13. Le Timer 0

Dans ce chapitre, nous allons parler temporisations et comptages. La 16F84 ne comporte qu'un seul timer sur 8 bits, contrairement à d'autres PICs de la famille (comme la 16F876). Si on examine attentivement le fonctionnement du timer0, nous verrons qu'il s'agit en fait d'un compteur.

13.1 Les différents modes de fonctionnement

Nous avons vu que le timer0 est en fait un compteur. Mais que compte-t-il ? Et bien, vous avez deux possibilités.

- En premier lieu, vous pouvez compter les impulsions reçues sur la pin RA4/TOKI. Nous dirons dans ce cas que nous sommes en mode compteur

- Vous pouvez aussi décider de compter les cycles d'horloge de la PIC elle-même. Dans ce cas, comme l'horloge est fixe, nous compterons donc en réalité du temps. Donc, nous serons en mode « timer ».

La sélection d'un ou l'autre de ces deux modes de fonctionnement s'effectue par le bit 5 du registre OPTION : T0CS pour Tmr0 Clock Source select bit.

T0CS = 1 : Fonctionnement en mode compteur
T0CS = 0 : Fonctionnement en mode timer

Dans le cas où vous décidez de travailler en mode compteur, vous devez aussi préciser lors de quelle transisition de niveau le comptage est effectué. Ceci est précisé grâce au bit 4 du registre OPTION : T0SE pour Timer0 Source Edge select bit.

T0SE = 0 : comptage si l'entrée RA4/TOKI passe de 0 à 1
T0SE = 1 : comptage si l'entrée RA4/TOKI passe de 1 à 0

13.2 Le registre tmr0

Ce registre, qui se localise à l'adresse 0x01 en banque0, contient tout simplement la valeur actuelle du timer0. Vous pouvez écrire ou lire tmr0. Si par exemple vous avez configuré tmr0 en compteur, la lecture du registre tmr0 vous donnera le nombre d'événements survenus sur la pin RA4/TOKI.

13.3 Les méthodes d'utilisation du timer0

Comment utiliser le timer0, et quelles sont les possibilités offertes à ce niveau, voilà de quoi nous allons parler ici.

13.3.1 Le mode de lecture simple

La première méthode qui vient à l'esprit est la suivante : Nous lisons le registre tmr0 pour voir ce qu'il contient. La valeur lue est le reflet du nombre d'événements survenus, en

prenant garde au fait que le tmr0 ne peut compter que jusque 255. En cas de dépassement, le tmr0 recommence à 0. C'est donc à vous de gérer cette possibilité.
Petit exemple :

```
        Clrf        tmr0            ; début du comptage
        ; ici un certain nombre d'instructions
        movf        tmr0 , w        ; charger valeur de comptage
        movwf       mavariable      ; sauver pour traitement ultérieur
```

13.3.2 Le mode de scrutation du flag

Nous devons savoir à ce niveau, que tout débordement du timer0 (passage de 0xFF à 0x00) entraîne le positionnement du flag T0IF du registre INTCON. Vous pouvez donc utiliser ce flag pour déterminer si vous avez eu débordement du timer0, ou, en d'autres termes, si le temps programmé est écoulé. Cette méthode à l'inconvénient de vous faire perdre du temps inutilement

Petit exemple :

```
        clrf        tmr0                ; début du comptage
        bcf         INTCON , T0IF       ; effacement du flag
loop
        btfss       INTCON , T0IF       ; tester si compteur a débordé
        goto        loop                ; non, attendre débordement
        ; suite du programme            ; oui, poursuivre : 256 événements écoulés
```

Mais vous pourriez vous dire que vous ne désirez pas forcément attendre 256 incrémentations de tmr0. Supposons que vous désiriez attendre 100 incrémentations. Il suffit dans ce cas de placer dans tmr0 une valeur telle que 100 incrémentations plus tard, tmr0 déborde.

exemple

```
        movlw       256-100         ; charger 256 – 100
        movwf       tmr0            ; initialiser tmr0
        bcf     INTCON , T0IF       ; effacement du flag
loop
        btfss   INTCON , T0IF       ; tester si compteur a débordé
        goto    loop                ; non, attendre débordement
        ; suite du programme        ; oui, poursuivre : 100 événements écoulés
```

13.3.3 Le mode d'interruption

C'est évidemment le mode principal d'utilisation du timer0. En effet, lorsque T0IE est positionné dans le registre INTCON, chaque fois que le flag T0IF passe à 1, une interruption est générée. La procédure à utiliser est celle vue dans la leçon sur les interruptions.

