

PICGénial – Microcontrôleur PIC 16F84/16F870

Introduction à la programmation avec le PICGénial – suite

3. Boucles d'attente, sons et clignotements

3.1. Oscillations de quelque kHz

Notre premier programme génère un son, puis nous ferons clignoter les LEDs sur le port B. Dans les deux cas, il faut mettre à 1, puis mettre à zéro les bits du port. La solution directe, mais peu élégante est d'écrire:

```
Boucle: Move    #2'10101010,W (ou Move #0,W)
        Move    W,PortB
        ... attente
        Move    #2'01010101,W (ou Move #2'11111111,W)
        Move    W,PortB
        ... attente
        Jump    Boucle
```

Si le but est de faire clignoter toutes les sorties (ici en alternance), l'instruction NOT, qui s'applique sur les variables et sur les 2 ports, inverse tous les bits:

```
        Move    #2'10101010,W
        Move    W,PortB
Boucle: Not    PortB
        ... attente
        Jump    Boucle
```

Sans attente, le port B va osciller à 330 KiloHerz. En effet, le processeur a une horloge 4 MHz, chaque instruction prend 4 coups d'horloge, donc 1 microseconde, sauf les sauts qui prennent le double. La boucle dure donc 3 microsecondes. On peut l'allonger à 4 microsecondes en insérant l'instruction Nop (no-operation), mais il faudrait insérer beaucoup trop de Nop avant de pouvoir clignoter à quelques Hz.

Pour perdre du temps, on fait une boucle d'attente, avec une variable compteur, ou de préférence décompteur. Tous les processeurs savent bien reconnaître quand un registre ou une variable passe à zéro: le fanion Z s'active. Le PIC ne connaît pas les sauts conditionnels, mais comme toutes les instructions ont la même longueur, il peut sauter conditionnellement par-dessus une instruction, et c'est souvent très efficace. Une boucle qui se répète 100 fois (et dure 400 μ s car elle prend 4 cycles processeurs) s'écrit:

```
Boucle: Move    #100,W
        Move    W,C1          ; Variable décompteur, à déclarer au début
B$:     Dec     C1            ; Décrémente la valeur dans C1
        Skip,EQ             ; Skip si le résultat précédent est égal à zéro
        Jump   B$           ; Si non, on passe ici et on recommence
        ... ce que l'on doit faire toutes les 400  $\mu$ s
        Jump   Boucle
```

Nous utiliserons souvent les étiquettes locales terminées par un signe dollar dans des modules de programme. Les étiquettes sans dollars caractérisent l'entrée dans des blocs de programme, des procédures, et doivent être toutes différentes. Les étiquettes avec dollars sont locales et le même nom peut être utilisé une fois dans chaque bloc précédé d'une adresse absolue (sans \$).

Au lieu de décompter, on peut naturellement compter, après avoir initialisé la variable à la différence:

```
Boucle: Move    #256-100,W    ; #-100,W revient au même
        Move    W,C1          ; Une variable compteur, à déclarer au début
B$:     Inc     C1
        Skip,EQ
        Jump   B$
        ... ce que l'on doit faire toutes les 400  $\mu$ s
        Jump   Boucle
```

La durée de la boucle B\$ est 4 microsecondes, donc la grande boucle dure 400+4 microsecondes, plus le temps pour ce que l'on doit faire tous les 400 microsecondes et quelques. Si on ajoute l'instruction "Not PortC" et si le port C est initialisé en sortie

(donc le bit RC4 connecté au buzzer), on entendra un son continu à 2,5 kHz. Le programme complet (sauf les pseudo-instructions initiales et la déclaration de C1) est:

```

Deb:   Move    #0,W           ; Clr W revient au même
       Move    W,TrisC       ; port A en sortie (ne pas agir sur les poussoirs)
Boucle: Move   #100,W
       Move    W,C1          ; Une variable compteur, à déclarer au début
B$:    Dec     C1            ; On décrémente C1
       Skip,EQ              ; Skip si le résultat précédent est égal à zéro
       Jump   B$            ; Si non, on passe ici et on recommence
       Not    PortC
       Jump   Boucle

```

Pour un retard maximum, l'initialisation est plus simple:

```

Boucle: Clr     C1
B$:     Dec     C1
       Skip,EQ
       Jump   B$
       ... ce que l'on doit faire toutes les ms
       Jump   Boucle

```

La première fois que l'on entre dans la boucle après initialisation, le décompteur passe de 0 à -1, égal à 255 décimal ou 2'11111111 binaire. Il faut faire 256 fois la boucle pour arriver à zéro et sortir de la boucle B\$. La valeur finale est zéro.

