CH-1092 Belmont, Switzerland
info@didel.com, www.didel.com



# PIC*Génial* - Microcontrôleur PIC 16F84/16F870

**Introduction à la programmation avec le PIC*Génial* - suite**

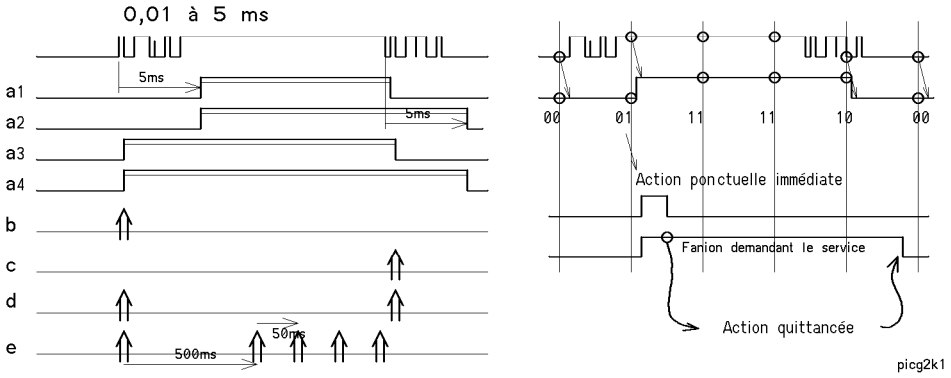
## 6. Touches, conversion A/D et transfers série

### 6.1. Rebonds

Tout contact mécanique (touche, relais) a des rebonds pendant 1 à 5 ms. Pour les supprimer, il suffit de lire les touches et claviers avec une période de 5 ms au moins. Une période supérieure à 50 ms peut faire perdre des touches frappe "en vol" sur un clavier complet).

Dans le cas d´une touche simple, il y a 5 cas à distinguer, avec des sous-cas:

a) Action tant que la touche est pressée (variantes a1..a4 selon position du retard)
b) Action unique quand on presse
c) Action unique quand on relâche
d) Action quand on presse et relâche (exceptionnel)
5) Action quand on presse, puis répétition à fréquence fixe après un retard donné (section suivante).




picg2k1

*Fig. 1 Rebonds et signal mis en forme par le processeur*

Le premier cas est simple, on filtre le signal en sous-échantillonnant:

```
    .... toutes les 5 ms environ
    TestSkip,BC   PortA:#bSw
    Jump      Action
    TestSkip,BS   PortA:#bSw
    Jump      Inaction
```

Le 2e et 3e cas peuvent se traiter de 2 facon différente. Il faut mémoriser l´état précédent dans Mode:#bCopySw. En testant l´état 01 (10 dans le cas C) lorsque le signal est échantillonné, on peut déclencher immédiatement l´action correspondante, par exemple incrémenter un compteur comme dans l´exemple ci-dessous, qui incrémente le port B à chaque action.

```
; Initialisation (portB en sortie)
        Clr       Cnt
; Toutes les 5 ms
        TestSkip,BC   PortA:#bSw      ; On lit, état 0 sans intérêt
        TestSkip,BC   Mode:#bSwCopy ; Etat 1 nouveau et état 0 ancien
        Jump      F$
        Set       Mode:#bSwCopy
        Inc       Cnt         ; Action
        (jump suite inutile si l´action ext brève)
  F$:   TestSkip,BS   PortA:#bSw
        Clr       Mode:#bSwCopy
suite:  ...
```

S´il y a plusieurs touches à surveiller, la situation reste simple si une seule touche est pressée à la fois. Si plusieurs touches sont pressées à la fois, et si une nouvelle touche doit être décodée, une comparaison entre l´ancienne et la nouvelle valeur lue permet de détecter quelle touche est pressée.

Un clavier complet est surveillé en balayant les rangées de touches avec un décodeur, un compteur en anneau et/ou un multiplexeur.

