Apprendre à programmer avec le 16F877A - 4e partie

Exemples et exercices sous www.didel.com/pic/Cours877Exos-4.pdf
1ere partie sous www.didel.com/pic/Cours877.pdf
2e partie sous www.didel.com/pic/Cours877-2.pdf
3e partie sous www.didel.com/pic/Cours877-3.pdf

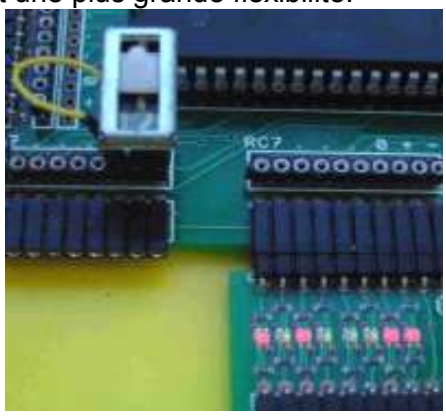
4. Interfaces programmables et applications
- en travail -

Entrées analogiques

Les PICs n'ont pas tous des canaux analogiques. Si oui, ils ont la même structure avec une précision de 10 bits, et nous utiliserons le mode qui laisse tomber les 2 bits de poids faible : il faut une construction qui respecte plusieurs règles pour avoir des mesures fiables en 10 bits. Si le PIC n'a pas d'entrées analogiques, on peut lire très efficacement des signaux analogiques en mesurant des temps de charge et décharge de condensateurs. Voir http://www.didel.com/picg/doc/DopicAD.pdf

5.1 Initialisation

Le 16F877 a 8 canaux analogiques possibles sur le portA et le portE. Il en suffit souvent d'un ou deux, et on veut garder les autres lignes pour des entrées-sorties tout-ou-rien. Sur le 16F877, 4 bits du port AdCon0 offrent quelques combinaisons, et il faut bien réfléchir pour câbler son application. Les processeurs plus récents offrent une plus grande flexibilité.



Il faut comprendre que dans l'initialisation, on dit quels sont les canaux analogiques en service (registre AdCon1), mais le programme lit un canal à la fois et il faut dire lequel on lit (registre AdCon0, nommé AnSel sur d'autres PICs).
Pour la lecture, il faut sélectionner le canal, attendre 20 microsecondes que les aiguillages internes se positionnent, démarrer la conversion et attendre qu'elle soit terminée, avec le résultat dans AdResH (on a décidé d'oublier la partie poids faibles AdResL, mais on peut la lire sur les bits 7 et 6).
Le microdule Pot (rotatif ou linéaire) permet d'injecter un signal ente 0 et 5V sur les bits 0 et 1. Mettons-le sur le portA .

Initialisons RA0 en entrée analogique et llisons la valeur.
Dans les définitions on déclare

```
DirA  = 2'000001 ; ou 2'111111 (RA0 seul doit être en entrée)
IniAdCon1   = 2'00000010   ; sel RA5 RA3 RA2 RA1 RA0
IniAdCon0   = 2'00001110   ; sel RA0 seul
SelAd0      = 2'11000001   ; sel AD0 en mode8 bits
```

(les valeurs de ces bits sont extrait de la documentaion de 200 pages du 16F877)
Dans l'initialisation on doit avoir

```
Clr   PortA
Set   Status:#RP0
Move  #IniAdCon1,W
Move  W,AdCon1
Move  #DirA,W
Move  W,PortA
Move  #0,W
Move  W,TrisC
Clr   Status:#RP0
```

Et dans le programme

```
      Move  #SelAd0,W
      Move  W,AdCon0
      Call  Del20       ; attente 20 us
      Set   AdCon0:#Go  ; C'est parti
 C$:  TestSkip,BC AdCon0:#Go
      Jump  C$          ; On attend
      Move  AdResH,W
      Move  W,PortD     ; On affiche la valeur
```

Pour le test, il faut brancher un potentiomètre sur RA0 et utiliser le programme **T877Ana.asm**

Sans potentiomètre, si on touche avec le doigt, on perturbe l'entrée à haute impédance.

5.2 Lecture par interruption

L'attente sur la conversion prend 100 à 200 microseconde. On a donc avantage à lire les canaux analogiques par interruption. Passez plus loin si cela semble compliqué.

Le timer déclanche toutes les 20ms une interruption qui démarre une cascade d'interruptions pour lire les canaux sur RA0, RA1, RA3 (RA2 est spécial).

Au moment de l'interruption, il faut décider d'où elle vient, et un pointeur doit mettre les valeurs lues dans une zone mémoire.

Dans les définitions, il faut déclarer

```
      IniOption  = 2'00000110 ; divise par 128
      IniTmr0    = -156           ; 156 x 128us = 20ms
      IniAdCon310 = 2'00000100    ; sel RA3 RA1 RA0
      SelAd0     = 2'11000001; sel AD0 en mode8 bits
      SelAd1     = 2'11001001
      SelAd2     = 2'11010001
```

Dans les variables, il faut déclarer

```
      Var   SaveF
      Var   SaveW
      Var   NCanal
      Var   Canal0
      Var   Canal1
      Var   Canal2
```

L'initialisation contient les instructions

```
      Clr   PortA
      Set   Status:#RP0
      Move  #TrisA,W
      Move  W,PortA
      Move  #TrisC,W
      Move  W,PortC     ; Affichage
      Move  #IniAdCon310,W
      Move  W,AdCon1
      Clr   Status:#RP0
      Move  #IniOption, ; Le timer
      Move  W,Option
      Clr   IntCon:#TOIF
      Move   #2**GIE+2**PEIE+2**TOIE,W    ; Timer seul
      Move  W,IntCon
```