13.3.4 Les méthodes combinées

Supposons que vous vouliez, par exemple, mesurer un temps entre 2 impulsions sur le broche RB0. Supposons également que ce temps soit tel que plusieurs débordements du tmr0 puissent avoir lieu. Une méthode simple de mesure du temps serait la suivante :

1) A la première impulsion sur RB0, on lance le timer 0 en mode interruptions.
2) A chaque interruption de tmr0, on incrémente une variable
3) A la seconde interruption de RB0, on lit tmr0 et on arrête les interruptions
4) Le temps total sera donc (256*variable)+tmr0

On a donc utilisé les interruptions pour les multiples de 256, et la lecture directe de tmr0 pour les « unités ».

13.4 Le prédiviseur

Supposons que nous travaillons avec un quartz de 4MHz. Nous avons donc dans ce cas (4000000/4) = 1.000.000 de cycles par seconde. Chaque cycle d'horloge dure donc 1/1000000ème de seconde, soit 1µs.

Si nous décidons d'utiliser le timer0 dans sa fonction timer et en mode interruptions. Nous aurons donc une interruption toutes les 256µs, soit à peut près toutes les quarts de millième de seconde.

Si nous désirons réaliser une LED clignotante à une fréquence de +- 1Hz, nous aurons besoin d'une temporisation de 500ms, soit 2000 fois plus. Ce n'est donc pas pratique. Nous disposons pour améliorer ceci d'un PREDIVISEUR .

Qu'est-ce donc ? Et bien, tout simplement un diviseur d'événements situé AVANT l'entrée de comptage du timer0. Nous pourrons donc décider d'avoir incrémentation de tmr0 tous les 2 événements par exemple, ou encore tous les 64 événements.

Regardez tableau de la page 16. Vous voyez en bas le tableau des bits PS0 à PS2 du registre OPTION qui déterminent la valeur du prédiviseur.

Ces valeurs varient, pour le timer0, entre 2 et 256. Le bit PSA, quand à lui, détermine si le prédiviseur est affecté au timer0 ou au watchdog. Voici un tableau exprimant toutes les possibilités de ces bits :

PSA	PS2	PS1	PS0	/tmr0	/WD	Temps tmr0	Temps typique Watchdog (minimal)
0	0	0	0	2	1	512 µs	18 ms (7ms)
0	0	0	1	4	1	1024 µs	18 ms (7ms)
0	0	1	0	8	1	2048 µs	18 ms (7ms)
0	0	1	1	16	1	4096 µs	18 ms (7ms)
0	1	0	0	32	1	8192 µs	18 ms (7ms)
0	1	0	1	64	1	16384 µs	18 ms (7ms)
0	1	1	0	128	1	32768 µs	18 ms (7ms)
0	1	1	1	256	1	65536 µs	18 ms (7ms)

1	0	0	0	1	1	256 µs	18 ms (7ms)
1	0	0	1	1	2	256 µs	36 ms (14 ms)
1	0	1	0	1	4	256 µs	72 ms (28 ms)
1	0	1	1	1	8	256 µs	144 ms (56 ms)
1	1	0	0	1	16	256 µs	288 ms (112 ms)
1	1	0	1	1	32	256 µs	576 ms (224 ms)
1	1	1	0	1	64	256 µs	1,152 Sec (448 ms)
1	1	1	1	1	128	256 µs	2,304 Sec (996 ms)

- PSA à PS0 sont les bits de configuration du prédiviseur
- /tmr0 indique la valeur du prédiviseur résultante sur le timer0
- /WD indique la valeur du prédiviseur résultante sur le Watchdog
- temps tmr0 indique le temps max entre 2 interruptions tmr0 avec quartz de 4MHz
- Temps watchdog indique le temps typique disponible entre 2 reset watchdog (indépendant du quartz utilisé). La valeur entre parenthèses indique le temps minimal, qui est celui à utiliser pour faire face à toutes les circonstances.

