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

Exemple de programmation synchrone

Le principe de la programmation synchrone est expliqué dans des documents écrits à différentes époques, et se recoupant partiellement:
www.didel.com/picg/doc/DocPicSync.pdf (bon complément de ce texte)
www.didel.com/picg/doc/DocSync.pdf (le plus complet)
www.didel.com/picg/doc/DopiSync.pdf (en anglais)

Ce document donne un exemple pratique, bien optimisé par une utilisation fréquente dans des projets de Didel. Des modules de librairie (librairie YY car tous les programmes commencent par ces deux lettres) sont documentés, et associés à des programmes de test. A noter encore que les modules arbre de tâches ont l´extension .ASI et terminent avec la lettre T, qui indique qu´ils doivent être insérés dans une boucle ou interruption synchrone.

Module base de temps et clignotement

Ce module active un flag quand 20ms se sont écoulées, décompte une variable Timer1 toutes les 20ms (si elle est différente de zéro), et clignote une LED toutes les secondes.

Regardons le code avant de le commenter et comprendre comment il est appelé et comment il s´utilise.

```
Module YYTimeT.asi  Timer 20ms et clignotement d´une led à 1 Hz
Cette tâche est appelée toutes les Perio (=100 microseconde par exemple)
; Doit être en page zéro. Durée des tâches 8 à 15 us
Time:     Aiguillage des tâches
          Move      TkTime,W
          Add       W,PCL
          Jump      Tt0
          Jump      Tt1
.If       (APC/256) .NE. 0
Aie, l´aiguillage des tâches Time n´est plus entièrement en page 0
.Endif
Tt0:       On reste dans cette tâche 20 ms
          DecSkip,EQ  Cnt20ms
          Jump      Zt          ; temps pas encore écoulé
          Move      #IniCnt20ms,W
          Move      W,Cnt20ms
; On passe ici toutes les 20 ms
          Set       FlagTimer:#b20ms
          Test      Timer1
          Skip,EQ
          Dec       Timer1
          Inc       TkTime    ; Pasera à la tâche suivante
          Jump      Zt
Tt1:      Clignote toutes les secondes
          LedOff
          DecSkip,EQ  Cnt1s
          Jump      N$
          Move      #IniCnt1s,W
          Move      W,Cnt1s   ; 1 sec
; On passe ici toutes les secondes
          Set       FlagTimer:#b1s
          LedOn
 N$:      Clr       TkTime    ; Recommencera
;         Jump      Zt
Zt:
.End
```

Si l´aiguillage des tâches n´est pas compris, relire www.didel.com/picg/pic87x/Picg74.pdf.

La première tâche décompte Cnt20ms et recharge ce décompteur quand il arrive à zéro. La valeur IniCnt20ms dépend de la période d´appel de l´arbre (égale à Perio exprimé en microsecondes). On doit donc déclarer IniCnt20ms = 20000/Perio. La variable Cnt20ms doit également être déclarée.

Si Cnt20ms n´a pas atteint la valeur zéro, on saute à la fin du module, sans modifier le pointeur de tâche, donc on décomptera toutes les durées Perio=100ms par exemple.

Si Cnt20ms passe par zéro, toutes les 20ms, on recharge le compteur, active un flag et décrémente un compteur, s'il est différent de zéro.

Le bit b20ms dans la variable FlagTimer va rester activé si on ne fait rien ailleurs. L'idée est que dans une application, si on veut par exemple lire une touche toutes les 20ms pour éviter les rebonds de contact, on attendra pour lire que le flag soit à un (instruction TestSkip,BS FlagTimer:#b20ms), et on remettra le flag à zéro immédiatement après.

Le décompteur Timer1 reste à zéro si on ne fait rien. Si dans une autre tâche ou dans le programme principal on veut attendre jusqu'à 255 fois 20 ms, on initialise Timer1 et on attend qu'il soit à zéro (instruction Test Timer1 suivi de Skip,EQ ou Skip,NE). Le temps d'attente max est donc de 5,1 secondes. Il n'y a rien à faire quand le compteur est arrivé à zéro: il se bloque à cette valeur.