La routine d'interruption contient les instructions

```
      TestSkip,BC  IntCon :#TOIF
      Jump  InterTimer
      TestSkip,BC  PIR1 :#ADIF
      Jump  InterAna
      Jump  Error
InterTimer :
      Clr   IntCon:#TOIF
      Move  #IniTmr0,W
      Move  W,TMR0
      Clr   IntCon:#TOIE      ; Coupe  les interrupt timer
; On met en route 3 interrupt Ana
      Move  #3,W
      Move  W,NCanal
      Move  #SelAd0,W
      Move  W,SaveSelAD
```

```
      Set   PIE1:#ADIE
DebAna:
      Clr   PIR1 :#ADIF
      Move  SaveSelAD,W
      Inc   SaveSelAD
      Move  W,AdCon0
      Move  #2'00001000,W
      Add   W,SaveSelAD
      Call  Del20
      Set   AdCon0:#Go  ; C'est parti
      Jump  Rt    ; Suite au prochain interrupt Ana
InterAna :        ; AdCon0:#Go a déclanché l'interrupt
                  ; Il faut sauver AdResL au bon endroit
      Move  Canal0,W
      Move  W,FSR
      Move  NCanal,W
      Add   W,FSR
      Move  AdResH,W    ; mets à zéro Go
      Move  W,{FSR}     ; voir section 6 ??
; on prepare la lecture suivante
      DecSkip,EQ  NCanal
      Jump  Rt
; On a lu les 3 canaux, on repasse in interrupt timer
      Clr   PIE1:#ADIE
      Set   Intcon :#TOIE
Rt :
```

Le programme complet se trouve sous **T877AnaInt.asm**

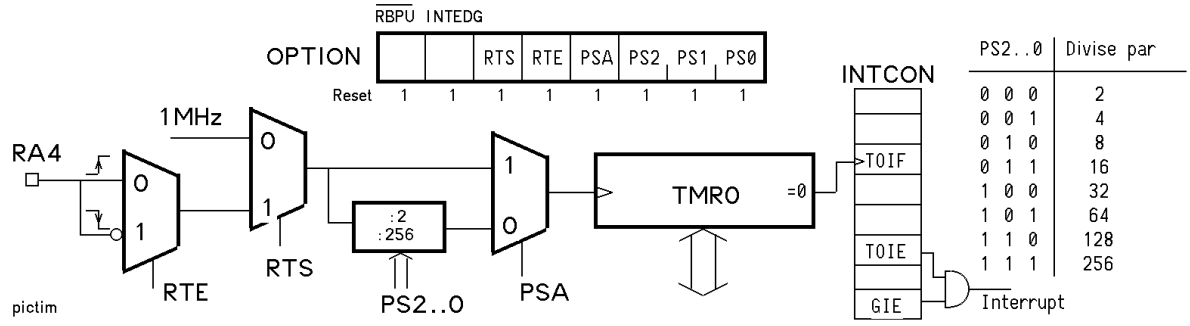
On voit que gérer plusieurs évènements par interruption n'est pas très simple. Pour une télécommande infrarouge qui doit lire trois potentiomètres et envoyer un train d'impulsions toutes les 40 us, les interruptions ne sont pas utilisées car les deux tâches à effectuer ne sont pas simultanées. L'écriture du programme est très simple.

4.Timers et interruptions

Les PICs ont 1,2,3 timers. Ce sont des compteurs avec de la logique autour pour compter plus ou moins vite et mesurer ou générer des signaux. Ce document se limite au TMR0, qui existe dans tous les PICs 10F, 12F, 16F. Les autres timers doivent être étudiés avec les programmes de test mis à disposition et avec la documentation du fabricant ou des traductions.

4.1 Timer 0

Le timer 0 est un exemple simple d'un circuit programmable. Le TMR0 est un compteur que le programme peut lire ou écrire comme une variable. Il tourne toujours, et quand il déborde, il active une bascule, appelée TOIF (Timer Overflow Flag). Cette bascule est dans le registre IntCon.
Le compteur est précédé d'un prédiviseur programmable, c'est à dire que l'oscillateur qui fait tourner le processeur peut être divisé par une valeur codée sur 3 bits pour avoir un compteur qui tourne à 1 MHz, 0.5 Mhz, … 3.9 Khz (processeur à 4 MHz). Il y a encore une possibilité pour compter des impulsions extérieures propres (s'il y a des rebonds, ils seront tous comptés).



Le schéma-bloc est très clair. On voit que pour compter le front descendant d'un signal câblé sur RA4 sans prédivision, il faut charger 2'00111000 dans Option.

Testons la fonctionnalité du TOIF en incrémentant un compteur (PortD) quand le TMR0 déborde. On compte si TOIF est actif, et on efface immédiatement TOIF (on ré-arme la bascule), donc on ne compte qu'une fois par tour.

```
            Move  #2'00000111,W     ; prédiviseur max
            Move  W,Option
            Clr   TMR0
            Clr   Option :#TOIF
            Clr   PortC
     At$:   TestSkip,BS Intcon:#TOIF
            Jump  Ato   ; on attend
            Clr   Intcon:#TOIF
      ; on pourrait initialiser TMR0 à une valeur qui raccourcirait le
      cycle
            Inc   PortC
      ; on pourrait utiliser le PortC comme compteur 16 bits, (voir T877T0c.asm)
            Jump  At$
```

Le programme **T877Tmr0.asm** permet de tester. Modifiez la valeur de prédivision dans Option.

4.2 Mesures de durées

Le timer peut compter plus vite qu'une boucle d'attente, et il compte sans ralentir le programme. Pour mesurer la durée d'une impulsion sur le poussoir (ou l'impulsion plus rapide d'une télécommande infrarouge), on met le compteur à zéro au début de l'impulsion et on attend la fin de l'impulsion pour faire la mesure.

```
      ; le signal est sur bPous sur le portC
      Atp:  TestSkip,BS  PortC :#bPous
            Jump  Atp   ; on attend l'action
            Clr   TMR0  ; Prédiviseur selon les durées à mesurer
      Atr:  TestSkip,BC PortC :#bPous
            Jump  Atr
            Move  TMR0,W
      ; W contient la durée. On peut l'afficher ou l'utiliser.
```

A vous de faire le programme, cette fois ! Mais attention, le temps maximum mesurable est 256X255 microsecondes = 60ms, trop peu pour un poussior. Il faut ajouter un compteur, qui mesure par exemple le centième. Le truc est expliqué plus loin si vous ne voyez pas..

