J.D. Nicoud, Mouette 5
CH-1092 Belmont, Switzerland
Tel +41 21 728-6156, Fax 728-6157
Email info@didel.com, www.didel.com

DIDEL

Fichier www.didel.com/picg/doc/DocSync.pdf

Programmation synchrone

La programmation synchrone permet d´avoir plusieurs tâches qui se déroulent en parallèle, avec une connaissance exacte des timings et du séquencement de chaque tâche. La réponse à une action extérieure peut être moins rapide que dans le cas des interruptions, mais elle est prévisible. La programmation synchrone permet de se constituer une librairie de périphériques virtuels, mais l´implémentation est assez différente de ce que proposent Scenics/Ubicom/Parallax et Microchip.

Principe des interruptions

Chaque source d´interruptions nécessite un câblage spécial sur le processeur. Un signal extérieur ou intérieur (timer) active un sémaphore; si l´interruption correspondante est activée, et que l´interruption générale l´est aussi, le processeur appelle la routine à l´adresse 4 au lieu d´exécuter l´instruction suivante. Il y reviendra après un Retl (Return d´interruption qui réactive l´interruption générale).

La routine d´interruption sauve les registres qui sont utilisés à la fois dans le programme principal et la routine d´interruption. On sauve au moins F et W. Il est nécessaire de sauver PcLatH si le programme principal et la routine d´interruption utilisent des tables. Il faut sauver FSR si le programme principal et la routine d´interruption utilisent ce registre. On ne sauve naturellement pas les registres qui sont en écriture seulement dans la routine d´interruption et en lecture seulement dans le programme principal (une valeur analogique), ou l´inverse (un caractère série).

Dans la routine d´interruption, il faut désactiver (en général remettre à zéro) le sémaphore qui a causé l´interruption et éventuellement réinitialiser des registres (par exemple le timer).

S´il y a plusieurs sources d´interruptions, il faut commencer par les trier (polling) pour partir dans la routine spécifique. A la fin de cette routine spécifique, on retourne en général au programme principal, et si une 2e interruption est en attente, la routine d´interruption est immédiatement rappelée (figure 1). Il peut donc être rentable dans certains cas, de tester s´il y a d´autres interruptions pendantes avant le Retl.

Donc, le problème est que si une interruption survient pendant qu´une autre interruption est servie, le temps d´attente est inprévisible. On pourrait naturellement réactiver l´interruption dans une routine d´interruption, mais cela implique de gérer une pile des variables sauvées à chaque interruption imbriquée, et le PIC n´est vraiment pas fait pour cela.