Remarques importantes :

- Il n'y a qu'un prédiviseur, qui peut être affecté au choix au timer du watchdog (que nous verrons plus tard) ou au timer0. Il ne peut être affecté aux deux en même temps.

- Il n'existe pas de prédiviseur = 1 pour le timer0. Si vous ne voulez pas utiliser le prédiviseur, vous devez donc impérativement le sélectionner sur le watchdog avec une valeur de 1 (ligne jaune du tableau).

- La valeur contenue dans le prédiviseur n'est pas accessible. Par exemple, si vous décidez d'utiliser un prédiviseur de 64, et qu'il y a un moment donné 30 événements déjà survenus, vous n'avez aucun moyen de le savoir. Le prédiviseur limite donc la précision en cas de lecture directe.

- L'écriture dans le registre tmr0 efface le contenu du prédiviseur. Les événements survenus au niveau du prédiviseur sont donc perdus.

13.5 Application pratique du timer0

Nous allons mettre en œuvre notre tmr0 dans une première application pratique. Reprenons donc notre premier exercice, à savoir, faire clignoter une LED à la fréquence approximative de 1Hz.

13.5.1 Préparations

Faites un copier/coller de votre nouveau fichier m16f84.asm et renommez cette copie « Led_tmr.asm ». Relancez MPLAB et créez un nouveau projet intitulé « Led_tmr.pjt ». Ajoutez-y votre nœud « Led_tmr.asm ».

Créez votre en-tête (je continue d'insister)

```
;*****************************************************************
;                                                                *
; Fait clignoter une LED à une fréquence approximative de 1Hz    *
;                                                                *
;*****************************************************************
;                                                                *
;   NOM: LED CLIGNOTANTE AVEC TIMER0                             *
;   Date: 17/02/2001                                             *
;   Version: 1.0                                                 *
;   Circuit: Platine d'essai                                     *
;   Auteur: Bigonoff                                             *
;                                                                *
;*****************************************************************
;                                                                *
;   Fichier requis: P16F84.inc                                   *
;                                                                *
;*****************************************************************
;                                                                *
;   Notes: Utilisation didactique du tmr0 en mode interruption   *
;*****************************************************************
```

Définissez ensuite les _CONFIG en plaçant le watch-dog hors service.

Calculons ensuite le nombre de débordement de tmr0 nécessaires. Nous avons besoin d'une temporisation de 500 ms, soit 500.000µs. Le timer0 génère, sans prédiviseur, une interruption toutes les 256µs . Nous allons donc utiliser le prédivideur. Si nous prenons la plus grande valeur disponible, soit 256, nous aurons donc une interruption toutes les (256*256) = 65536µs.

Nous devrons donc passer (500.000/65536) = 7,63 fois dans notre routine d'interruption. Comme nous ne pouvons pas passer un nombre décimal de fois, nous choisirons 7 ou 8 fois, suivant que nous acceptons une erreur dans un sens ou dans l'autre. Notez que si vous passez 7 fois, vous aurez compté trop peu de temps, il sera toujours possible d'allonger ce temps. Dans le cas contraire, vous aurez trop attendu , donc plus de correction possible.

Il est évident que l'acceptation d'une erreur est fonction de l'application. Si vous désirez faire clignoter une guirlande de Noël, l'erreur de timing sera dérisoire. Si par contre vous désirez construire un chronomètre, une telle erreur sera inacceptable. Commençons donc par ignorer l'erreur.

Nous allons décider d'utiliser une prédivision de 256 avec 7 passages dans la routine d'interruption. Le temps obtenu sera donc en réalité de (256*256*7) = 458752 µs au lieu de nos 500.000µs théoriques.