4.3 Timer par interruption

Dans l'exemple précédent, on voit que le timer augmente la précison de la mesure, mais le programme est bloqué à surveiller le poussoir.
L'interruption permet d'avoir une tâche qui s'exécute quand il le faut, en suspendant l'exécution du programme principal, qui en fait ne se rend pas compte qu'il est interrompu.
La routine d'interruption est à l'adresse 4. Le programme principal doit sauter par dessus..
Prenons un exemple trop simple. Le programme principal compte de façon visible sur le portC, toutes les 0.1s par exemple. L'interruption du timer remet à zéro ce compteur toutes les secondes.
En plus de l'initialisation du port qui compte et du timer, il faut initialiser l'interruption, ce qui veut dire activer 2 bits dans le registre IntCon, une pour une activation générale des

interruptions (GIE General Interrupt Enable) et une pour l'interruption spécifique du Timer0 (TOIE Timer0 Interrupt Enable).
Ce qu'il faut faire à chaque interruption due à la bascule TOIF, c'est de quittancer le TOIF, et utiliser un compteur auxiliaire pour effacer le PortC toutes les secondes (on ne peut pas ralentir le timer suffisamment).
Le cœur de la routine d'interruption est donc

```
; On désactive le bit qui a demandé l'interruption
Clr   Intcon:#TOIF
Move  #IniTmr0,W
Move  W,TMR0
Inc   CInt  ; Compteur auxiliaire par 256 pour ralentir
Skip,NE
Clr   PortC ; Chaque 256 x 256 x 16 us = ~1s
```

On voit que ce programme, qui va être appelé sans avertir quand le timer déborde, utilise le regitre W. Il ne faudrait pas que le registre W, fréquemment utilisé par le programme, soit modifié par l'interruption. On doit donc rajouter 3 instructions au début et 5 instructions à la fin pour que la routine d'interruption sauve et rétablisse W, ainsi que les flags Z et C.

Le programme **T877T0Int.asm**, est le suivant :

```
\prog;T877T0Int.asm|Timer par interruption
; Le programme principal compte à ~15Hz sur le
; PortC et l'interruption mets à zéro ce portC toutes les 256 boucles d'interrupt.
; On voit que le compteur ne vas jamais très loin.
;Agir sur IniOption et sur IniTmr0
.Proc  16F877
.Ref   16F877

\var;Registres|
SaveF  = 16'20
SaveW  = 16'21
CInt   = 16'22
CX1    = 16'23
CX2    = 16'24

\var;Ports|
\b;PortC comme compteur
  DirC  = 0
\const; Initialisation
IniOption  = 2'0000001; :16
IniTmr0 = -200  ; 256-200
```

```
.loc    0
 Jump   Deb

.loc    4
; Interrupt tous les 256x16 us
 Move   W,SaveW ; Ne modifie pas F
 Swap   F,W     ; Truc
 Move   W,SaveF
; On désactive le bit qui a demandé l'interruption
 Clr    Intcon:#TOIF
 Move   #IniTmr0,W
 Move   W,TMR0
 Inc    CInt  ; Compteur auxiliaire par 256 pour ralentir
 Skip,NE
 Clr    PortC  ; Chaque 256 x 256 x 16 us = ~1s
F$:
 Swap   SaveF,W
 Move   W,F
 Swap   SaveW
 Swap   SaveW,W
 RetI
```

```
\b;Programme principal
Deb:
 Set    Status:#RP0
 Move   #DirC,W
 Move   W,PortC
 Clr    Status:#RP0
 Move   #IniOption,W
   ; Prescaler :16, 16 us
 Move   W,Option
 Clr    IntCon:#TOIF
 Move    #2**GIE+2**TOIE,W
 Move   W,IntCon

Loop:
 Inc    PortC
; Attente pour incrémenter env 15 fois par seconde
  A$:   DecSkip,EQ     CX1
 Jump   A$
 DecSkip,EQ  CX2
 Jump   A$
 Jump   Loop

.Loc    16'2007
.16     16'3F39
.End
```

Rappelons que le TMR0 est un compteur. Si on veut que TOIF s'active après un comptage de 200, il ne faut pas l'initialier avec la valeur 200 (il augmentrait de 200 à 256=0 et TOIF s'activerait), mais avec la valeur 256-200, identique pour l'assembleur à -200.

4.4 Timer1

Le timer 1, quand il est disponible comme sur le 16F877, est un compteur 16 bits avec prédiviseur. Le programme T877T1Int.asm teste son fonctionnement. Il peut être couplé à 2 pins sur lesquelles on branche un quartz horloger à 32 Khz, ce qui permet de programmer une horloge précise..
Le timer2 est beaucoup plus riche et permet du PWM (voir section 6.2) et une mesure plus efficace de durées d'impulsions.

6 Commande de moteurs

Il existe plusieurs types de moteurs de puissance très variée. Seuls les moteurs de faible puissance nous intéressent. La commande en tout-ou-rien d'un moteur ou électro-aimant s'apparente à la commande d'une LED, avec éventuellement un transistor amplificateur, donc rien de plus à dire, si ce n'est que les pointes de courant au démarrage du moteur

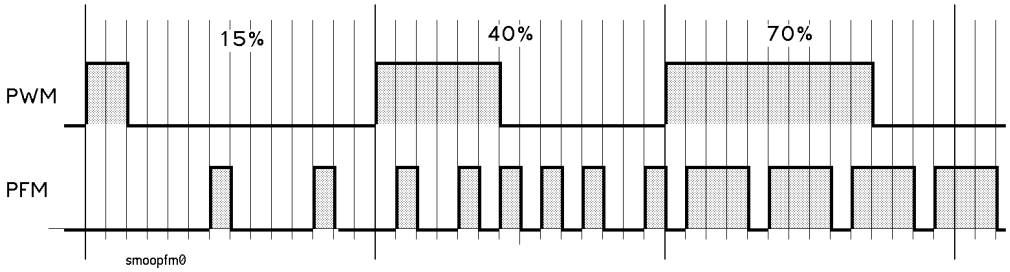
peuvent perturber le fonctionnement du processeur. On vérifie en remplaçant les moteurs par des LEDs.

6.1 Moteurs bidirectionnels

Les moteurs à courant continu ont deux fils et se commandent comme des diodes bicolores, mais en général aux travers d'amplificateurs appelés ''ponts en H''.
Si le courant demandé est entre 10 et 50mA, on peut lier des sorties du processeur entre elles pour augmenter le courant. Microchip parle de 20mA par sortie, mais c'est en court-circuit !

6.2 PWM (Pulse Width Modulation)

Le principe du PWM est de varier la largeur d'une impulsion répétitive. Les processeurs performants ont des canaux PWM obtenus en programmant le timer 2 ou le timer3, mais il faut utiliser les sorties prévues sur le microcontrôleur. Ce n'est pas très simple si le sens du moteur doit être inversé. Il faut bien lire la documentation du fabriquant et passer du temps sur le programme de test.
En logiciel, le principe est simple et efficace, et permet de commander autant de moteurs que l'on veut, sur les sorties que l'on veut, à une fréquence et résolution adéquates pour des application non industrielles.
Les explications générales sont données sous www.didel.com/picg/picg87x/Picg75.pdf page 28.
Le programme de test **T877Pwm.asm** montre comment commander un moteur unidirectionnel et un moteur bidirectionnel en logiciel. Le programme de test **T877PwmT2.asm** utilise le Timer2.