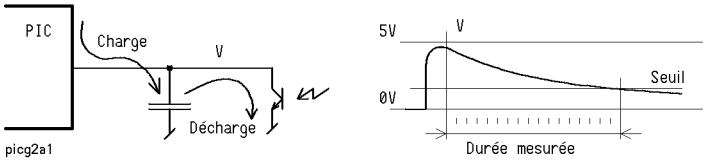
## 6.2. Touches répétitives

Un clavier avec touches répétitives est un joli exercice. Une fois qu'une touche est repérée pressée, il faut envoyer son code et mettre en route un retard de 0.5 seconde environ. Après ce temps, on envoie le code à nouveau, et on répète le code 20 fois par seconde environ, jusqu'à ce que la touche soit relâchée.

## 6.3. Conversion A/D

Les processeurs bas de gamme de la série PIC n'ont pas de canaux A/D. On peut s'en passer en utilisant un convertisseur extérieur avec une interface série, ou en effectuant une conversion temporelle avec le processeur lui-même.

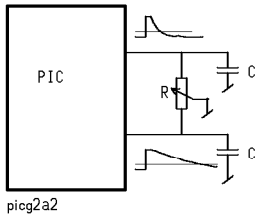
La figure 2 explique la mesure d'une résistance variable, par exemple celle d'une photodiode qui dépends de l'intensité lumineuse reçue. La durée de la mesure, pour atteindre le seuil, dépend de la valeur de la capacité et du courant dans la photodiode. Etant donné que le seuil est vers 1.4V, le temps de décharge de 5V à 1.4V est plus long que le temps de charge de 0 à 1,4V, il peut sembler avantageux de mesurer le temps de décharge. Mais la précision est plus faible puisque la courbe de décharge est plus "tangente". Si possible, il faut utiliser l'entrée RA5/T0Clk du 16F84, qui a une entrée en bascule de Schmitt avec des seuils à 1,4 et 3,8V. Un signal de charge qui passe de 0 à 1 est alors préférable. Dans la figure figure 2, le programme prépare le port en sortie, écrit un "1", commute en entrée et mesure le temps jusqu'à ce que l'entrée passe à zéro.




picg2a1

*Fig. 2 Mesure d'une résistance variable*

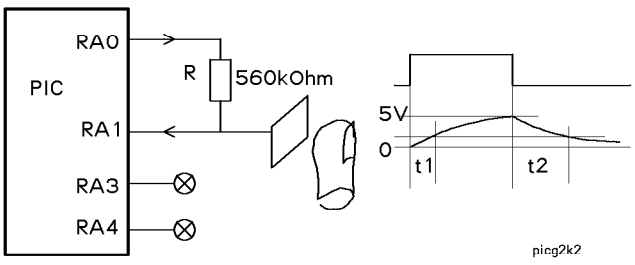
Pour mesurer un potentiomètre, dans un manche à balais par exemple, le schéma de la figure ci-contre est préférable. En fin de course, la mesure est peu précise. En calculant la différence des deux mesures, on a une bonne linéarité.




picg2a2

*Fig. 3 Mesure de la position d'un potentiomètre*

```
Clr         PortA:#0        ; Mesure du temps de d
TestSkip,BC PortA:#bIn      ;Suite de tests
Inc         Cnt                 ; en attentant le se
TestSkip,BC PortA:#bIn
Inc         Cnt
TestSkip,BC PortA:#bIn      ; 3 µs chaque fois
Inc         Cnt
TestSkip,BC PortA:#bIn
Inc         Cnt
TestSkip,BC PortA:#bIn
Inc         Cnt
TestSkip,BC PortA:#bIn
Inc         Cnt
TestSkip,BC PortA:#bIn
Inc         Cnt
TestSkip,BC PortA:#bIn
Inc         Cnt
TestSkip,BC PortA:#bIn
Inc         Cnt
TestSkip,BC PortA:#bIn
Move        Cnt,W           ;Affiche Cnt
Move        W,PortB
Jump        Loop
```




picg2k2

*Fig. 4 Touche capacitive*

```
Loop:   Clr     Cnt
        Set     PortA:#0  ; On charge pendant 6 µs
        Move    #2,W
        Move    W,C1
   B$:  DecSkip,EQ C1
        Jump    B$
; suite programme colonne de droite
```

