernier automate (clavier), l'appui sur la touche du clavier ne devra être pris en compte qu'une seule fois jusqu'au relâchement de cette touche. La fonction automate du clavier et la fonction de décodage sont données dans la librairie LIBTP2.LIB.

3. Sur la base du programme de la question 1, remplacez la temporisation logicielle par une temporisation du TIMER 0 (Timer interne du 80C592 Fréquence d'horloge de base 16MHz). Nous utiliserons le Mode 2 de fonctionnement : mode à rechargement automatique et comptage sur 8 bits. Dans ce mode, un registre 8 bits (TL0) sert de compteur, à chaque fois qu'il passe de 255 à 0 (débordement) un Flag TF0 (dans le registre d'état du Timer) est généré. Le registre TH0 contient la valeur à recharger. Si le bit d'autorisation du Timer0 est mis à 1 (ET0=1), une interruption numéro 1 est déclenchée, forçant un chargement du vecteur d'interruption depuis l'adresse 000BH. Ensuite, un branchement vers la routine de traitement de l'interruption est effectué.

Procédure de mise en route de l'interruption Timer0 :

```
TMOD=TMOD | 2;   /* Mode 2 pour Timer0 */
TH0=0;           /*tampon pour période maximum ~192µs car quartz µC à 16MHz */
TL0=0;           /* Compteur mis à 0*/
ET0=1;           /*  validation de l'interruption Timer0 bit 1 du registre IE */
EA=1;            /*  autorisation globale des interruptions bit 7 du registre IE */
TR0=1;           /*  mise en route du Timer0 */
```

Pour avoir un délai d'attente suffisant, il est nécessaire de créer un Timer logique, sur la base du Timer physique, qui s'incrémentera à chaque interruption et produira un Flag logique. Ce Flag logique est une variable qui sera mise à 1 par l'interruption TIMER lorsque le Timer logique déborde. Ce Flag sera utilisé par le programme principal pour activer l'affichage du motif sur le Barregraph. Le temps maximum entre 2 interruptions matérielles du Timer est de 192µs.

• Combien d'interruptions matérielles du Timer sont nécessaires pour produire un flag logiciel indiquant une période de 250ms (ordre de grandeur).
• Vérifiez avec un oscilloscope la période de mise à 1 du Flag logique.

Mise en œuvre d'une communication série asynchrone:

Une communication asynchrone ne pourra être assurée que par une entente préalable de la vitesse de transmission, côtés émetteur et récepteur et de la définition d'un protocole de synchronisation. Nous utiliserons une vitesse de transfert de 372bits/s. En effet, le quartz du microprocesseur (Qrtz=16,0Mhz) sera utilisé comme fréquence de base pour obtenir une telle vitesse de transmission. Le Timer0 sera obligatoirement utilisé afin d'assurer la stabilité de la période pour chaque bit de la trame émise.

Ce Timer0 sera programmé en mode 2 avec une fréquence de base de Qrtz/(256*12)=5,20833333kHz. Cette fréquence de base est assimilée à la fréquence de base de l'automate de contrôle.
Il est important d'avoir une fréquence de base supérieure à la vitesse de transmission (au moins 10 fois supérieure) afin d'assurer un meilleur échantillonnage de la ligne de transmission. Ceci justifie la vitesse de transmission évoquée (372bits/s = $\frac{\text{la Fréquence du Timer0}}{14}$).

La figure 1 présente le chronogramme du signal émis sur la ligne. Il est important de respecter le point d'échantillonnage sur le signal reçu. Nous fixons ce point au centre de chaque bit de la trame. Une trame sera composée d'un bit de START, des 8 bits de la donnée et terminée par un bit de STOP.

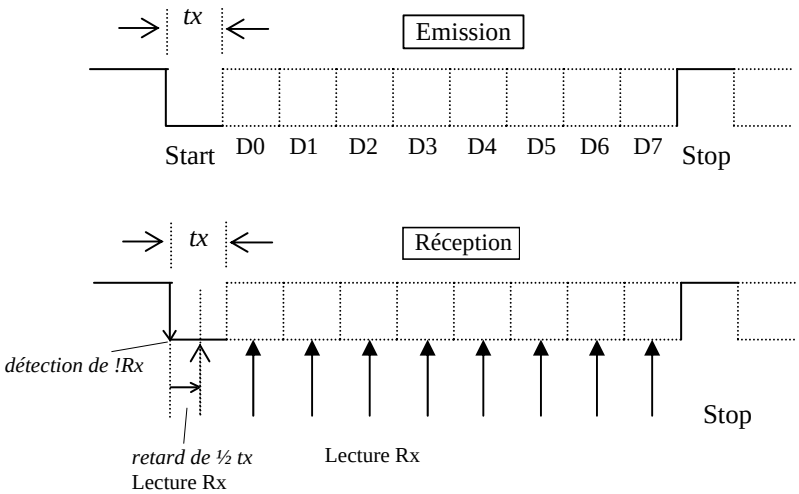


Figure 1 : chronogramme du signal vu en émission et par le récepteur

Lors de la réception, la détection du bit de START sera synchrone (la fréquence de base sera l'horloge de l'automate de réception).
Les automates de l'émetteur et du récepteur seront implantés dans la routine d'interruption du TIMER0. Cela permettra de dissocier le calcul (effectué dans le main) de la logique de transmission qui est très fortement synchronisé. La routine d'interruption sera la suivante :

