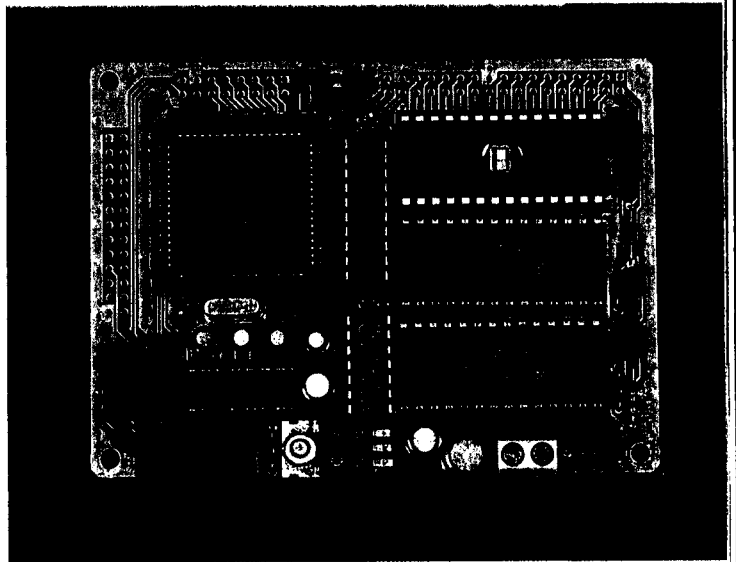


GESTION DE PLUSIEURS INTERFACES SERIE

Les interfaces série asynchrones sont très utilisées dans de nombreuses applications : monnayeurs, lecteurs de codes à barres, minitel, réseaux de contrôle d'automates, modems. Tous utilisent cette interface pour communiquer avec le calculateur.

Les microcontrôleurs modernes

intègrent généralement une interface série asynchrone. Malheureusement, la multiplicité des périphériques à contrôler nécessite souvent plusieurs interfaces série.



Ainsi, par exemple, on désirera commander un minitel et un lecteur de codes barre ou bien un modem et un monnayeur, etc. La solution généralement adoptée est l'acquisition d'un boîtier périphérique qui gèrera les interfaces série supplémentaires. Dans le cas où le débit binaire des périphériques est lent (1200 ou 2400 bauds) il est possible de gérer entièrement par programme plusieurs interfaces série. On pourra ainsi avec un simple 80C51 gérer trois interfaces série (avec l'interface hardware) simultanément.

Le programme utilisera une interruption périodique dont la période sera égale au quart de la durée d'un bit. A 1200 bauds, la durée d'un bit est de 833 µs, un timer (le timer 1) sera donc utilisé pour générer une interruption périodique toutes les 208,33 µs. Cette interruption sera l'interruption la plus prioritaire de l'application (registre IP). On pourra d'ailleurs se servir de cette interruption pour gérer les bases de temps utiles dans le reste du programme. Si le quartz utilisé est un 11,0592 MHz, le temps entre deux interruptions sera un nombre entier de cycle, en l'occurrence 192 cycles, ce qui nous

```
RX_PIN EQU 090H ; P1.0 ligne de reception de donnée (RxD soft)
TX_PIN EQU 091H ; P1.1 ligne de sortie de donnée (TxD soft)
MSB EQU 0
LSB EQU 1
RSEG SEGMENT_DATA
RX_CPT: DS 1 ; automate de la rs RXD
TX_CPT: DS 1 ; automate de la rs TXD
TX_BIT: DS 1 ; compteur de bits TX
RS_ACC: DS 1 ; accumulateur de bit
RS_RX: DS 1 ; donnée recue
RS_TX: DS 1 ; donnée émise
RSEG SEGMENT_BIT
SFT_RI: DBIT 1 ; flag donnée disponible dans RS_RX
SFT_TI: DBIT 1 ; flag interface d'émission libre
```

```
; PROGRAMME D'INTERRUPTION
; l'interruption est générée par le timer0 toute les 192 cycles soit tous les
; 1/4 bits pour une vitesse de transmission de 1200Baud (avec un Xtal a 11.059)
```

```
ITEMPS: PUSH PSW ; 2 cycles + 9 cycles de latence max
        PUSH ACC ; 2 cycles
```

```
; gestion de la RS soft en emission (max 21 cycles, 2 cycles si pas d'émission)
```

```
RSEMIT: JB SFT_TI,RSTEND
        MOV A,TX_CPT
        ANL A,#00001111B ; isole les ticks
        JZ RSEMIE ; exécute l'état courant
        DEC TX_CPT ; décremente le compteur de ticks
        SJMP RSTEND ; 9 cycles depuis RSEMIT
```

```

; tous les 4 ticks
RSEMIE: MOV A,TX_CPT ; prend l'état de l'automate
        JNZ TXISAC ; 9 cycles depuis RSEMIT
; etat 0 : preparation de l'emission
TXSTAR: CLR TX_PIN ; bit de start (TX_PIN à 0)
        MOV TX_CPT,#13H
        MOV TX_BIT,#8
        SJMP RSTEND ; 16 cycles depuis RSEMIT
TXISAC: CJNE A,#10H,ISTXST
; etat 1 : decalage des huit bits
TXACCU: MOV A,RS_TX
        RRC A ; décale le buffer d'émission
        MOV RS_TX,A
        MOV TX_PIN,C ; sort le bit sur la ligne d'émission
        DJNZ TX_BIT,TXACCU
        MOV TX_CPT,#23H ; état suivant si dernier bit
        SJMP RSTEND
TXACCO: MOV TX_CPT,#13H ; même état si il reste encore de bits
        SJMP RSTEND ; 22 cycles depuis RSEMIT
ISTXST: CJNE A,#20H,TXEND
; etat 2 : envoie du bit de stop
TXSTOP: SETB TX_PIN
        MOV TX_CPT,#33H
        SJMP RSTEND ; 18 cycles depuis RSEMIT
; etat 3 : fin de l'emission
TXEND: MOV TX_CPT,#0 ; automate à 0 pour prochaine émission
        SETB SFT_TI ; 15 cycles depuis RSEMIT
RSTEND:
;
; gestion de la RS soft en reception (max 21 cycles, 9 cycles si attente)
RSSOFT: MOV A,RX_CPT
        ANL A,#00001111B ; isole le compteur de ticks
        JZ RCV
        DEC RX_CPT
        SJMP ENDRS ; 7 cycles depuis RSSOFT
RCV: MOV A,RX_CPT
        JNZ RCVST1
; etat 0 : attend le bit de start
WTSTAR: JB RX_PIN,ENDRS ; si pas de start, attend (9 cycles)
        MOV RX_CPT,#11H ; sinon automate état suivant
        SJMP ENDRS ; (prochain test après le temps de 1/2 bit)
RCVST1: CJNE A,#10H,RCVST2
; etat 1 : est-ce un bit de start ?
ISSTAR: JB RX_PIN,ISSTA0 ; faux bit de start, réinitialise
        MOV RS_ACC,#80H ; initialise l'accumulateur de bits
        MOV RX_CPT,#23H ; état suivant
        SJMP ENDRS
ISSTA0: MOV RX_CPT,#0 ; faux bit de start, automate à 0
        SJMP ENDRS
RCVST2: CJNE A,#20H,ISSTOP
; etat 2 : accumule les 8 bits
ACCUBT: MOV C,RX_PIN ; charge la carry avec la ligne rs
        MOV A,RS_ACC
        RRC A ; accumule le bit
        MOV RS_ACC,A
        JNC ACCUB0
        MOV RX_CPT,#33H ; si dernier bit, etat suivant
        SJMP ENDRS
ACCUB0: MOV RX_CPT,#23H ; sinon, même état
        SJMP ENDRS ; 21 cycles depuis RSSOFT
; etat 3 : est-ce un bit de stop ?
ISSTOP: MOV RX_CPT,#0
        JNB RX_PIN,ENDRS ; pas de stop, erreur, sort
        MOV RS_RX,RS_ACC ; transfere l'accumulateur de bits
        SETB SFT_RI ; marque donnee recue
ENDRS: POP ACC ; 2 cycles
        POP PSW ; 2 cycles
        RETI ; 2 cycles