En reprenant notre tableau page16 sur le contenu du registre OPTION, nous devrons donc initialiser celui-ci avec : B'10000111', soit 0x87. En effet, résistances de rappel hors-service (on n'en n'a pas besoin), source timer0 en interne et prédiviseur sur timer0 avec valeur 256. Nous obtenons donc :

```
OPTIONVAL EQU   H'0087'          ; Valeur registre option
                                 ; Résistance pull-up OFF
                                 ; Préscaler timer à 256
```

Ensuite nous devons déterminer la valeur à placer dans le registre INTCON pour obtenir les interruptions sur le timer0. Ce sera B'10100000', soit 0xA0

```
INTERMASKEQU    H'00A0' ; Interruptions sur tmr0
```

Ensuite, nos définitions :

```
;*****************************************************************
;                       DEFINE                                    *
;*****************************************************************

#DEFINE LED         PORTA,2             ; LED
```

Ne touchons pas à notre routine d'interruption principale, car nous avons suffisamment de place pour conserver nos tests. Ecrivons donc notre routine d'interruption timer. Nous voyons tout d'abord que nous allons devoir compter les passages dans tmr0, nous allons donc avoir besoin d'une variable. Déclarons-la dans la zone 0X0C.

```
        cmpt : 1                ; compteur de passage
```

13.5.2 L'initialisation

Comme il est plus facile de détecter une valeur égale à 0 qu'à 7, nous décrémenterons donc notre variable de 7 à 0. Nous inverserons la LED une fois la valeur 0 atteinte. Nous devons donc intialiser notre variable à 7 pour le premier passage.

Nous effectuerons ceci dans la routine d'intialisation, avant le goto start. Profitons-en également pour placer notre port LED en sortie.
Nous obtenons donc :

```
;*****************************************************************
;                   INITIALISATIONS                               *
;*****************************************************************

init
        clrf        PORTA           ; Sorties portA à 0
        clrf        PORTB           ; sorties portB à 0
        BANK1                       ; passer banque1
        clrf        EEADR           ; permet de diminuer la consommation
        movlw       OPTIONVAL       ; charger masque
        movwf       OPTION_REG      ; initialiser registre option

                    ; Effacer RAM
                    ; ------------
        movlw       0x0c            ; initialisation pointeur
        movwf       FSR             ; pointeur d'adressage indirect
init1
```

```
clrf        INDF        ; effacer ram
incf        FSR,f       ; pointer sur suivant
btfss       FSR,6       ; tester si fin zone atteinte (>=0x40)
goto        init1       ; non, boucler
btfss       FSR,4       ; tester si fin zone atteinte (>=0x50)
goto        init1       ; non, boucler

            ; initialiser ports
            ; -----------------
bcf         LED         ; passer LED en sortie
BANK0                   ; passer banque0

movlw       INTERMASK   ; masque interruption
movwf       INTCON      ; charger interrupt control

        ; initialisations variables
        ; ------------------------
movlw       7           ; charger 7
movwf       cmpt        ; initialiser compteur de passages

goto        start       ; sauter programme principal
```

13.5.3 La routine d'interruption

Réalisons donc maintenant notre routine d'interruption :

Tout d'abord, on décrémente notre compteur de passage, s'il n'est pas nul, on n'a rien à faire cette fois.

```
decfsz  cmpt , f        ; décrémenter compteur de passages
return                  ; pas 0, on ne fait rien
```

Ensuite, si le résultat est nul , nous devons inverser la LED et recharger 7 dans le compteur de passages. Voici le résultat final :

```
;*****************************************************************
;               INTERRUPTION TIMER 0                            *
;*****************************************************************
inttimer
        decfsz      cmpt , f    ; décrémenter compteur de passages
        return                  ; pas 0, on ne fait rien
        BANK0                   ; par précaution
        movlw       b'00000100' ; sélectionner bit à inverser
        xorwf       PORTA , f   ; inverser LED
        movlw       7           ; pour 7 nouveaux passages
        movwf       cmpt        ; dans compteur de passages
        return                  ; fin d'interruption timer
```

Il ne nous reste plus qu'à effacer la ligne

```
clrwdt                          ; effacer watch dog
```

du programme principal, puisque le watchdog n'est pas en service.

Compilez votre programme. Nous allons maintenant le passer au simulateur.