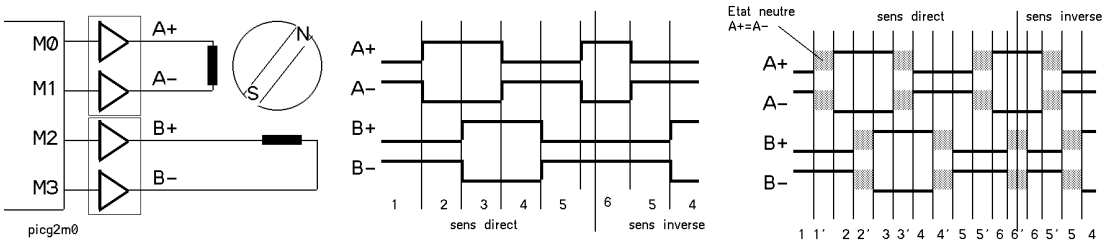


6.3 PFM (Pulse Frequency Modulation)

Le PFM est mal connu car il n'est pas utilisé sur les moteurs performants. Mais pour les moteurs jouet ou miniatures, il permet une commande à faible vitesse très intéressante. La fréquence est faible et facilite la programmation multi-tâche.
Les explications sont données sous www.didel.com/picg/picg87x/Picg75.pdf page 29.
Le programme de test **T877Pfm.asm** montre comment commander un moteur unidirectionnel et un moteur bidirectionnel en logiciel. Pour des moteurs ''pager'' qui ont une tension de démarrage importante à cause du frottement, le PFM avec des impulsions de 2 à 5ms suffisantes pour faire ''décoller'' le moteur permettent de couvrir toute la gamme de vitesses.

6.4 Moteur pas-à-pas

L'avantage du moteur pas à pas est que son angle de rotation dépend du nombre de pas, tant que le couple maximum n'est pas dépassé.
Ces moteurs sont commandés par 4 phases, avec des demi-phases possibles pour un mouvement plus régulier. Une table définit la séquence. Pour changer de sens on déplace le pointeur en sens inverse, en gérant correctement le passage par les extrémités. Des explications générales se trouvent sous www.didel.com/picg/picg87x/Picg75.pdf page 27, mais les exemples de programmes concernent des moteurs à 6 phases. La transposition est facile.



Les moteurs horlogers (Lavet) ont 6 phases (Switec, Wellgain). Des explications détaillées sont données dans le lien ci-dessus et sous www.didel.com/picg/doc/DopiSwi.pdf . Une doc plus récente avec exemples se trouve sous www.didel.com/bot/step/Step.doc

Avec du PWM ou du PFM, on peut lisser les phases pour se rapprocher d'une excitation sinusoidale. Ceci ne présente pas d'intérêt, mais c'était une recherche intéressante ! http://www.didel.com/picg/doc/DopiSmoo.pdf et sous http://www.didel.com/picg/doc/DocLimot.pdf

6.5 Moteurs ''brushless''

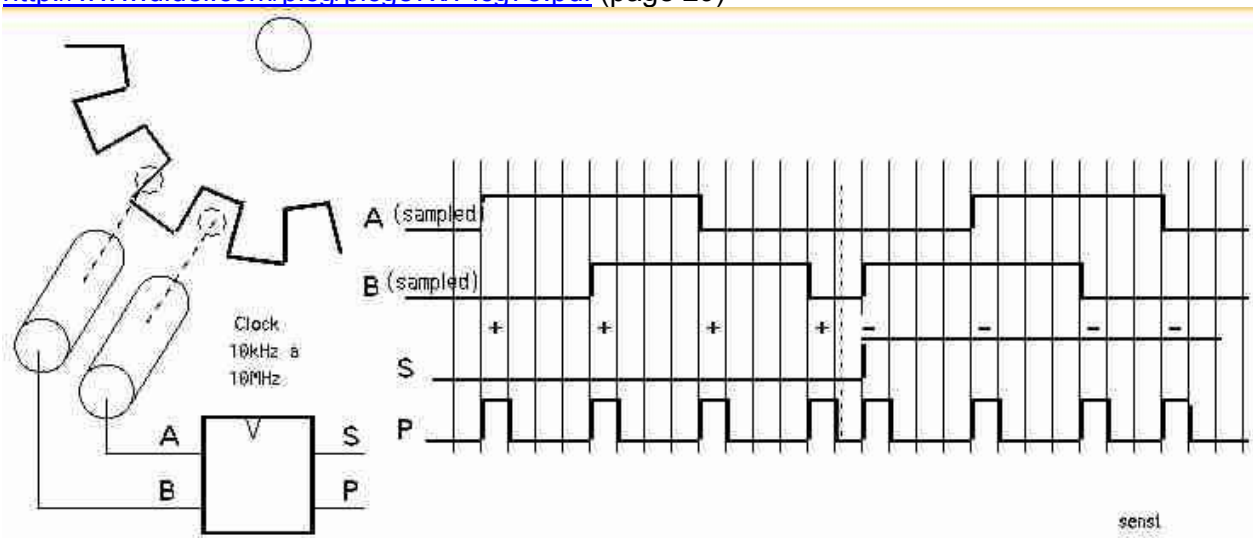
Les moteurs sans collecteur sont des moteurs pas à pas asservis par une électronique qui permet d'obtenir le couple maximum. Ils sont donc commandés par des circuits extérieurs au microcontrôleur qui reçoivent les ordres de vitesse via une interface parallèle ou série (les modélistes avion utilisent des circuits avec une entrée PPM comme les servos).

6.6 Servos de télécommande et PPM

Les servos de télécommande ont une entrée qui est une impulsion de 1 à 2ms répétée toutes les 20ms. La durée de l'impulsion fixe la position du servo (Pulse Position Modulation).
Pour un exemple de programme de codage simple (monotâche), voir http://www.didel.com/picg/doc/DocServo.pdf.