Donc, toutes les 20ms, on passe dans la 2e tâche, Tt1. On retrouve un décompteur qui va décompter 50 fois avant d'être réinitialisé. Dans les deux cas, on remet à zéro le pointeur de tâches, donc cette 2e tâche est appelée toutes les 20ms. Toutes les secondes, on ne revient pas immédiatement. Après réinitialisation du compteur, on active un Flag, on pourrait ajouter un Timer2 qui irait jusqu'à 255 secondes (ou plus si c'est un compteur 16 bits), et on allume la LED. Elle sera éteinte au prochain passage dans cette tâche, puisque la macro LedOff est au début de la tâche Tt1. La Led reste donc allumée 20ms, et on lui dit 50 fois de s'éteindre avant de la rallumer immédiatement après la 50e fois. C'est le plus simple!

On va garder cette Led qui clignote dans tous nos programmes. Si elle clignote, c'est que beaucoup de choses fonctionnent, et cela permet de mieux cerner le bug. Si elle ne clignote pas, le problème est que cela déraille dans l'exécution des arbres de tâches.

Le module YyTimeT.asi doit être précédé par les déclarations des variables, constantes et macros, comme on le voit dans le programme de test.
On remarque que IniCnt20ms résulte de la division par Perio; ce serait une erreur de déclarer InCnt20ms = 200, car on peut être amené à changer Perio pour ajouter des tâches, ou changer un timing, et il ne faut pas devoir corriger à plus d'un endroit. Si Perio est inférieur à 80, il y a dépassement de capacité et il faut par exemple passer à 10ms plutôt que 20ms.

La macro LedOn est en général un Clr PortX:#bLed, puisque l'on préfère que par câblage un zéro allume la LED.

Le programme de test www.didel.com/picg/pics/pict87x/Py870t.asm est écrit pour un 16F87x qui a une Led sur RC0 (c'est le cas du WdPicDev87x).

On remarque ce programme initialise le timer TMR0 et attend que le flag T0IF passe à 1 pour lancer la tâche (voir www.didel.com/picg/pic87x/Picg77.pdf).

Program **Pict870y.asm** Test YyTimeT.asi

```
.Proc     16F870
.Ref      16F870
```

Variables **Variables**

```
.Loc      DebVar
TkTime:   .16      1          ; Pointeur de la tâche Time
Cnt20ms:  .16      1
Cnt1s:    .16      1
Timer1:   .16      1
FlagTimer:         .16      1
 b20ms    = 1      ; numéro du bit
 b1s      = 2
```

Constant **Constantes**

```
IniOption = 2'00001000     ; 1 us decrement
Perio     = 100    ; microsecondes
IniCnt20ms         = 20000/Perio
IniCnt1s  = 1000/20
```

```
Constant Ports
bLed      = 0
DirC      = 0       ; Il faut au moins le bit bLed en sortie

Macros Led
.Macro    LedOn
          Clr       PortC:#bLed
.Endmacro
.Macro    LedOff
          Set       PortC:#bLed
.EndMacro
.Loc      0

Program Initialisation
; On initalise les ports
          Move      #DirC,W
          Move      W,TrisC
; On initialise les variables
          Move      #IniCnt20ms,W
          Move      W,Cnt20ms
          Move      #IniCnt1s,W
          Move      W,Cnt1s
          Clr       FlagTimer
          Clr       Timer1
; On met en route le timer
          Move      #IniOption,W
          Move      W,Option
          Move      #256-Perio+2,W
          Move      W,TMR0
Loop:
 W$:      TestSkip,BS   IntCon:#TOIF   ; Attente overflow timer
          Jump      W$
          Clr       Intcon:#TOIF
          Move      #256-Perio+2,W
          Move      W,TMR0
 S1On     ; Active un bit pour mesurer la durée des tâches
.Ins      YyTimeT.asi         ; L'arbre des tâches
 S1Off
          Jump      Loop
.End
```

A noter que les tâches Time durent entre 8 et 15 μs. La réinitialisation du timer 4-5 μs. Il reste donc 80 μs pour d'autres tâches à exécuter toutes les 100 μs.