N'oubliez pas de mettre le simulateur en service, et ouvrez la fenêtre des registres spéciaux. Avancez ensuite votre programme en pas à pas jusqu'à ce qu'il arrive dans le programme principal.

Remarques

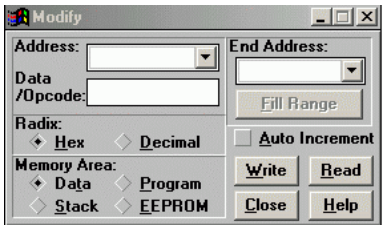
- Le dernier registre en dans la fenêtre des registres spéciaux « T0pre » est en rouge car c'est un registre qui n'existe pas physiquement dans la PIC. C'est MPLAB qui compte les prédivisions pour les besoins de la simultation.

- Chaque fois que « T0pre » atteint la valeur de prédivision, tmr0 est incrémenté de 1. Ce n'est que lorsqu'il débordera que nous aurons une interruption.

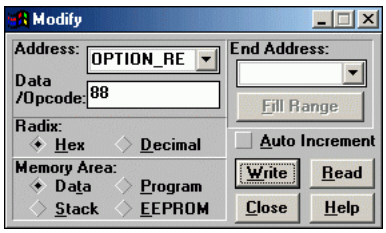
- Dans le programme principal, « T0pre » est incrémenté de 2 unités à chaque pression sur <F7>. C'est normal, car ce programme ne comporte qu'un saut (goto), et chaque saut prend 2 cycles.

13.6 Modification des registres dans le simulateur

Comme nous n'allons pas passer des heures à simuler ce programme, nous allons modifier les registres en cours de simulation. Allez dans le menu « Window » et sélectionnez « Modify… » La fenêtre ci-dessous s'ouvre.



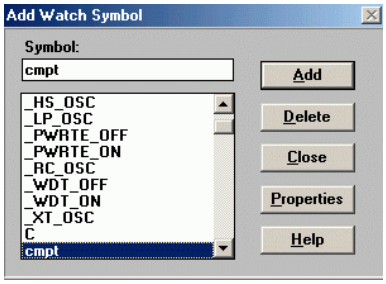
Cette fenêtre vous permet de modifier un registre (ou un emplacement mémoire) en y inscrivant la valeur de votre choix. Nous allons nous servir de cette possibilité. Premièrement, supprimons le prédiviseur. Pour ce faire, nous allons écrire B'10001000', soit 0x88. Choisissez donc OPTION_REG comme adresse, 88 comme data et Hex comme radix. Pressez <Write> et vérifiez que votre registre option est bien passé à 0x88.



Maintenant, chaque pression de <F7> incrémente tmr0 (pas de prédiviseur)

Ouvrez maintenant une fenêtre de visualisation des variables, avec « window->watch->new watch window ». Choisissez d'afficher la variable « cmpt ». Cliquez <Add>.

Continuez de presser <F7> et constatez que le débordement de tmr0 provoque une interruption et que cette interruption provoque la décrémentation de cmpt. Pour ne pas attendre trop longtemps, servez-vous de la fenêtre « modify » pour positionner cmpt à 1. Ensuite, poursuivez la simulation. Vous constaterez que la prochaine interruption provoque la modification de RA2.



13.7 Mise en place sur la platine d'essais

Chargez le fichier .hex obtenu dans votre PIC et alimentez votre platine d'essais. Comptez les allumages de la LED obtenus en 1 minute. Vous devriez trouver aux alentours de 65/66 pulses par minute. Ceci vous montre la précision obtenue.

En réalité, un allumage toutes les (256*256*7*2) = 917504µS.

En 1 minute, 60.000.000/917504 = 65,3 allumages. La théorie rejoint la pratique. Le fichier est disponible sous la dénomination « led_tmr1.asm ».

13.8 Première amélioration de la précision

Nous allons chercher à améliorer la précision de notre programme. Nous pouvons commencer par modifier notre prédiviseur. Essayons plusieurs valeurs successives :

/1 : donne 500000/256 = 1953,125 passages. Pas pratique
/2 : donne 500000/512 = 976,5625 passages. Pas plus pratique
/4 : donne 500000/1024 = 488,28125 passages . Idem
/8 : donne 500000/2048 = 244,140625 passages. Dans ce cas, un seul compteur est également nécessaire, car le nombre de passages est inférieur à 256.