La solution du programme précédent permet de décoder une touche capacitive, formée d'un fil ou plaquette métallique. Si le doigt est présent, il faut environ 10 µs pour décharger la capacité, au lieu de 5µs lorsque le doigt n'est pas présent. Le programme

compte cette durée, et l'affiche sur le portB, ce qui permet de vérifier les deux valeurs, et juger si le fonctionnement est assez fiable (si ces solutions capacitives étaient fiables, on les verrait partout).

## 6.4. Conversion A/D sur le 17F870

Le 16F870 a 5 canaux analogiques avec une précision de 10 bits, On peut en mettre 0 à 5 en service, l'un des canaux pouvant jouer le rôle de référence de tension. Le principe est de configurer le registre AdCon1 dans l'initialisation, puis de dire quel canal on veut lire, attendre 20 $\mu$s (pour que l'entrée sélectionnée soit stable à lintérieur) et démarrer la conversion. Elle dure un certain temps, que l'on teste en en attendant que le bit GO repasse à zéro. Deux registres de 8 bits reçoivent le résultat, et on peut demander d'aligner le résultat 10 bits à droite ou à gauche (si on ne s'intéresse pas aux deux bits de poids faible). Les registres associés sont parfois en bank0 et parfois en bank1 et il faut en tenir compte (les macros définies dans la section 9 facilitent l'écriture). L'initialion et la lecture du canal AN0 (sur RA0) a donc l'allure suivante:

```
; Initialisation
  Bank0to1
        Move    #IniAdConLeft,W
        Move    W,AdCon1
  Bank1to0
; Routine qui rends dans W les poids forts
GetAD0: Move    #SelAd0,W
        Move    W,AdCon0
        Call    Del20     ; Attente 20 µs
        Set     AdCon0:#Go
 A$:    TestSkip,BC AdCon0:#Go
        Jump    A$
        Move    AdResH,W
        ; On ignore les poids faibles dans AdResL en bank1
        Ret
```

Pour définir les constantes IniADConLeft et SelAD0, il faut lire toutes les options dans la documentation de Microchip.

## 6.5. Conversion D/A

La conversion D/A se fait le plus efficacement avec un circuit série à 8 pattes comme le Max548. On peut aussi générer du PWM filtrer, ou câbler un réseau de quelques résistances si la précision est très faible.