L'aiguillage de tâche doit être en page zéro. Pour les grands programmes, on initialise PcLath (voi www.didel.com/picg/doc/DocPage.pdf) ou on met tous les aiguillages de tâche en page 0, et les tâches à la suite comme des routines; le fichier YxxT.asi est coupé en deux: YxxS.asi et YxxR.asi.

```
PgTime:   Move      #(PgTime/256),W
          Move      W,PcLatH
          .Inc      YyTimeT.asi
(mettre dans ce module un .If  APC/256 .NE. PgTime/
Pgxx:     Move      #(Pgxx/256),W
          Move      W,PcLatH
          .Inc      YxxT.asi
(mettre dans ce module un .If  APC/256 .NE. Pgxx/25
          etc...
```

```
          Move      TkTime,W        ; |
          Add       W,PCL           ; |
          Jump      Tt0             ; > .Ins YxxS
          Jump      Tt1             ; |
Zt:
          Move      Tkuu,W
          Add       W,PCL
          Jump      Tuu0
          Jump      Tuu1
Zuu:
          Jump      Loop
.Ins      YyTimeR.asi
.Ins      YyUuR.asi
```

Les tâches des fichiers YyTimeR se terminent par un Jump Zt, pour continuer par l'aiguillage des tâches suivant. Le groupe de tâche suivant saute en Zuu.

Interruptions

Lancer les tâches par interruption permet d'avoir un programme principal sans contraintes temps réel, par exemple pour intéragir en série. Ce programme principal sera haché régulièrement par l'interruption du timer, et utilisera le temps restant. Si les tâches utilisent 80μs. le programme principal a 1/5 de la puissance du processeur à disposition, ce qui permet encore de faire beaucoup. Et il tourne rond; il n'y aura pas tout à coup un temps de réponse plus grand car il y a accumulation de tâches prioritaires en interruption.

L'interruption s'initialise et se sert très facilement, mais il faut sauver les registres et variables utilisées comme variables locales par le programme principal et les tâches (voir

www.didel.com/picg/pic87x/Picg77.pdf).

Il faut donc ajouter une définition et initialisation pour le registre IntCon, et modifier le début et la fin de l´appel des tâches. Le programme devient:

Program **Pict870yi** **Test Time et clignotement par interruption**

```
; ... définitions
.Loc    0
        Call    Init        ; Initialisation des ports, variable et timer
        Move    #2´10100000,W  ; GIE and TOIE on
        Move    W,IntCon
        Jump    Main
.Loc    4       ; Interrupts
```

Program **Interruptions**

```
.If     S1
S1On    ; Sync
.EndIf
        Move    W,SaveW  ; Sauvetage des registres
        Swap    F,W
        Move    W,SaveF
        Clr     Intcon:#TOIF
        Move    #256-Perio+2,W
        Move    W,TMR0
.Ins    YTimeT.asi
        Swap    SaveF,W  ; Récupération des registre
        Move    W,F
        Swap    SaveW
        Swap    SaveW,W
.If     S1
S1Off
.Endif
        RetI
```

Program **Main**

```
Main:
; Initialisations pour le programme principal
Loop:
; A vot´ bon coeur
        Jump    Loop
```

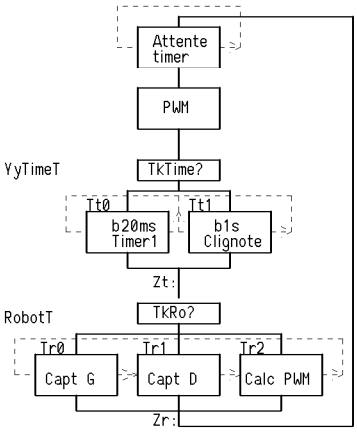
Routine **Init** **Initialisation ports et variables**

```
Init:
        ....
        Ret
.End
```

On remarque que l´interruption fait perdre 10 μs en appel et sauvetage/récupération de registres.

