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

Programmation synchrone

Note: Ce document est complémentaire du document DocExSync.pdf qui donne un noyau optimisé autour duquel il est facile de construire des applications. Le document ci-dessous se veut didactique: il explique les principes, en supposant que le lecteur saura adapter les exemples et programmer son application. DocExSync.pdf décrit un programme qui fonctionne, et documente une librairie de tâches pour comander des moteurs, capteurs et interfaces.

La programmation synchrone permet d'avoir plusieurs tâches qui se déroulent en parallèle. Ce type de programmation est très modulaire, et relativement facile à mettre au point; c'est comme si on avait plusieurs processeurs qui chacun n'ont qu'une tâche à exécuter. Mais il faut être attentif à l'utilisation des variables communes entre plusieurs tâches.

Les tâches pour une même action sont regroupées dans un arbre. On parle souvent de machine d'état, pousque l'on passe d'un état à l'autre selon les conditions de transition. Chaque tâche ne prend que 20 à 30 instructions et est exécutée cycliquement. Chaque tâche définit quelle sera la prochaine tâche exécutée (elle-mêm, la suivante dans la liste ou une autre).

Le principe est de balayer régulièrement les arbres de tâches. On verra comment plus loin.

Exemple

Prenons l'exemple simple de faire clignoter une LED à 1 Hz avec une durée d'allumage en paramètre. 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 leurs durées seront augmentées.

Synchronisation

S'il n'y a pas de contraintes de timings précis, on exécute un arbre après l'autre. L'une des tâches d'un arbre peut être synchronisée par un signal extérieur, par exemple une horloge à la seconde.

```
Loop:
        .Ins    Arbre1
        .Ins    Arbre2
        .Ins    Arbre 3
        Jump    Loop
```

Si dans chaque arbre, les tâches ont la même durée, ce qui est facile à équilibrer avec des Nops, la synchronisation sera parfaite. Mais si on rajoute un arbre, toutes les tâches seront ralenties.

Il est donc préférable de synchroniser avec le timer. Il y a trois solutions

1) Test timer à zéro

Le Timer étant réinitialisé à la valeur de synchronisation, on attend qu'il passe à zéro

```
  W$:   Move    TMR0,W  ; Ne pas utiliser Test TMR0
        Skip,EQ
        Jump W$
        Move    #256-TPhase+2,W
        Move    W,TMR0
```

Cette solution, la seule possible avec les Pics 12 bits (12C508 par exemple) est dangereuse si le timer n'est pas en prédiviseur par 4 au moins: il ne faut pas que le timer incrémente plus souvent qu'il n'est testé.

2) Test du flag

Lorsque le timer passe à zéro, un sémaphore s'active. Il suffit de le tester, et de le désactiver après.

```
 W$:    TestSkip,BS IntCon:#TOIF   ; pas compatible 12C508
        Jump    W$
        Clr     Intcon:#TOIF
        Move    #256-TPhase+2,W
        Move    W,TMR0
```

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

3) Interruption

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

```
.Loc    0
        Call    Init
        Move    #2'10100000,W   ; GIE and TOIE on
        Move    W,IntCon
        Jump    Main
.Loc    4       ; (on y est)
        Clr     Intcon:#TOIF
        Move    #256-TPhase+2+2,W
        Move    W,TMR0
... Arbre 1
... Arbre 2
        Retl    ; retour de la routine d'interruption
Main:
        ; les interruptions sont lancées, autres préparations éventuelles
Loop:
                ; D'habitude rien à faire ici
```

```
        Jump    Loop
```

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:#T0IF
        Move    #256-TPhase+2+2,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. Naturellement, les macros S1On S1Of agissent sur la bonne pin.
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 ou 870
        Move    W,PortB
.Endif
```

Si le port B est utilisé dans l´application, on le désactivera ailleurs avec un

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

Dans la mise au point d´un arbre de tâches, 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 tâches lit un potentiomètre (ou un capteur), et un 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    TkPotT.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
```

jdn 01014/030311/031112