## 6.6. Transferts série

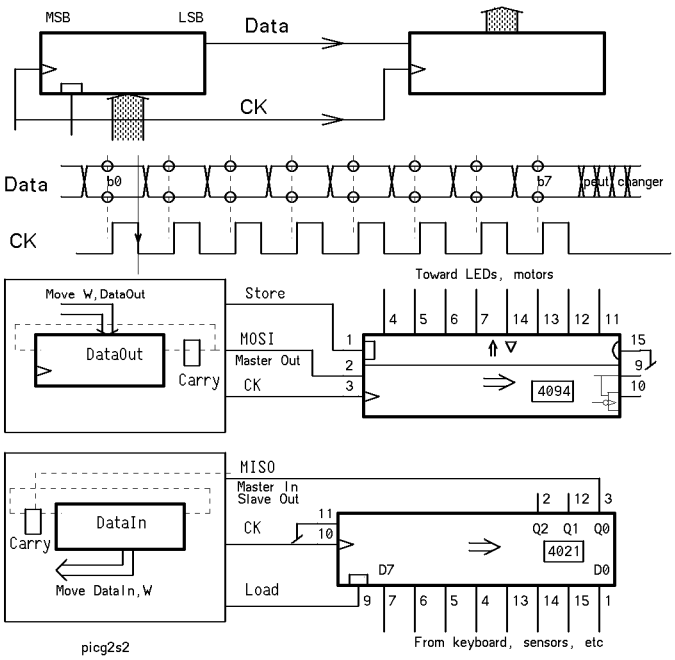
## 6.7. Principe

On utilise beaucoup les transferts série avec des petits microcontrôleurs, car les connecteurs et fils sont chers, et les vitesses relativement lentes. Le prix d'un PIC et d'un circuit capteur, convertisseur A/D, etc., est assez proportionnel au nombre de pattes du circuit. Il y a donc tout avantage à sérialiser lorsque les vitesses de transfert ne sont pas très grandes, ce qui est en général le cas quand de la mécanique (automate, robot mobile) ou l'homme sont concernés (capteurs de température, etc.).

Les transferts série de mots de 8 bits et plus peuvent se faire avec 1 à 4 fils, avec des contraintes temporelles plus serrées lorsque le nombre de fils est réduit. Il faut transférer un bit en écriture ou lecture, avec une horloge de synchronisation pour les bits et pour les mots. Ceci correspond aux quatre lignes du SPI (Serial Peripheral Interface). La ligne de donnée peut être unique et bidirectionnelle. La synchronisation mot peut se faire avec un silence, un compteur ou un encodage ($I^2C$). La synchronisation bit sans horloge nécessite un start bit et des timings précis (RS232). La modulation Manchester et les codes RC5 des télécommandes infrarouges sont d'autres solutions, faciles à gérér avec un PIC bien programmé.

## 6.8. Série simple

Envoyer 8 bits en série demande 8 coups d´horloge. Il faut bien définir ce que l´on appelle coup d´horloge (impulsion positive ou négative) et distinguer l´instant où l´information change (entre deux coups d´horloge) et l´instant où elle est échantillonnée. A la vitesse des microcontrôleurs, il n´y a pas de problème de timings. Le récepteur échantillonne l´information sur le front montant ou descendant. Adoptons des impulsions positives avec décalage au front descendant et un échantillonage au front montant (fig 13). Les instructions d´assignation et de test de bits facilitent la programmation. La rotation de l´information à travers le Carry est en général la solution la plus efficace. Les routines de transfert en écriture et lecture s´écrivent:




picg2s2 *Fig. 5 Transferts série*

Program **Picgs2.asm** Routines série

Routine **Write** Transfer 8 bit serially

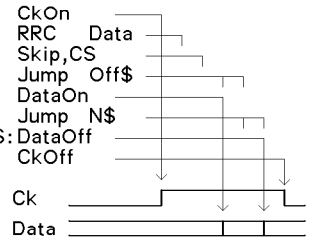
```
   in:  DataOut, transferred LSB first
   mod: DataOut, C1
Write:
        Move        #8,W
        Move        W,C1
L$:
        RRC         DataOut
        Skip,CC
        Set     PortA:#bData
        Skip,CS
        Clr     PortA:#bData
        Set     PortA:#bCk
        Clr     PortA:#bCk
        DecSkip,EQ C1
        Jump        L$
        Set         PortA:#bStore
        Clr         PortA:#bStore
        Ret
```

Routine **Read** Get 8 bit (provide the clock)

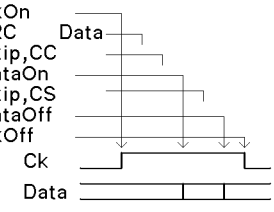
```
   out: DataIn read, transferred LSB first
   mod: DataIn, C1
Read:
        Move        #8,W
        Move        W,C1
        Set         PortA:#bLoad
        Clr         PortA:#bLoad
L$:
        ClrC                        ;*
        TestSkip,BC PortA:#bData    ;*
        SetC                        ;*
        RRC         DataIn
        CkOn
        CkOff
        DecSkip,EQ C1
        Jump        L$
        Ret

; Les trois instructions de la routine Read, marquées
; par un *, peuvent être remplacées par une seule
; (RRC PortA,W) si le bit d´entrée est câblé sur le
; bit 0 du port A (i.e. bData=0). Cette instruction
; décale le bit de poids faible de PortA dans le Carry.
```

```
bsf  PortA,bCK          CkOn
rrf  Data,1             RRC  Data
btffs Status,0          Skip,CS
goto Off$               Jump  Off$
bsf  PortA,bData        DataOn
goto N$                 Jump  N$
bcf  PortA,bData  Off$: DataOff
bcf  PortA,bCK    N$:   CkOff
```