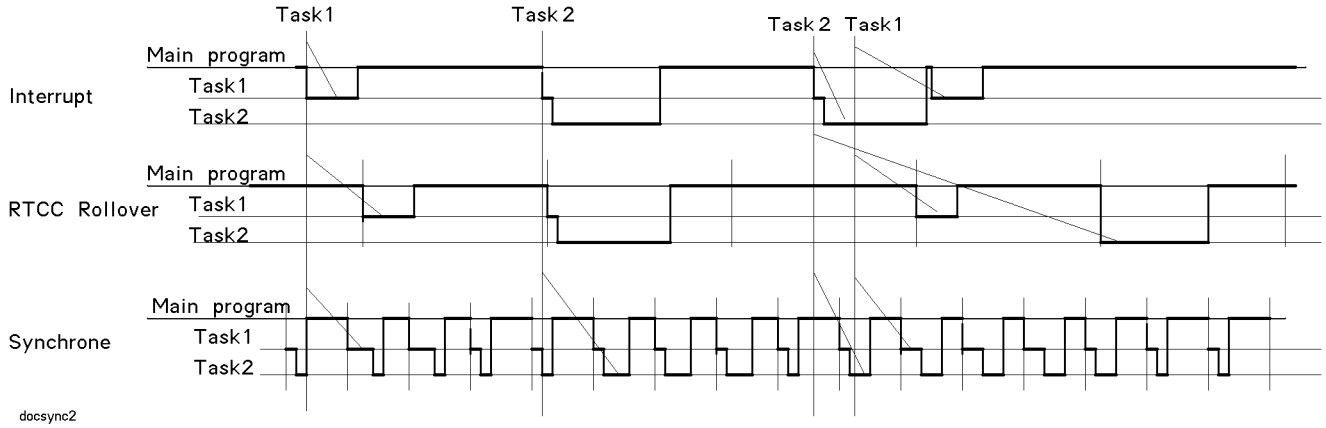

Figure 1 Trois techniques pour servir 2 tâches

Service d´interruption via le timer

Si un signal ne peut pas créer d´interruption directe, par exemple un poussoir sur le port RB, le timer peut être programmé pour déclencher une interruption régulière qui examine si une action doit être déclenchée suite à l´état extérieur. Dans la figure 1, deux tâches demandent une action aléatoire. L´interruption les sert immédiatement, sauf en cas de simultanéité. Le timer RTCC (Real time clock) examine périodiquement s´il y a une demande pendante. Le temps de réponse est naturellement augmenté.

La solution synchrone utilise également le timer, mais la tâche à exécuter à chaque interruption est coupée en petites tranches, de façon que en cas de simultanéité, toutes les tâches sont exécutées avec un temps de réponse prévisible, mais un pourcentage important du temps est perdu dans l´entrée dans l´interruption et dans les tâches.

Principe de la programmation synchrone

Un timer crée un séquencement de métronome à chaque 100 microsecondes par exemple. Le processeur exécute alors une séquence de tâches qui sont chacunes des machines à état fini comportant des sous-tâches pointées par un compteur. Par exemple pour clignoter une Led à 1 Hz, la première tâche (on devrait dire sous-tâche) boucle 10´000 fois avant d´incrémenter le pointeur de tâche. La tâche suivante inverse la Led, réinitialise le compteur par 10´000 et met à zéro le pointeur de tâche.

La décision pour d´autres tâches se prend avec la même période, pour exécuter à chaque cycle du PWM soft, surveiller des interrupteurs toutes les 5ms (à cause des rebonds de contact), etc. Chaque tâche ne fait que 10 à 20 instructions, et en 100 microsecondes on peut caser 4 ou 5 grandes tâches coupées en tranches de 20 μs au plus) en sachant exactement quand elles seront exécutées.

Le temps restant est pour un programme principal, qui n´a pas de contraintes temporelles, mais si la durées des tâches synchrones est régulière, le temps à disposition pour le programme principal est un pourcentage régulier du temps du processeur.

La figure 2 illustre ce séquencement. Toutes les 100 microsecondes, on gère le PWM bidirectionnel de 3 moteurs (32 us) et on passe à la tâche pointée par le compteur Task. Cette tâche décompte Tm1, et le pointeur de tâche est incrémenté dès que Tm1=0. On peut réinitialiser le compteur Tm1 dans la même tâche, ou dans la tâche suivante.

Les tâches 1,2,3 se suivent; il s´agit par exemple d´une opération complexe qui a été coupée en morceaux de 10-15 instructions (commande d´un moteur pas-à-pas par exemple). A noter que des variables globales passent les paramètres d´une tâche à l´autre, et que le programme principal ne doit pas lire une variable modifiée dans des tâches différentes.