6.7 Encodeur

Pour connaître la position d'un moteur continu ou d'un axe, on utilise un encodeur (quadrature encoder) qui génère des impulsions déphasées. A défaut de circuits spécialisés, un programme échantillonne et compte/décompte les incréments jusqu'à des périodes de 100 us par transition. Des algorithmes efficaces sont expliqués sous http://www.didel.com/picg/doc/PicSoft.pdf (page 15-16) et http://www.didel.com/picg/picg87x/Picg75.pdf (page 29)



6.8 Capteurs de distance et autres

Les capteurs génèrent des impulsions ou des niveaux analogiques que les microcontrôleurs gèrent facilement. Les capteurs intelligents ont un contrôleur propre et communiquent en général en série. Une documentation sur les capteurs de distance par infrarouge se trouve sous

http://www.didel.com/doc/sens/DocIr.pdf
http://www.didel.com/doc/sens/DocIrt.pdf
http://www.didel.com/doc/sens/DocSharp.pdf
Internet permet de trouver beaucoup d'information plus ou moins complète sur les capteurs et leurs interfaces.

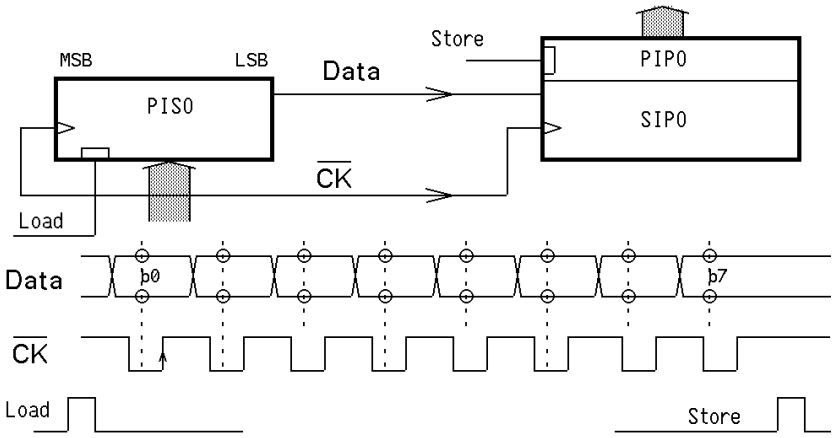
Les capteurs «''intelligents'', c'est à dire qui ont un préconditionnement des signaux, envoient souvent l'information en série vers le processeur.

7 Transferts série

Si une application comme un affichage a besoin de plusieurs lignes de commande, il est préférable d'utiliser un circuit spécialisé et lui transférer l'information en série sur quelques fils, donc en n'utilisant que 1 à 4 sorties du processeur.

7.1 Registres série

L'information d'un registre peut être décalée en série. A chaque impulsion d'horloge, un bit est décalé. Une impulsion Load charge au début l'information dans le registre série (PISO – Parallel In Serial Out), 8 impulsions CK décalent. Si le registre destination (SIPO – Serial In Parallel Out) a un registre en sortie une impulsion Store charge ce registre après les 8 impulsions de décalage. La polarité de l'horloge et des impulsions dépend des circuits utilisés.



Le microcontrôleur peut être d'un côté ou de l'autre. L'instruction de décalage va simuler le registre et les signaux nécessaires seront activés/désactivés par programmation.

Les routines d'écriture et de lecture sont données sous www.didel.com/picg/picg87x/Picg76.pdf page 36. Le circuit Ext8i8o ajoute 8 entrées et 8 sorties en n'utilisant que trois lignes du processeur www.bricobot/kits/Ext8i8o.pdf Les signaux Load et Store sont générés à l'intérieur du module. Le programme **T877Ext8i8o.asm** teste ce module d'extension.
Notons encore que les registres à décalage peuvent être cascadés. Avec le même nombre de lignes utilisées sur le microcontrôleur, on peut commander autant d'entrées sorties que l'on veut, mais évidemment cela prend du temps de transfert.

7.2 Transfert en écriture

La routine exemple ci-dessous doit être adaptée à l'application. Le décalage ne doit pas nécessairement de faire à droite, et la polarité des impulsions peut être inversée.

```
\rout:Write|Transfer 8 bits serially
\in:DataOut, transferred LSB first
\mod:DataOut, C1
Write:
      Move  #8,W
      Move  W,CntCk
L$:
      RRC   DataOut
      Skip,CC
```

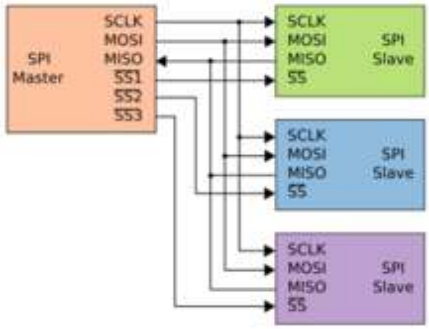
```
        Set    PortA:#bData
        Skip,CS
        Clr    PortA:#bData
        Clr    PortA:#bCk
        Set    PortA:#bCk
        DecSkip,EQ CcntCk
        Jump   L$
        Set    PortA:#bStore
        Clr    PortA:#bStore
        Ret
```

7.3 Transfert en lecture

```
\rout:Read|Get 8 bits serially
\out:DataIn read, transferred LSB first
\mod:DataIn, CntCk
Read:
        Move   #8,W
        Move   W,CntCk
        Set    PortA:#bLoad
        Clr    PortA:#bLoad
L$:
        ClrC
        TestSkip,BC PortA:#bData
        SetC
        RRC    DataIn
        Clr    PortA:#bData
        Clr    PortA:#bCk
        DecSkip,EQ CntCk
        Jump   L$
        Ret
```

7.4 SPI

Le SPI utilise 3 lignes pour le transfert et une ligne pour la sélection de chaque esclave.
http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus
Une logique interne envoie et reçoit l'information et active un flag quand le transfert est terminé. Par interruption, des débits importants, par exemple pour commander un affichage graphique, sont obtenus, mais les PICs sont rarement adaptés pour ce type d'application.
Voir aussi la doc plus ancienne
www.didel.com/picg/doc/DocSPI.pdf

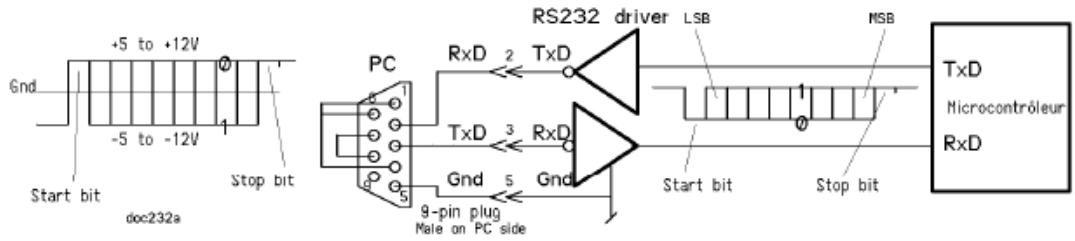


7.5 I2C

I2C est souvent intéressant pour se relier à des circuits horloge ou mémoire de stockage d'information. Les PICs haut de gamme ont des circuits câblés pour gérer le I2C, mais pour des applications simples et lentes, les routines logicielles sont tout aussi faciles à maîtriser et prennent une centaine d'octets.
Pour plus de détails, voir www.didel.com/picg/doc/DopicI2C.pdf .

