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

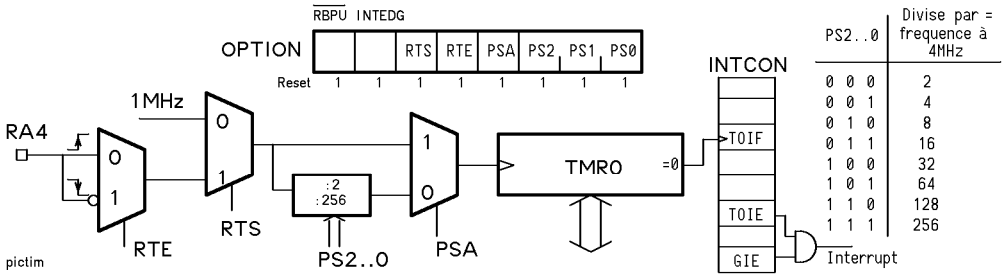
PIC*Génial* - Microcontrôleur PIC 16F84/16F870

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

7. Timer, interruptions et programmation synchrone

7.1. Le timer

Le timer est associé à un "watchdog" que nous ignorerons ici. Six bits du registre OPTION définissent l'action sur le compteur TMR0 (figure 1). Les deux bits de poids fort sont vus plus loin. TMR0 peut être lu en tout temps. Lorsque le timer arrive à zéro, le flag TOIF dans le registre INTCON s'active. En général, le programme surveille ce fanion, déclenche l'action quand il passe à un, remet le fanion à zéro, recharge le timer, et c'est reparti pour un tour. Comme c'est un compteur par 256, pour compter 100 incréments du compteur, il faut charger TMR0 avec la valeur 256-100, ce que l'assembleur calcule si on écrit -100. Une écriture bloque le comptage pour 2 μs.



pictim

Fig. 1 Schéma du timer

Le programme suivant incrémente le port B en fonction d'un timer le plus lent possible. Si on veut aller plus lentement, il suffit de déclencher l'action (ici incrémenter le port B) qu'une fois sur n, en utilisant un compteur décrémenté à chaque synchronisation du timer.

Program **Picgt8** **Test Timer**

```
.Proc    16F84
TOIF     = 2
```

Program **Initialisation**

```
.Loc     0
         Move    #0,W              ; Sorties
         Move    W,TrisA
         Move    W,TrisB
         Move    #2'00000111,W     ; Prescaler :256
         Move    W,Option
         Clr     TMR0
```

Program **Incrémente les LEDs sur le Port B**

(il faut décrémenter le port B puisqu'il y a inversion)

```
Loop:
         Dec     PortB      ; Every 256 x 256 µs
 W$:     TestSkip,BS  IntCon:#TOIF  ; Attente overflow timer
         Jump    W$
         Clr     Intcon:#TOIF
         Jump    Loop
.End
```

Si on veut compter les impulsions sur RA4 et afficher en continu l'état de TMR0, il faut modifier le registre OPTION, et le programme devient trivial:

```
         Loop:
                 Move    TMR0,W
                 Move    W,PortB
                 Jump    Loop
```

Si l'on veut qu'il se passe quelque chose de spécial après N impulsions externes, on initialisera et réinitialisera le registre TMR0 avec 256-N, et on surveillera le bit de dépassement TOIF.

0.95

7.2. Interruptions

Le PIC 16F84/16F870 accepte quelques interruptions, dont le mécanisme est très simple. L'interruption activée fait sauter le processeur à l'adresse 4 en sauvant l'adresse de retour sur la pile et en désactivant le fanion d'interruption général. L'instruction Reti (retour d'interruption) retourne au programme interrompu en réactivant le fanion d'interruption. Dans la routine d'interruption, il faut sauver tout ce que le processeur peut être en train d'utiliser et que la routine d'interruption va modifier (on sait ce que fait la routine d'interruption, mais pas ce que fait le programme principal). En général, W et F sont sauvés dans des variables réservées. Contrairement à d'autres processeurs, la pile n'est pas utilisable pour sauver des variables, et on ne peut pas écrire des routines récursives.

A noter encore que Move W,Var ne modifie pas Z, alors que Move Var,W ou Move Var,Var (noté Test Var) modifie Z. Il faut faire une astuce lors du sauvetage et rétablissement de W dans la routine d'interruption, en utilisant l'instruction Swap, qui elle ne modifie pas les fanions. Les instructions que l'on retrouve dans toutes les routines d'interruption du PIC 16F84/16F870 sont visibles dans le programme suivant. Si le programme principal n'a pas d'instruction Skip,EQ, Skip,NE qui testent le fanion Z modifié une ou plusieurs instructions avant, on peut éviter de sauver F. Ce genre de simplification ne se fait que si le gain de quelques microsecondes est critique pour l'application.