```

; lecture et écriture d'un caractère A depuis l'interface soft

```

GET_SF: JNB SFT_RI,GET_SF
        CLR SFT_RI ; remet le flag donnée reçu a 0
        MOV A,RS_RX ; et prend la donnee
        RET

PUT_SF: JNB SFT_TI,PUT_SF
        MOV RS_TX,A ; transfere la donnée
        CLR SFT_TI ; et démarre la transmission
        RET

```

donne un temps de traitement maximum de l'interruption de 192 - 11 = 181 cycles ; les routines de réception et d'émission prennent au maximum 21 et 22 cycles, ce qui laisse du temps pour d'autres traitements. La routine de réception de caractères est gérée sur la base d'un automate à 4 états :

- le premier état est l'attente d'un bit de start.
- le deuxième état est la vérification du bit de start.
- le troisième état est la saisie des huit bits envoyés par le périphérique.
- Le quatrième état est la vérification du bit de stop.

A la fin de chaque état si le déroulement est normal, l'automate passe dans l'état suivant.

Deux variables contrôlent l'automate (regroupées dans un seul octet). La première contient le numéro de l'état courant, la deuxième est un compteur de temps qui permet de définir à quel moment un état donné est actif.

- Etat 0 : à chaque interruption, le programme teste l'état de la ligne de réception de données, si cette ligne est haute, pas de changement d'état, sortie de l'interruption, si la ligne est basse, charger le compteur de ticks avec une valeur correspondant à un demi-bit et la variable d'état à 1 puis sort.

- Etat 1 : si le compteur n'est pas arrivé à échéance, sortir, sinon, tester la ligne de réception de données. Si la ligne est toujours basse, charger le compteur de ticks avec la valeur correspondante à un bit et faire progresser l'état de l'automate à l'état 2.

Si la ligne est haute, faux bit de start, revenir à l'état 0.

- Etat 2 : si le compteur n'est pas arrivé à échéance, sortir sinon accumuler la valeur lue sur la ligne de réception, charger le compteur de ticks avec la valeur correspondante à un bit. Si le dernier bit est chargé, passer à l'état suivant, sinon rester dans le même état.

- Etat 3 : si le compteur n'est pas arrivé à échéance, sortir, sinon, tester la ligne de données à l'état haut. Si c'est le cas (bit de stop) transférer la valeur accumulée dans le registre de données, positionner le flag donnée prête puis revenir à l'état 0.

Le programme du listing 1 se comporte exactement comme l'interface hardware. Pour lire un caractère, l'utilisateur doit tester le bit SFT_RI (pendant du bit RI).

Si ce bit est à un, la donnée est prête dans le registre RS_RX.

Une routine similaire associée au registre RS_TX et au bit SFT_TI gère aussi l'émission d'un caractère.

Le listing complet ainsi qu'un exécutable gérant une file de réception de caractères sont disponibles sur le serveur ERP.

J.L. Vern