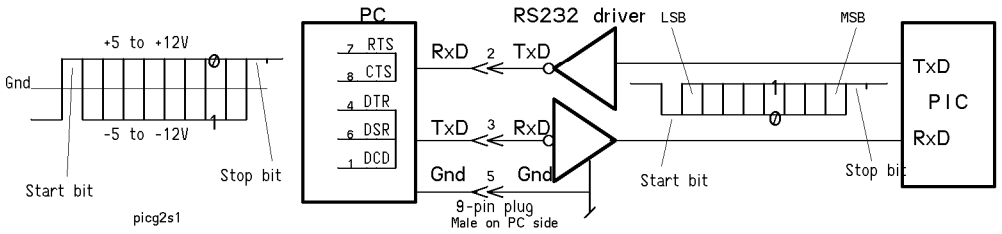

```
bsf  PortA,bCK   CkOn
rrf  Data,1      RRC  Data
btfsc SFR,0      Skip,CC
bsf  PortA,bData DataOn
btffs SFR,0      Skip,CS
bcf  PortA,bData DataOff
bcf  PortA,bCK   CkOff
```




picgs3 *Fig. 6 Timing pour un bit*

## 6.9. RS232

Dans le langage courant, RS232 caractérise l´interface série asynchrone disponible sur tous les PC. Les amplificateurs de ligne inversent le signal. Les signaux RTS et DTR sont connectés de façon à être compatible avec les protocoles "none" et "hardware" des programmes simulateurs de terminaux. Un PIC 16F84 à 4 MHz est facilement programmé pour travailler à 9600 bits/s. Si le PIC n´a pas un oscillateur à quartz garantissant la durée d´une $\mu$s par instruction, il faut calibrer cette durée et en tenir compte dans les boucles d´attente ou initialisations de timers.




picg2s1 *Fig. 7 Signaux série asynchrone*

La période entre bits est fixée par le "bit rate" (ne pas utiliser le terme Baud rate)

| | |
|---|---|
| 9600 b/s | 104,166 $\mu$s |
| 38400 b/s | 28,73 $\mu$s |

La précision doit être de 5% pour ne pas perdre le dernier bit transmis, qui est le bit de poids fort.

Le document www.didel.com/doc/Doc232.pdf explique les interfaces élecriques possibles entre le PIC et le PC. Le document www.didel.com/picg/DopiSer.pdf donne les routines pour le 16F84 et le 16F870..pdf

## 6.10. I²C

Le bus I²C de Philips, repris par Intel sous le nom SM-Bus, utilise 2 fils et permet d´adresse 127 registres répartis dans les unités branchées au bus. La vitesse de 100 kbit/s pose problème avec le PIC 16F84, 12C508 à 4 MHz qui n´ont pas d´interface I²C câblé comme les circuits 16C71. Il faut donc programmer les maîtres à une vitesse inférieure à 10 kb/s et programmer en synchrone avec un échantillonage à 100 $\mu$s.