7.6 USART

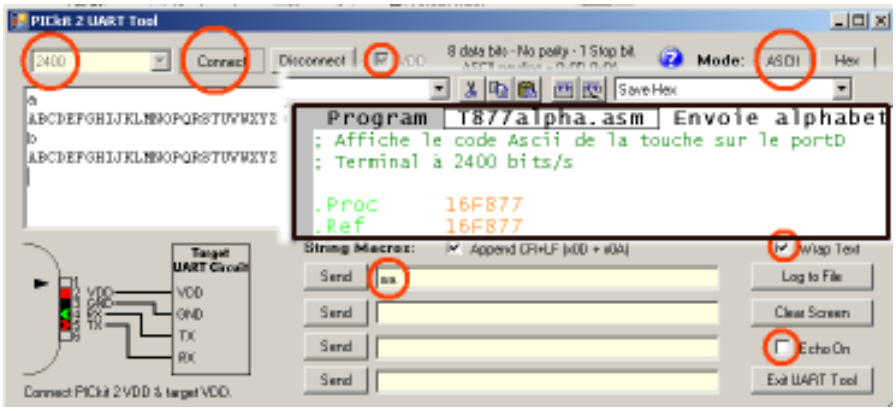
Tous les PC avaient anciennement une entrée dite ''RS232'' très pratique pour communiquer avec le microcontrôleur. Voir www.didel.com/dev877/PdSerie.doc comme exemple. Avec un adaptateur USB-RS232 on obtient théoriquement le même service. Côté PC, un programme ''terminal'' dialogue avec le clavier-écran via la ligne série.
Une fois la liaison établie, il faut installer un programme de communication série. Hyperterminal est difficile à configurer. TeraTerm est facile d'emploi
http://en.wikipedia.org/wiki/Tera_Term
Il peut se télécharger depuis www.didel.com/dev877/Ttermpro.zip.



Tous les microcontrôleurs PIC qui ont un USART (Universal Serial Asynchronous Receiver and Transmitter) câblé semblent avoir les mêmes registres de commande. Il faut définir la vitesse de transmission et gérer le protocole qui est simple si on ne se préoccupe pas de la gestion des erreurs de transmission. Pour envoyer, on écrit dans le registre TxD et on surveille le bit TxIF pour savoir si on peut envoyer le byte suivant.
Si on attend de l'information, on surveille un bit RcIF qui s'active quand un byte est arrivé. La lecture dans le registre RcReg met à zéro RcIF.
Le programme **T877Alpha.asm** envoie l'alphabet sur l'écran en réponse à une touche pressée.
Le transfert se fait à 2400 bits/s, ce que l'on peut facilement changer par programmation (à la fois dans l'initialisation et dans l'UART tool.
Le document http://www.didel.com/picg/doc/DopicSer.pdf donne des routines pour contrôleur sans interface série et avec.

A noter encore que le Pickit2 a un mode convertisseur série-UART (menu Tools, déplacer le Pickit). Le microdule 2840, avec sauf erreur tous les PICs 28 et 40 broches, peut donc être utilisé très facilement pour développer des applications avec communication série.

On peut taper sur le clavier, envoyer une chaîne de caractères.



On peut imaginer das applications ou le microcontrôleur est un esclave du PC, répond à des ordres, donne l'état de ses capteurs. L'application principale tourne dans le PC et n'est pas limitée en complexité.

7.7 USART par logiciel

Programmer un transfert série compatible RS232 est facile, mais il faut respecter un timing précis, et ne pas avoir d'interruption pendant le transfert. La vitesse de transfert est limitée à 2400 bits/s avec un processeur à 4 MHz et les routines nécessaires utilisent moins de 50 instructions. Voir www.didel.com/picg/doc/DopicUti.pdf qui documente aussi des routines pour afficher des textes, valables dans tous les cas.

7.8 Transferts ''one-wire'' Dallas

Dallas propose des circuits commandés par 2 fils seulement : la masse (gnd) et l'alimentation qui est pulsée pour transférer une information bien assez rapide pour beaucoup d'applications.
Pour plus de détails, voir http://www.maxim-ic.com/products/1-wire/

7.9 Petra

Le Bus Petra de Didel permet d'adresser 16 unités qui ont leur propre processeur, et d'avoir dans 16 variables successives l'image des informations gérées par ces unités (par exemple une mesure de distance.
Pour plus de détails, voir http://www.didel.com/Petra.pdf

7.10 Autres bus

Industriellement, le Can-bus est très utilisé, et les microcontrôleurs haut de gamme le supportent. Pour les petites applications, on peut faire beaucoup avec des registres à décalage. Pour communiquer entre microcontrôleur, ce que fait Petra, il faut une programmation très attentive concernant les durées d'exécution.

8 EeProm et bootloader

Seuls quelques PICs en boitier 6 et 8 pattes n'ont pas de mémoire Eeprom. Cette mémoire de 64 à 256 octets peut être écrite et lue par le programme.
Quelques processeurs 28 et 40 pattes, dont le 16F877, ont une possibilité supplémentaire de lire et modifier la mémoire programme, dite mémoire Flash, ce qui permet de lire ou écrire des grandes tables, et charger un programme sans passer par le programmateur, mais via une ligne série.

8.1 Mémoire EeProm

La lecture de la mémoire EeProm se fait en initialisant une variable adresse EeAdr et en lisant l'information dans la variable donnés EeData. Ces variables ne sont pas dans la banque 0 avec les ports, et il faut changer souvent de banque.
L'écriture est tout aussi simple, mais un groupe d'instructions bizarres doit être ajouté pour se protéger d'une écriture intempestive si le programme déraille et l'écriture dure quelques ms. Le bit WR du registre EECON1 en banque 3 est actif pendant que l'écriture se fait.
Le programme de test **T877EeProm.asm** contient les macros et routines que l'on peut directement utiliser dans ses programmes, s'ils n'utilisent pas l'interruption.