Quelle va être la précision obtenue ? Et bien, nous initialiserons cmpt à 244, avec prédiviseur à 8. Dans ce cas, la durée obtenue sera de :

256*8*244 = 499712 µs, donc 499712*2 = 999424µs par allumage.

En une minute, nous aurons donc 60000000/999424 = 60,034 allumages. Voici donc une précision nettement meilleure.

Vous pouvez maintenant modifier vous-même votre programme selon ces indications. Vous voyez que vous devez modifier la valeur 07 en 244 à 2 endroits. Ce n'est pas pratique. Ajoutez donc une assignation, par exemple

```
TIMEBASE    EQU    D'244' ; base de temps = 244 décimal
```

Si vous avez un problème, le fichier fonctionnel est disponible sous « Led_tmr.asm ».

- Avantages obtenus : Une plus grande précision

- Inconvénient : plus d'interruptions générées, donc plus de temps perdu pour le programme principal. Dans notre cas, cela n'a pas d'importance, le programme ne fait rien d'autre, mais ce ne sera pas toujours le cas.

13.9 Seconde amélioration de la précision

Vous pouvez encore améliorer la précision de votre programme. En effet, vous pouvez ne pas utiliser de prédiviseur, donc utiliser plusieurs compteurs pour 1953,125 passages. Au 1953ème passage, vous pourrez même générer une dernière tempo en ajoutant une valeur au tmr0. Par exemple :

- On détecte 1953 passages à l'aide de plusieurs compteurs

- Lors du 1953ème passage, on en est à 1953*256 = 499968 µs, il nous manque donc : 500.000 - 499.968 = 32µs.

- On ajoute donc 256-32 = 224 à tmr0, de la manière suivante :

```
movlw    224
addwf    tmr0
```

Bien entendu, 32µs pour tout réaliser, c'est très court, aussi nous devrons optimiser les routines d'interruption au maximum. Suppression des tests et des sous-programmes etc. Mais cela reste en général à la limite du possible. Nous ne traiterons pas ce procédé ici, car cela ne présente pas d'intérêt, d'autant que les interruptions vont finir par occuper la majorité du temps CPU.

13.10 La bonne méthode - Adaptation de l'horloge

Supposons que vous voulez construire un chronomètre. La précision est la donnée la plus importante dans ce cas, et passe bien avant la vitesse. Nous allons donc nous arranger pour que les démultiplicateurs tombent sur des multiples entiers. Comment ? Et bien simplement en changeant le temps d'une instruction, donc, en changeant le quartz de la PIC.

Exemple :

Comme nous ne pouvons pas accélérer une PIC au dessus de sa vitesse maximale (nous utilisons une PIC 4MHz), nous pouvons seulement la ralentir. Nous partons donc d'une base de temps trop rapide.

Par exemple : reprenons notre cas de départ : prédiviseur à 256, compteur de passages à 7. Durée avec un quartz de 4MHz : 256*256*7 par tempo, donc 256*256*7*2 par allumage, soit 917504 µs. Or, nous désirons 1000000µs.

Il suffit donc de recalculer à l'envers :

Que doit durer une instruction ? 1000000/(256*256*7*2) = 1,089913504µs.

Cela nous donne donc une fréquence d'instructions de 1/1,089913504µs = 0,917504 MHz.

Comme la fréquence des cycles internes est égale à la fréquence du quartz/4, nous aurons donc besoin d'un quartz de 0,917504 * 4 = 3,670016 MHz. (MHz car nous avons divisé par des µs : or, diviser par un millionième revient à multiplier par un million).

La seule contrainte est donc de savoir s'il existe des quartz de cette fréquence disponibles dans le commerce. Dans le cas contraire, vous recommencez vos calculs avec d'autres valeurs de prédiviseurs et de compteur.

Si vous trouvez donc un quartz de fréquence appropriée, vous obtenez une horloge de la précision de votre quartz. Vous ajoutez un affichage, et voilà une horloge à quartz.