Il y a en plus du fanion GIE (General Interrupt Enable) des masques d'autorisation d'interruption et des fanions spécifiques à chaque type d'interruption. Ils se trouvent dans le registre IntCon (interrupt control). Pour le timer, il faut activer avec le bit TOIE (Timer Interrupt Enable) l'appel de la routine d'interruption quand TOIF (Timer Interrupt Flag) passe à 1. Il faut naturellement remettre à zéro TOIF avant de revenir avec l'instruction RETI.

0.9

Program **Picgt9** Timer par interruptions

```
Beep sur RA4 par interruptions et incrémentation du p
.Proc 16F84
DebVar= 16'20
; Bits du registre IntCon
TOIF = 2        ; Timer interrupt flag
TOIE = 5        ; Timer interrupt enable
GIE  = 7        ; Global interrupt enable
ModeIntCon      = 2**GIE+2**TOIE
ModeOption      = 2'00000011        ; Prédiviseur
```

Variables **Registres**

```
.Loc  DebVar
C1:   .16       1           ; Compteur pour
C2:   .16       1           ; boucle d'attente
CInt: .16       1           ; Compteur utilisé par le
CntLow:         .Blk.16     1       ; Compteur su
 MaskLow        = 2'00001111        ; Masque pour
 MaskHi         = 2'11110000        ; Masque pour
 MotifHi        = 2'10100000        ; Motif initial
SaveW:          .Blk.16     1       ; Sauvetage p
SaveF:          .Blk.16     1
```

Variables **Ports** RA4 et RB7..0 utilisés en sortie

```
bHP   = 4                   ; Buzzer sur RA4
DirA  = 2'01111             ; RA4 sortie buzzer, RA
DirB  = 0                   ; Tout en sortie
.Loc  0
      Jump      Deb         ; Sauter la zone d'interr
```

Program **Interruptions** Interruption chaque 125 x 16 μs = 2 ms

```
.Loc  4                     ; Adresse d'interruption
      Move      W,SaveW     ; Ne modifie pas F
      Swap      F,W         ; A cause du retour
      Move      W,SaveF
      TestSkip,BS   PortB:#bHP ;- Ces 5 instructions
      Jump      S$                ;- sont équiv
      Clr       PortA:#bHP  ;-- Not PortA:#bHP
      Jump      T$                ;-
S$:   Set       PortA:#bHP  ;-
 T$:  DecSkip,EQ  CInt
      Jump      F$
 Chaque 256 x 125 x 16 us = ~0.5s
      Not       PortB,W     ; On inverse les bits
      Xor       #MaskHi,W   ; concernés de port B
      Move      W,PortB
F$:
      Move      #256-125,W  ; On recharge le timer
      Move      W,TMR0
      Clr       IntCon:#TOIF ; Flag d'interruption
      Swap      SaveF,W     ; Un Move modifierait F!
      Move      W,F           ; artefact du PIC
      Swap      SaveW         ; se souvenir du truc
      Swap      SaveW,W     ; Un Move modifierait F!
      Retl
```

Program **Initialisations**

```
Deb:
      Move      #DirA,W
      Move      W,TrisA
      Move      #DirB,W
      Move      W,TrisB
      Clr       CInt
      Clr       CntLow      ; Compteur 4 bits à zér
      Move      #MotifHi,W
      Move      W,PortB     ; Initialise le motif osci
      Move      #ModeOption,W
      Move      W,Option
      Move      #-125,W
      Move      W,TMR0
      Move      #ModeIntCon,W ; GIE et TOIE activés
      Move      W,IntCon
Loop:
      Inc       CntLow        ; Tâche du programme
      Move      CntLow,W    ; Compter et afficher sur
      And       #MaskLow,W  ; On ne garde que ce q
      Move      W,CntLow
      Not       PortB,W     ; Diodes allumées à zéro
      And       #MaskHi,W   ; On efface la partie co
      Or        CntLow,W    ; On remet la nouvelle v
      Xor       #-1,W         ; Inverser avant affich
      Move      W,PortB     ; C'est affiché
; Boucle d'attente de 65 ms
  A$: DecSkip,EQ  C1
      Jump      A$
      DecSkip,EQ  C2
      Jump      A$

      Jump      Loop
.End
```

Le programme ci-dessus montre de plus comment réinitialiser le timer à 1ms, et comment agir sur le même port à la fois dans la routine d'interruption et dans le programme principal. Un Set/Clr bit peut être utilisé pour modifier un seul bit; dans le cas d'un moteur, il est parfois préférable de modifier un groupe de bits ensemble, ou un bit après l'autre pour commuter les transistors dans le bon ordre.