8.2 Mémoire Flash

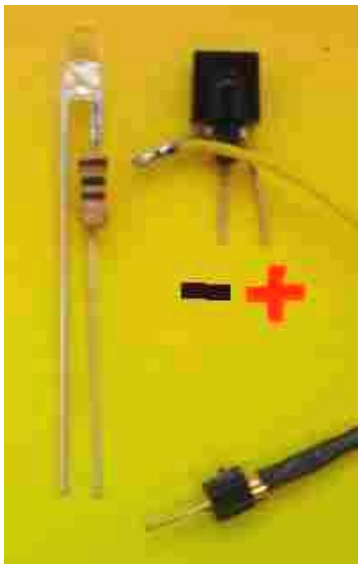
La mémoire programme du 16F877 peut être lue et écrite. La lecture est facile comme pour l'EeProm. Toutefois, le codage de l'adresse et des données se fait sur des paires de variables, puisque la mémoire peut avoir 4k (adresse 2 bits) et les instructions ont 14 bits.
L'écriture se fait instruction par instruction avec le 16F877, mais pour le 16F877A et le 16F882/884 l'écriture se fait par groupes de 4 positions consécutives pour gagner du temps, ce qui pose de joli problèmes d'alignements. La routine a été développée pour le bootloader du Dev877 et est à disposition.

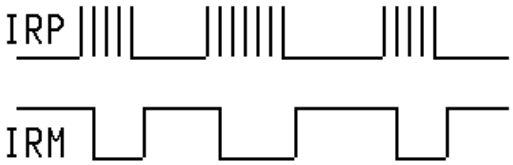
8.3 Boot loader

L'intérêt de l'écriture en mémoire programme est que l'on peut avoir dans une zone réservée du processeur qui ne sera jamais modifiée, un programme appelé ''boot loader'' qui charge depuis l'interface série le programme à exécuter. Ceci évite un programmateur de PIC, sauf naturellement pour mettre le boot loader en mémoire. Avec le Pickit2, on a le programmateur et l'interface série dans la même unité.
SmileNG peut être configuré pour transmettre en série le .hex à la fin de l'assemblage, si on sait initialiser le bon COM port. Le Didelbot www.didel.com/bot/ et la carte Dev877 www.didel.com/07dev877/indexF.html utilisent cette facilité, qui est prévue dans le nouveau Wellbot.

2.17 Infrarouge

Les télécommandes infrarouges transmettent des trains d'impulsion à 36-40 kHz. Un circuit IRM (Infra Red Module) filtre et donne sur sa sortie l'enveloppe du signal, en impulsions négatives de 0.8 à 3 ms en général.





Le plus simple est de brancher l'IRM sur une entrée, et de reconnaître l'action d'une télécommande de TV quelconque, comme si c'était un poussoir avec des rebonds.Essayez le programme **T877CntPous.asm** avec une attente à définir.
Un oscilloscope est nécessaire pour mettre au point des programmes non triviaux. Le principe des télécommandes Emir est expliqué brièvement sous www.didel.com/Ir/EmirSpecs.pdf
Les télécommandes PicoZ sont expliquées sous http://www.didel.com/Ir/IrControl.pdf .
Le format RC5 utilisé dans la plupart des télécommandes commerciales est abondamment documenté sur le Web.

11 Séquencement et multitâche

On doit souvent faire plusieurs choses en même temps, et chaque chose se fait en plusieurs étapes successives. Par exemple, si on attend un signal infrarouge tout en commandant la vitesse des moteurs, la tâche infrarouge est surveillée et un compteur suit l'évolution.
Si on doit surveiller un ascenseur, une tâche surveille les demandes et une tâche s'occupe du déplacement de l'ascenseur. Il y a beaucoup de façons de procéder, selon les applications.

11.1 Exemple 1 – Clignotement à vitesse variable

Prenons un cas simple avec deux tâches, une qui clignote une LED, et une qui surveille un poussoir pour modifier la vitesse de clignotement.
Toutes les 20ms puisque les tâches sont très lentes, on s'occupe successivement des deux tâches, qui partagent la variable VitCligno, qui définit le temps entre les impulsions lumineuses.
Clignotement : La routine TaskCli a deux sous-tâches définies par le compteur TkCli.
- Allumer la Led pendant 0.1s (IniLedOn=5 cycles compté par la variable CntLed)
- Eteindre pendant VitCligno cycles compté par la variable CntLed

Poussoir : La routineTaskPous a trois sous-tâches définies par le compteur TkPou
- Attendre que l'on pèse
- Attendre que l'on relâche en mesurant la durée
- Agir sur la variable VitCligno

Détaillons les instructions nécessaires pour ces tâches.

```
Tkcl1:  Clr PortC:#bLed ; allumé
        DecSkip,EQ  CntLed
        Ret         ; on reste dans la même tâche
; On prépare la tâche suivante
        Move  VitCligno,W
        Move  W,CntLed
        Inc   TkCli
        Ret
Tkcl2:  Set PortC:#bLed  ; éteint
        DecSkip,EQ  CntLed
        Ret         ; on reste dans la même tâche
; On prépare la tâche suivante
        Move  #IniLedOn,W
```

```
        Move  W,CntLed
        Clr   TkCli
        Ret
```

Pour la tâche poussoir, on veut distinguer les actions courtes, qui vont augmenter la fréquence de clignotement, et les actions longues, qui vont diminuer cette fréquence.
Ceci peut se faire avec un décompteur de durée de l'action, et le truc de saturer le compteur à zéro. On initialise le compteur pour qu'il arrive à zéro au bout d'une seconde (IniCntPous = 50 puisque 50x20ms = 1s)

```
TkPou1:  TestSkip,BS  PortC :#bPous
        Ret   ; on attend
; Action sur le poussoir, on prépare la tâche
        Move  #IniCntPous,W
        Move  W,CntPous
        Inc   TkPous
        Ret
TkPou2:  Dec CntPous
        Skip,NE
        Inc   CntPous     ; Sature à 1
        TestSkip,BC  PortC :#bPous
        Ret   ; on attent le relâchement
; On décide si on augmente ou diminue VitCligno
        DecSkip,EQ  CntPous
        Jump  Augmente
Diminue:  Dec     VitCligno
        Skip,NE
Augmente:   Inc   VitCligno   ; sature à 1
        Clr   TkPous
Ret
```