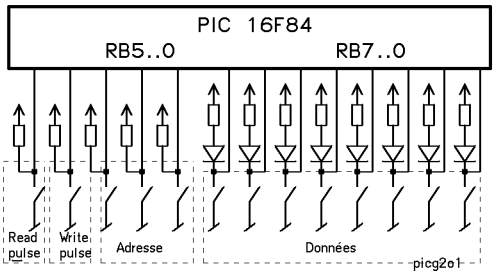
Le document www.didel.com/picg/DopicI2C.pdf détaille les macros et routines qui permettent de lire et écrire dans un esclave I2C, et donne comme exemple l´inteface avec un I/O expander PCF 8574 et une EEPROM 24C01..pdf

## 6.11. Programmation de l´E²PROM

L´E²PROM du 16F84/16F870 et d´autres PICs a 64 bytes et permet de mémoriser des paramètres ou un numéro de révision. On ne peut pas y stocker un programme. Le principe d´accès est de préparer un registre contenant l´adresse, et de transférer des commandes et les données. Les explications du fabricant sont claires.

Le programme suivant permet de tester, en supposant que des interrupteurs sont disponibles sur le port A pour donner l´adresse (en se limitant aux adresses 0..7) et les ordres d´écriture et lecture. Pour une autre application, on récupérera les routine ReadEe et WriteEe. Il n´y a pas besoin d´initialisation.

Des lampes et interrupteurs sont câblés sur le port B pour entrer/afficher les données (les interrupteurs dovent être ouverts lorsque le processeur affiche une donnée lue). La liaison au PC avec les routines série est un excellent exercice.




picg2o1 *Fig. 8 Câblage de test*

Program **Picgp1** Test E²PROM

```
; Valable seulement pour le 16F84
; Le programme écrit un mot préparé sur le portB
; et l´affiche sur le PortB
; (si tous les interrupteurs sont à 1)
.Proc     16F84
DebVar    = 16´20
RP0       = 5       ; sélection Bank1 dans registre Statu
EECon1    = 16´08   ; en Bank1   F84
 EEIF     = 4
 WrEn     = 2
 WR       = 1
 RD       = 0
EECon2    = 16´9    ; en Bank1   F84
```

Constant **Ports** PortA Ctrl et adr, PortB Data

```
DirA      = 2´11111          ; entrées
mAdr      = 2´111   ; seulement adresses 0..7
bWr       = 3       ; Poussoir Write sur RA3
bRd       = 4       ; Poussoir Read sur RA4
DirBout   = 0       ; en sortie pour affichage temporaire
DirBin    = 2´11111111       ; en entrée
```

Variables **Registres**

```
          .Loc      DebVar
C0:       .Blk.16   1
C1:       .Blk.16   1         ; Compteurs
C2:       .Blk.16   1
C3:       .Blk.16   1
.Loc      0
```

Program **Programme**

```
Deb:
          Move      #DirA,W
          Move      W,TrisA
          Move      #DirBin,W
          Move      W,TrisB
Loop:
; On attend que les 2 poussoirs soient à zéro
L:        TestSkip,BS   PortA:#bRd
          TestSkip,BC   PortA:#bWr
          Jump      L
; On attend l´action sur le poussoir Read ou Write
A:        TestSkip,BC   PortA:#bRd
          Jump      DoRead
          TestSkip,BC   PortA:#bWr
          Jump      DoWrite
          Jump      A
```

Module **DoWrite** Ecriture

```
DoWrite:
          Move      PortB,W   ; Est en entrée
          Move      W,EEData
          Move      PortA,W
          And       #mAdr,W
          Move      W,EEAdr
          Call      WriteEe
          Jump      Loop
```

Module **DoRead** Lecture

```
DoRead:
; On vérifie que les interrupteurs sur le PortB sont tous à 1
          IncSkip,EQ  PortB,W
          Jump        Erreur        ; On refusera d´écrire
          Move        #DirBout,W
          Move        W,TrisB       ; On commute en sortie
          Move        PortA,W
          And         #mAdr,W
          Move        W,EEAdr
          Call        ReadEe
          Move        EEData,W
          Move        W,PortB
; On affiche la valeur lue pendant 2 secondes avant de repa
          Move        #20,W         ; 2s
          Call        LongDelai
          Move        #DirBin,W
          Move        W,TrisB
          Jump        Loop
```