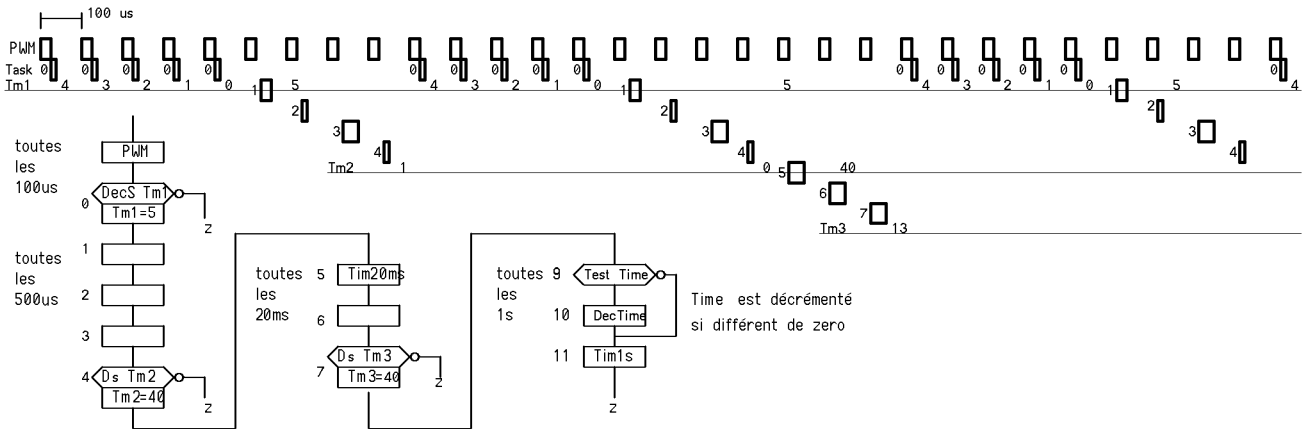

Figure 2 Programmation synchrone de plusieurs tâches

La tâche 5 décrémente un compteur Tm2, qui va donc être décrémenté toutes les 900 us. Si ce compteur est différent de zéro, on mets à zéro le pointeur de tâche TASK, et on recommence à la tâche 0 après le PWM. Lorsque Tm2 est à zéro, on le réinitialise à 40

(on veut des actions toutes les 20 ms) et on part dans les tâches 4,5,6,7.

La tâche 7, associée au décompteur Tm3, déclenche à son tour les actions qui doivent être exécutées toutes les secondes.

Programmation du timer

Si le programme principal est inexistant, ou ne contient qu'une tâche courte, le timer n'a pas besoin de créer une interruptionu. Le timer0 est initialisé et le sémaphore TOIF est testé. Lorsque le timer passe à zéro, le sémaphore s'active et il faut le désactiver.

```
Loop:
 W$:     TestSkip,BS  IntCon:#TOIF   ; pas compatible 12C508
         Jump     W$
         Clr      Intcon:#TOIF
         Move     #256-TPhase+2,W
         Move     W,TMR0
... Arbre des tâches
... Arbre 2 ou programme principal (durée limitée)
         Jump     Loop
```

Avec cette solution, le départ dans la première tâche se fait avec une incetitude (jitter) de 3 microsecondes, étant donné que la boucle W$: prend 3 μs.

L'initialisation du timer se fait avec les instructions

```
Move     #IniOption,W
Move     W,Option
Move     #256-TPhase+4,W
Move     W,TMR0
```

IniOption fixe le rapport de division du timer. Pour une division par 2

```
IniOption  = 2'00000000    ; :2  2us
```

TPhase dépends de cette division et de la fréquence d'horloge. A 4 Mhz et :2,

```
TPhase = Perio/2    ; Perio = 100
```

Le timer est un compteur, il faut donc le charger avec le complément à 256. Le +4 tient compte des 4 instructions qui sont en moyenne exécutées avant que le timer soit rechargé.

Mise au point

Pour aider à la mise au point du programme, il est très utile d'avoir une impulsion à chaque interruption. On écrira donc

```
.If      S1
S1On     ; Sync
.Endif
         Clr      Intcon:#TOIF
         Move     #256-TPhase+4,W
         Move     W,TMR0
.If      S1
S1Off
.Endif
```

Tout au début du programme, on déclare S1 = 1 si on veut que l'impulsion apparaisse sur la pin spécifiée.

Il est aussi très pratique de pouvoir afficher le pointeur d'une tâche sur le port B par exemple. Dans le programme principal on écrit:

```
Loop:
.If      Test
         Not      TkLed     ; PicG84, 870 ou WdPicDev87x inverse les bits
         Move     W,PortB
.Endif
```

Si le port B est utilisé ailleurs dans l'application, on le désactivera avec les instructions

```
.If      Test
.Else
         Move     W,PortB
.Endif
```

Dans la mise au point d'un arbre de tâche, il y a deux aspects: d'une part obtenir la bonne séquence de tâches en fonction des actions extérieures, d'autre part interpréter et générer correctement les paramètres. On peut souvent mettre au point une tâche après

l´autre en préparant les paramètres avant, et en affichant les résultats sur le port B.