13.11 La méthode de luxe : La double horloge

La méthode précédente présente l'inconvénient de ralentir la PIC. Que faire si vous voulez à la fois une vitesse maximale et une précision également maximale ? Et bien, aucun problème.

Vous alimentez votre PIC avec votre quartz et vous créez un autre oscillateur externe avec votre quartz spécial timing. Vous appliquez le signal obtenu sur la pin RA4/TOKI et vous configurez votre timer0 en mode compteur.

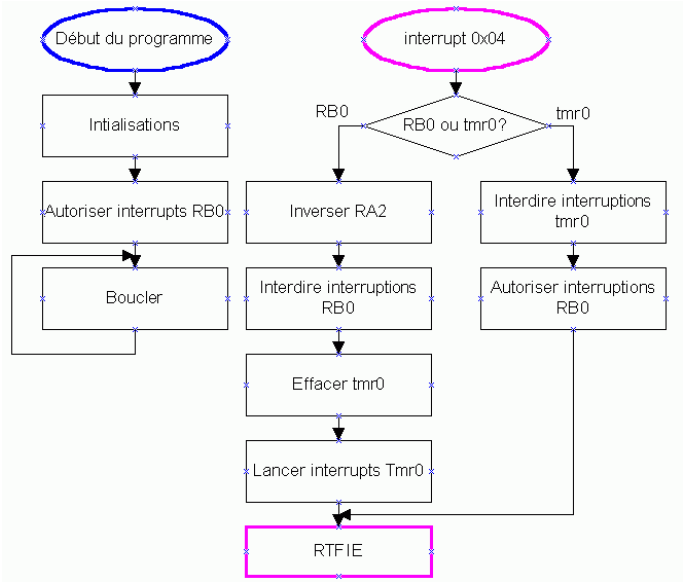
Donc, votre PIC tourne à vitesse maximale, et les interruptions timer0 sont générées par une autre base de temps, plus adaptée à la mesure de vos événements.

13.12 Exemple d'utilisation de 2 interruptions

Dans ce petit exemple nous allons utiliser 2 sources d'interruption différentes, afin de vous montrer un exemple concret de ce type d'utilisation. Nous allons recréer notre programme de télérupteur, mais en remplaçant la temporisation par une interruption sur le timer0.

Remarquez que notre programme principal ne fait plus rien. Vous pouvez donc utiliser d'autres possibilités sur cette carte sans perturber le fonctionnement du télérupteur.

Nous aurons donc une interruption pour RB0, et une autre pour tmr0. Vous voyez ci-dessous l'ordinogramme qui va nous servir.



Effectuez une copie de votre fichier m16f84.asm et renommez-le « telerupt.asm ». Créez un nouveau projet « telerupt.pjt ». Editez votre fichier comme précédemment : coupure du watchdog, positionnement de la LED en sortie, mise en service initiale des interruptions RB0/INT.

Créez votre routine d'interruption timer0 toutes les 260ms, soit prédiviseur à 256, et 4 passages .

Essayez de réaliser vous-même ce programme. Chargez-le dans votre PIC et lancez-le. Notez que l'ordinogramme ne contient pas le compteur de passages dans tmr0. Je vous laisse le soin de réfléchir.

Une pression sur le B .P. allume la LED, une autre l'éteint. Si cela ne fonctionne pas, cherchez l'erreur ou servez-vous du simultateur. Je vous fourni le programme fonctionnel dans le cas où vous seriez bloqués.

Remarque

Il est très important de bien comprendre qu'il faut effacer tmr0 AVANT d'effacer le flag T0IF et de relancer les interruptions tmr0. En effet, si vous faites le contraire, vous risquez que tmr0 déborde entre le moment de l'effacement du flag et le moment de l'effacement de tmr0. Dans ce cas le flag serait remis immédiatement après l'avoir effacé. Votre programme pourrait donc avoir des ratés par intermittence.

13.13 Conclusion

Vous savez maintenant exploiter le timer0. Les méthodes évoquées ici sont une base de travail pour des applications plus sérieuses. Je vous conseille vraiment d'effectuer toutes les manipulations évoquées. Même les erreurs vous seront profitables.