Module **Erreur** Tous les interrupteurs ne sont pas à un

```
Erreur:
          Not         PortB,W
          Move        W,TrisB       ; On met en sortie les
          Move        #3,W
          Move        W,C0          ; On veut clignoter 3 fo
 R:       Clr         PortB         ; On met les sorties per
          Move        #2,W
          Call        LongDelai
          Dec         PortB         ; On active les sorties
          Move        #2,W
          Call        LongDelai
          DecSkip,EQ  C0
          Jump        R
          Jump        Loop
```

Routine **ReadEe** Lit le mot à l´adresse dans EEADR et rend le contenu dans W

```
   in:   EEAdr
   out:  EEData
ReadEe:             Set           Status:#RP0   ; Bank 1
          Set       EECon1:#RD    ; Initialise la lecture
          Clr       Status:#RP0   ; Bank 0
          Ret
```

Routine **WriteEe** Ecrit le contenu de W dans EEADR

```
   in:   EEAdr   EEData
WriteEe:            Set           Status:#RP0   ; Bank 1
          ; Clr     IntCon:#GIE   Empêche les interruptions
          Set       EECon1:#WrEn  ; Autorise l´écriture
          Move      #16´55,W      ; Indispensable
          Move      W,EECon2      ; Indispensable  ; truc
          Move      #16´AA,W      ; Indispensable
          Move      W,EECon2      ; Indispensable
          Set       EECon1:#WR    ; Commence le cycle d´
          ; Set     IntCon:#GIE   Autorise les interruptions
  W$:TestSkip,BC    EECon1:#EEIF
          Jump      W$            ; Attente env 10ms cycl
          Clr       EECon1        ; Fin du cycle EEIF=0
          Clr       Status:#RP0   ; Bank 0
          Ret
; Ajuoter la routine LonDelai
LongDelai:
          Ret
.End
```

Dans le cas du 16F870, les routines sont très similaires, le seul changement étant que les registres EeCon1 et EeCon2 sont dans la bank3.

## 6.12. Accès dans la mémoire programme du 16F870

Le 16F870 a une possibilité très intéressante de pouvoir lire et écrire dans la mémoire programme, la flash. Comme les adresses sont 10 bits (2k de programmes) et les instructions 14 bits, un double registre d´adresse EeAdrH et EeAdr pointe dans la mémoire et un double registre EeDatH et EeData transfère la donnée pointée.

L´application principale est d´aller chercher un texte ou un morceau de musique en mémoire programme, avec une routine unique à laquelle on passe comme paramètre le

pointeur au texte, ce que ne permet pas le 16F84, qui n´a que la possibilité du RetMove. La partie de programme suivante montre comment lire un texte en page 0 (EeAdrH=0) avec les caractères codés dans les poids faibles seulement (on pourrait mettre deux caractères Ascii dans un mot de 14 bits).

**Fichier Picge1**

```
; dans le programme:
        Move    #Text1,W
        Move    W,PtText
        Call    GetText
        ....
Text1:          .Asciz      "S a l u t "  ; Il faut
```

Routine **GetText** **Envoie le texte pointé**

```
  in:  W adresse texte dans la même page
  mod:
GetText:
        Move    W,PtText
 L$:    Call    GetEe870
        Skip,NE
        Ret
        Call    SndSer      ; envoi en série
        Jump    L$
```

Routine **GetEe870** **Lit un car dans un .Asciz en flash**

```
; Chaque caractère est suivi d´un espace
  in:   PtText: pointeur dans le programme
  out:  W poids faible = caractère
  out:  EQ si car=0
GetEe870:
  Bank0to2
      Move      PtText,W      ; Postincrémenté
      Move      W,EeAdr
      Inc       PtText
  Bank2to3
      Set       EECON1:#EEPGD
      Set       EECON1:#RD    ; lecture
      Nop
      Nop
  Bank3to2
      Move      EEDATA,W      ; Le resultat de la lectu
      Move      W,SavW
  Bank2to0
      Ret
```