Le programme complet doit définir les constantes et les variables, initialiser les pointeurs de tâche à zéro et appeler dans une boucle synchronisée par le timer à 20ms les deux tâches.

```
Loop:  TestSkip,BS  Intcon:#TOIF
        Jump  Loop  ; On attend 20ms
        Clr   Intcon:#TOIF
        Move  #IniTmr0,W
        Move  W,TMR0      ; Réinit les 20ms
        Call  DoTaskCli
        Call  DoTaskPous
        Jump  Loop

DoTaskCli:
        Move  TkCli,W
        Add   W,PCL
        Jump  TkCli1
        Jump  TkCli2

DoTaskPous:
        Move  TkPou,W
        Add   W,PCL
        Jump  TkPou1
        Jump  TkPou2
```

Suivent les tâches décrites plus haut. Le programme complet est sous **T877TaskCli.asm**

11.2 Exemple 2 – Robot évitant les obstacles

Un robot a deux capteurs de distance en tout ou rien ou deux moustaches. On veut programmer un évitement d'obstacles où le robot va reculer et tourner pour éviter au mieux l'obstacle.
Il faut donc se souvenir de quelle moustache a touché jusqu'à ce que la manœuvre d'évitement soit terminée. Le robot est de type voiture avec une variable Vitesse qui définit le PFM du moteur et une variable Virage en tout ou rien.
Il faut bien se mettre d'accord sur le séquencement proposé, car il pourrait y avoir d'autres solutions.

1) Si pas de contact, on va tout droit
2) Si contact à droite, on inverse la vitesse pour DureeRecul=50 (50x20=1s) en tournant à droite.
3) Idem à gauche.

Cela fait 3 sous-tâches pour le déplacement (tâche TkDepl), et une tâche pour les capteurs qui ne semble pas nécessaire ; on s'en occupe pendant la tâche d'avance normale seulement.

```
TkDepl0 :   ; On avance en surveillant les capteurs d'obstacles
      TestSkip,BC  PortC :#bObstDroit
      Jump  ODroit
      TestSkip,BC  PortC :#bObstGauche
      Jump  OGauche
      Ret
Odroit :    ; On passe en tâche 1
      -- on modifie la vitesse en tournant à gauche
      Inc   TkDepl
      Ret

OGauche :   ; On passe en tâche 2
      -- on modifie les vitesses en tournant à droite
      Inc   TkDepl
      Inc   TkDepl
      Ret

TkDepl1 :   ; On recule pendant 1 s
      --- on décompte le temps
      Ret   ; le temps n'est pas écoulé
; on prépare pour reprendre la ligne droite (ou courbe)
      Clr   TkDepl
      Ret

TkDepl2 :   ; On recule pendant 1s
      --  on décompte le temps
      Ret   ; le temps n'est pas écoulé
; on prépare pour reprendre la ligne droite (ou courbe)
      Clr   TkDepl
      Ret
```

A vous de compléter les instructions. Elle n'ont pas été écrites pour vous montrer la façon dont on écrit le programme : on se préoccupe de la structure avant de coder le détail des actions, et simultanément on complète les liste des constantes et des variable. Il ne doit pas y avoir de nombres dans le programme. Toutes les valeurs doivent être déclarées au début pour être facilement changées.
Le programme complet pour une carte BimoPlus se trouve sous xxxx ??

11.3 Exemple 3 – Ascenseur 3 étages

Un ascenseur a 3 étages, soir un moteur, trois poussoirs pour appeler à l'étage (bAppel1 etc) et trois fin de course pour détecter l'arrivée de l'ascenseur à chaque étage (bStop1 etc). Les entrées sont sur le portC, les 2 sorties de commande de moteur sur le portA.
Définissons 9 tâches

Etage0 : ; arrêté à l'étages 0, surveille appel1 et appel2
Appel01 : ; on a pesé sur le bouton d'appel de l'étage 1 et l'ascenseur monte jusqu'au fin de course 1
Appel02 : ; on a pesé sur le bouton d'appel de l'étage 1 et l'ascenseur monte jusqu'au fin de course 2
Etage1 : ; etc

Ecrivons le début de ce programme, après les initialisations générales, y compris les instructions qui amènenent l'ascenseur à l'étage zéro, s'il n'y est pas.

```
        Clr     TaskAsc ; L'ascenseur est à l'étage zero au début
Loop :  Call    DoAsc
; rien d'autre à faire ?
```

```
        Jump     Loop

DoAsc : Move     TaskAsc,W
        Add      W,PCL
NoTk0 = 0
        Jump     Etage0
        Jump     Appel01
        Jump     Appel02
NoTk1 = 3
        Jump     Etage1
        Etc

Etage0 : TestSkip,BC  PortC :#bAppel1
        Jump     Ap01
        TestSkip,BC  PortC :#bAppel2
        Jump     Ap02
        Ret      ; On attend
Ap01:   Move     #Monte,W
        Move     W,PortC
        Inc      TaskAsc,W    ; continue en Appel01
        Ret
Ap02:   Move     #Monte,W
        Move     W,PortA
        Inc      TaskAsc,W
        Inc      TaskAsc,W    ; continue en Appel02
        Ret

Appel01 :        ; on attend le fin de course de l'étage 1
        TestSkip,BS PortC :#bStop1
        Ret
        Move     #Stop,W
        Move     W,PortA
        Move     NoTk1,W
        Move     TkAsc,W
        Ret      ; On continue en Etage1

Appel02 :        ; on attend le fin de course de l'étage 2
        TestSkip,BS PortC :#bStop2
        Ret
        Move     #Stop,W
        Move     W,PortA
        Move     NoTk2,W
        Move     TkAsc,W
        Ret      ; on continue en Etage2
```

Etc, c'est vraiment facile à écrire, facile à tester et facile à modifier. Le programme complet s'appelle **T877TaskAsc.asm**

11.4 Résumé

Décomposer un application en tâches qui se déroulent suffisamment souvent est plus simple qu'une gestion par interruption. Les tâches peuvent souvent se tester séparément en écrivant des programmes simples.
La difficulté est de trouver une période d'échantillonage assez rapide pour que toutes les tâches soient servies dans le temps voulu, et assez longue pour avoir assez d'instructions pour exécuter chaque sous-tâche.
Les interruptions sont réservées pour des communications rapides, RS232, I2C, et il faut veiller à ce qu'elles soient servies aussi rapidement que possible.