Ajout de tâches

La programmation synchrone a l´avantage d´être très modulaire. Si un moteur doit être commandé en PWM (voir www.didel.com/picg/pic87x/Picg76.pdf), il suffit d´ajouter les variables, déclaration et initialisations, et insérer la tâche dans la boucle. La communication se fait par la variable VitMot, qui assigne le pourcentage de PWM.



On a le choix entre ajouter des arbres de tâches qui vont s´exécuter en parallèle (ils avancent à chaque Perio), ou de compléter un arbre de tâches par des tâches supplémentaires. Dans le premier cas, il faudra augmenter la valeur Perio en tenant compte du cas pire de la somme des sous-tâches pouvant s´exécuter en parallèle. Dans le second cas, la mise à jour d´une variable selon les conditions extérieures sera ralentie.

Par exemple pour un suivi de piste pour un robot, la première tâche mesure le capteur droite, la 2e le capteur gauche, la 3e fait une différence ou un calcul plus savant pour mettre à jour une variable PwmMot, qui sera lue dans l´arbre des tâches PWM. Les trois premières tâches sont lentes, et peuvent être rassemblées dans un même arbre. Le PWM doit par contre être servi le plus souvent possible.

PWM

Un moteur peut être mono ou bidirectionnel, il peut y avoir plusieurs moteurs. Ces moteurs peuvent être commandés par des bits d'un même port, par des sorties mises en parallèle. Ces différents cas et des exemples de routines sont donnés dans le document www.didel.com/picg/doc/DocPwm.pdf

Encodeur

Un encodeur de souris permet de connaître la position et la vitese de rotation. Avec une période de 100 μs on ne peut pas aller en dessous de 500 μs par fente; si le disque a 60 fentes, cela correspond à une vitesse de rotation de 30 tours/s, 1800 t/min. Il faut donc un ou deux étages de réduction selon la taille du moteur. Le document www.didel.com/picg/doc/DocEnco.pdf donne quelques exemples.

L'éclairage de la diode infrarouge est usuellement commandé par logiciel pour économiser du courant: la diode est allumée 100 μs au moins avant la lecture, et elle peut être éteinte immédiatement après.

Connaissant la position, on détermine la vitesse avec une tâche qui mémorise la position toutes les 10 ms par exemple, et soustrait avec la valeur précédente pour obtenir la vitesse. A 30 t/s, la valeur de la vitesse est 40. La valeur absolue de la vitesse est souvent la variable utile.

Capteur analogique

Un ou plusieurs canaux analogiques peuvent être lus dans un arbre de tâches en effectuant successivement la sélection du canal, le départ de la conversion et la lecture.

Un capteur de Hall peut mesurer précisément la position d'un aimant.

Un capteur optique mesure une distance par différence entre l'intensité incidente avec la diode d'éclairage éteinte, puis allumée (voir www.didel.com/doc/sens/DocIr.pdf et www.didel.com/doc/sens/DocIrt.pdf).

Capteur de distance Sharp

Le capteur Sharp a un séquencement de transfert série qui se décompose facilement en une suite de tâches (voir www.didel.com/doc/sens/DocSharp.pdf).

Caméra linéaire

Un caméra linéaire est facile à interfacer (voir www.didel.com/doc/sens/Doc1301.pdf), mais les longues séquences de décalage série demandent une bonne réflexion pour être coupées en tâches de 100 μs. On donnera par exemple 4 ou 8 coups d'horloge par tâche, ce qui peut être une bonne solution pour réduire la résolution tout en moyennant, déterminant un maximum, etc..

jdn 031130