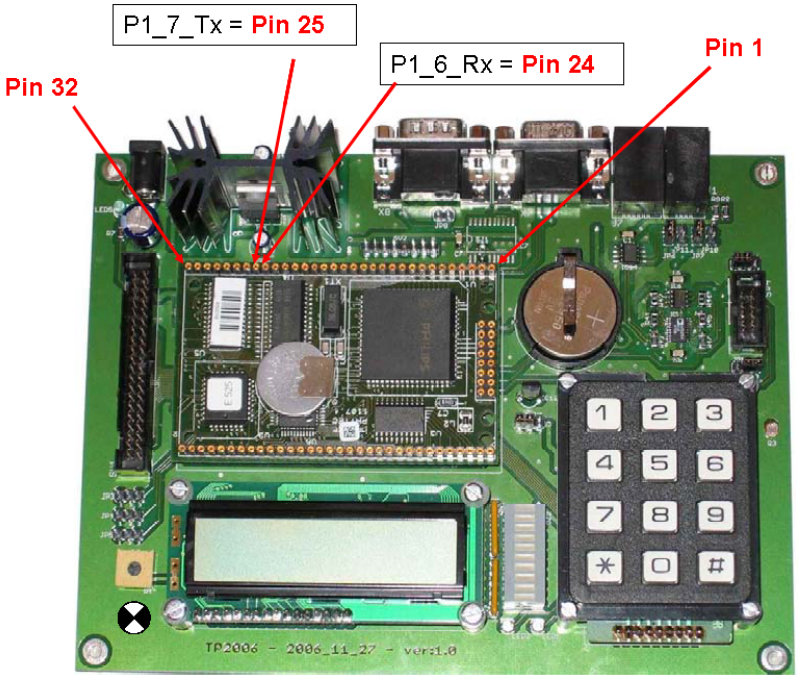
```
sbit broche_Tx = 0x97;      /* broche P1.7 pour l'émission */
sbit broche_Rx = 0x96;      /* broche P1.6 pour la réception */
bit Tx,Rx;                  /* bits mémoire pour échantillonner les broches */

TIMER0() interrupt 1  {
        broche_Tx=Tx;
        Rx= broche_Rx;
        Automate_emission();
        Automate_reception();
}
```

L'échantillonnage des broches en entrée de l'interruption permet de garantir une stabilité de la largeur de chaque bit émis. En effet, les deux automates étant bornés par un temps MIN et MAX de calcul, la modification des broches pendant ou après l'automate crée un décalage instable de la largeur de chaque bit.

1) Ecrire le code de l'automate émetteur et vérifier la trame série émise sur un oscilloscope. Il est important d'avoir un signal de synchronisation entre le programme principal qui place les caractères à émettre et l'automate (placé interruption) qui se charge du transfert effectif. Pour cela un signal TRANSMIT (variable de type bit) sera utilisée pour cette synchronisation. L'émetteur considère qu'il dispose d'un caractère à émettre lorsque la variable TRANSMIT passe à 1. Le programme principal (le main) ne pourra envoyer de caractère tant que ce bit est à 1. Lorsque le caractère en cours d'émission sera totalement envoyé, le bit TRANSMIT sera positionné à 0 par l'automate d'émission pour indiquer au programme principal qu'il est disponible pour envoyer un caractère suivant.