Le registre OPTION contient encore deux bits associés au port B. Le bit 7, $\overline{\text{RBPU}}$ commande les "weak pull-ups" sur RB7..0 (un 1 désactive) et le bit 6 INTEDG sélectionne le front montant d'un signal sur RB0, si ces sources d'interruptions sont activées.

7.3. Timer comme compteur d'événements extérieurs

Le Timer est programmable avec les bits RTS (Timer source), PSA (Prescaler assignement) et RTE (Timer Edge) pour compter des événements extérieurs et non pas l'oscillateur interne. Le prédiviseur divise par 2 .. 256 ces impulsions dont le front actif peut être choisi. Le programme peut initialiser ce compteur et le lire en tout temps. TOIF s'active quand le compteur TMR0 dépasse sa capacité.

Le WDT (watchdog timer) est un autre compteur qui permet de réveiller/redémarrer le processeur (voir la documentation du fabricant).

7.4. Autres spécialités du PIC

RB0 peut jouer le rôle d'une entrée d'interruption. Le bit 6 IntEdg (Interrupt edge) du registre Option définit si ces interruptions sont déclenchées par un front montant ou descendant. Le bit RBIE (Interrupt Enable bit) du registre IntCon les autorise (si GIE est activé) et la routine d'interruption à l'adresse 4 est appelée lorsque le sémaphore INTF (Interrupt Flag) est activé. Si l'interruption du timer est aussi activée, il faudra trier (polling) pour exécuter la bonne tâche).

Le registre OPTION contient encore deux bits associés au port B. Le bit 7, $\overline{\text{RBPU}}$ commande les "weak pull-ups" sur RB7..0 (un 1 désactive). Ceci veut dire que si l'on veut brancher des interrupteurs sur le PIC, il faut de préférence les mettre sur le port B avec un contact à la masse; ceci évite d'ajouter des résistances de tirage vers le plus.

Une autre possibilité d'interruption est prévue sur les broches RB7..4 en entrées, pour faciliter la programmation d'un clavier. Un changement de l'état de ces 4 lignes active le sémaphore RBIF. Si le bit RBIE est activé, ainsi que GIE, il y a interruption.

7.5. Programmation multitâche

On veut le plus souvent que le programme principal ne s'occupe pas sans cesse de certaines tâches. Les interruptions et la technique de programmation synchrone (seule utilisable avec les PIC 12C508 etc.) sont la solution.

Si une interruption prévue par le processeur est mise en service, le processeur saute à l'adresse 4 et, après sauvetage de F,W et PClatH, on teste les flags d'interruption pour servir les demandes en suspens. Le timer est souvent utilisé pour générer une interruption toutes les 10 ms par exemple, et aller surveiller des actions qui ne créent pas d'interruption.

Si les interruptions ne sont pas possibles, le timer permet de synchroniser et séquencer les évènements (www.didel.com/doc/DopiSync en anglais).

La gestion d'actions en parallèle (multitâche) est en fait une gestion en série avec des délais aussi court que possible. On teste et agit sur des sémaphores (flags) pour savoir si une action attendue s'est produite, ou si une action lancée est terminée.

Par exemple, si on doit surveiller un poussoir, la technique simple est des tester si ce poussoir est actif, voire attendre dans une boucle qu'il le soit. En mulitâche, une tâche qui se répète assez souvent (interruption du timer) teste la touche et active un flag si la touche est pressée. Dans le programme principal, on teste ce flag, image de la touche, éventuellement idéalisé par la routine multitâche (pas de rebonds de contacts, code Ascii si c'est un clavier, etc.).

Si la tâche est d´attendre une seconde, et si les interruptions du timer ont lieu toutes les 10ms, un compteur dans la routine d´interruption active un flag toutes les les fois qu´il arrive à 100. Dans le programme principal, on teste ce flag quand on veut savoir si si ce temps est écoulé. On réinitialise le compteur et on met à zéro le flag quand on veut démarret le délais de 1 seconde.

Chaque problème multitâche demande une solution spécifique. Il faut bien connaître le processeur, écrire avec peu de fautes d´inattentions et avoir une bonne habitude de la mise au point avant de ce lancer dans des applications complexes.