Par exemple, si un arbre de tâche lit un potentiomètre (ou un capteur), et autre arbre génére le PWM, on commence par préparer deux programmes de test:

1) test de la lecture du pot; on affiche le résultat sur le port B

```
; dans la boucle synchrone
          .Ins      TkPot.asi
; dans le programme principal
          Not       ValPot,W
          Move      W,PortB
```

2) Test du PWM; on prépare la consigne PWM à l´avance

```
; dans l´initialisation ou le programme principal
          Move      #16´40,W
          Move      W,PWM
;dans la boucle synchrone
          .Ins      TkPwmT.asi
```

Timer avec interruption

Lorsque le sémaphore s´active, il peut déclencher une interruption. Ceci garanti un temps de réponse constant. Le début du programme s´écrit

```
.Loc      0
          Call      Init
          Move      #IniIntCon,W  ; GIE and TOIE on
          Move      W,IntCon
          Jump      Main
.Loc      4         ; (on y est)
; Sauvetage selon le programme principal
 S1On     ; Marque pour l´oscillo
          Clr       Intcon:#TOIF
          Move      #256-TPhase+2+2,W
          Move      W,TMR0
... Arbre 1
... Arbre 2
; Récupération selon sauvetage
 S1Off    ; Marque pour voir la durée d´exécution à l´oscillo
          RetI      ; retour de la routine d´interruption
Main:
          ; les interruptions sont lancées, autres préparations éventuelles
Loop:
                    ; D´habitude rien à faire ici, ou des tâches de déverminage et surveillance.
          Jump      Loop
```

La routine d´initialisation configure les ports, efface toutes les variables et initialise celles qui sont différentes de zéro, comme les décompteurs Tm1 Tm2 Tm3 de la figure 2.

Le timer est initialisé comme avant, et les interruption sont lancées au retour de la routine d´initialisation, en activant les bits GIE (General Interrupt Enable) et TOIE (Timer Overflow Interrupt Enable).

```
          IniIntCon = 2´10100000     ; GIE and TOIE on
```

Exemple

Premons l´exemple simple de clignoter une LED à 1 Hz. Si chaque tâche est appelée toutes les Tphase=100 microsecondes, deux tâches d´attente doivent être répétées 5000 fois chacune: 5000x100 us = 0.5s

Allumer
Rester 5000 sur la même tâche
Eteindre
Rester 5000 fois sur la même tâche

Comme 5000 = 256 x195. On déclare (pour pouvoir facilement changer le rapport cyclique)

```
          TPhase    = 100
          PeriodeLed= (1000000/TPhase)/256
          DurLedOn  = 195
```

DurLedOff = PeriodLed-195

et on aura besoin de deux compteurs CntLedHigh et CntLedLow.

L´arbre des tâches se programme facilement:

```
        Move    TkLed,W
        Add     W,PCL    ; attention la page
        Jump    TkLed0
        Jump    TkLed1
        Jump    TkLed2
        Jump    TkLed3
TkLed0: LedOn
        Inc     TkLed
        Move    #DurLedOn,W
        Move    W,CntLedHigh
        Clr     CntLedLow
        Jump    ZLed
TkLed1: DecSkip,EQ  CntLedLow
        Jump    ZLed
        DecSkip,EQ  CntLedHigh
        Jump    ZLed
        Inc     TkLed
        Jump    ZLed
TkLed2: LedOff
        Inc     TkLed
        Move    #DurLedOff,W
        Move    W,CntLedHigh
        Clr     CntLedLow
        Jump    ZLed
TkLed3: DecSkip,EQ  CntLedLow
        Jump    ZLed
        DecSkip,EQ  CntLedHigh
        Jump    ZLed
        Clr     TkLed
;;      Jump    ZLed
ZLed:
        ; Arbre des tâches suivant, par exemple une led
        ; qui clignote à une vitesse différente
```

A noter que l´on peut faire la même chose avec deux tâches seulement, mais leur durée seront augmentée.

jdn 030916