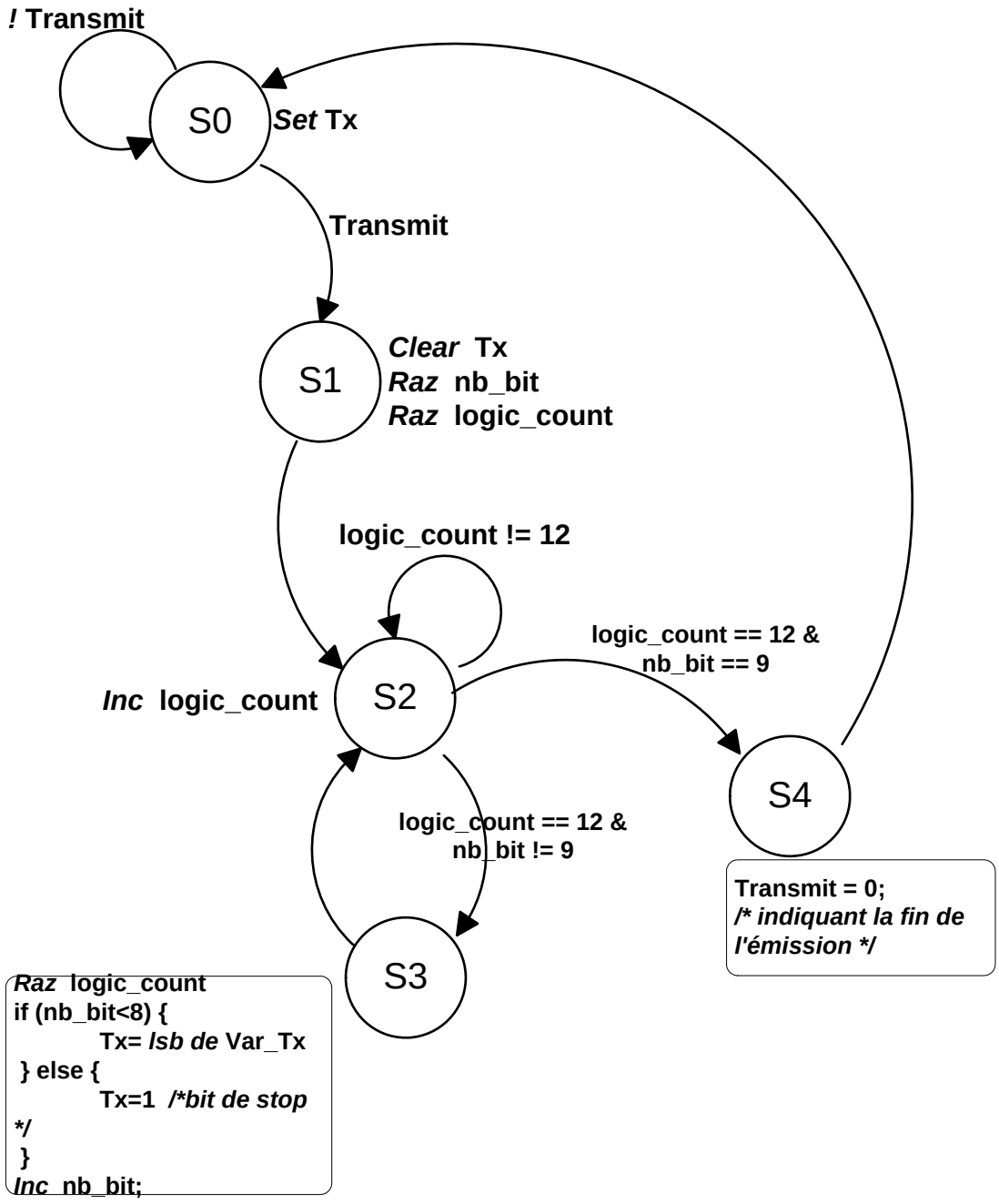
2) Ecrire le code de l'automate récepteur et vérifier son fonctionnement en vous reliant avec l'émetteur d'un autre binôme. Il est, dans ce cas également, important de prévoir un bit de synchronisation entre l'automate récepteur et la partie du programme principal qui récupère et affiche le caractère reçu. Une variable RECEIVED de type bit sera utilisée à cet effet. Lorsque le récepteur récupère un caractère de la ligne, il vérifie que le bit RECEIVED est à 0. Enfin il écrit la valeur du caractère reçu dans une variable d'échange avec le programme principal. Lorsque le programme principal trouve le bit RECEIVED à 1, il transfert le caractère reçu dans la variable d'échange et remet le bit RECEIVED à 0. Cette dernière opération indiquera à l'automate récepteur qu'un autre caractère peut être transféré dans la variable d'échange.

3) Croisez les fils TX et RX avec les autres groupes d'étudiants et vérifiez le bon fonctionnement des deux directions. On transmettra un caractère du clavier (voir les fonctions detect_clavier et decode_clavier de la librairie libtp.lib dont le header libtp.h est fourni et commenté) et on affichera le caractère reçu (envoyé par l'autre carte) sur l'afficheur LCD.



Compteur logic de temps: logic_count
Compteur nombre de bits : nb_bit
Tx : donnée du bit émission
Transmit: commande de l'émission
Var_Tx: registre d'émission

Automate émetteur



Compteur de bits: **nb_bit**
Compteur de latence: **logic_count**
Rx: ***signal d'entrée bit P1_6***
Var_Rx: ***registre d'entrée 8bits (Shift Right)***

Automate récepteur