On peut encore se demander si l'instruction Clr C1 est nécessaire. Si on l'enlève, la variable C1 a, la première fois que l'on entre dans la boucle, une valeur inconnue. La boucle n'aura donc pas pour cette première boucle la durée maximale, ce qui ne gêne pas dans le cas d'un buzzer ou d'une lampe qui clignote.

Etant donné que l'on fait souvent des boucles pour répéter des opérations, et que l'on ne voudrait pas perdre trop de temps dans les instructions nécessaires pour répéter et tester la fin de boucle, le PIC a une instruction spéciale "Decrement and Skip if Equal" (appelée decfsz dans l'assembleur Microchip). Notre boucle d'attente peut donc s'écrire en 2 lignes:

```

B$:    DecSkip,EQ C1
       Jump   B$

```

Chaque boucle dure une microseconde de moins, ce qui réduit le délai maximal que l'on peut atteindre (3 x 256 = 768 microsecondes).

Le programme le plus simple que l'on peut écrire pour générer un son avec notre carte est:

<pre> Program PicgT1 .Son continu sur un buzzer connecté sur une ligne du port C .Proc 16F870 .Ref 16F870 Variables Variables .Loc DebVar C1: .16 1 </pre>	<pre> .Loc 0 Program Début Debut: Move #0,W ; Tout en sortie Move W,TrisC Boucle: Not PortC B\$: DecSkip,EQ C1 Jump B\$ Jump Boucle .End </pre>
---	---

Ce programme doit s'assembler et se charger sans erreur. A noter que l'état du port C n'est pas initialisé (seule la direction est initialisée). On ne connaît pas l'état initial des LEDs et de la membrane du HP. C1 n'est pas initialisé, et la première demi-période est variable, ce que l'oreille ne peut pas reconnaître. Chaque boucle est exécutée 256 fois, car on sort de la boucle avec la valeur zéro. Quand on y retourne, la valeur est décrétementée (donc prend la valeur -1, codée 16'FF). Chaque boucle décompte FF FE FD ... 2 1 et à zéro on sort de la boucle. La boucle dure 3 microsecondes (1 microseconde pour toutes les instructions, sauf les sauts qui durent 2 microsecondes). Tous les bits du port C, et en particulier le buzzer, oscillent avec une période de 256x3 microsecondes (plus 3 microsecondes pour modifier le port C), soit avec une fréquence de 1,7 kHz. Les diodes lumineuses sont allumées 50% du temps et apparaissent à notre oeil en demi-intensité. Si on veut abaisser un peu la fréquence, il faut faire une boucle qui dure plus longtemps, par exemple en rajoutant des instructions inutiles dans la boucle (instruction Nop signifiant no-opération), ou, comme on le verra plus loin, imbriquer une deuxième boucle d'attente.

Pour augmenter la fréquence, il faut avant d'entrer dans la boucle initialiser C1 avec une valeur inférieure à 256. La boucle sera effectuée moins de fois, donc la fréquence va augmenter. Il faut pour cela rajouter deux instructions juste avant l'étiquette B\$:

```
Move #ValPeriod,W
Move W,C1
```

et déclarer, de préférence au début avant le bloc des variables

```
\const;Choix de la fréquence des sons
Freq = 2000 ; supérieur à 1300 Hz
Period = 1000000/Freq ; Période en microsecondes
ValPeriod = Period/3 ; Boucle de 3 µs
```

Il est plus simple de déclarer ValPeriod = 200 (ou une autre valeur entre 1 et 255) et de calculer la fréquence résultante si nécessaire. Une notation qui dit ce que l'on veut (une fréquence donnée) et laisse à l'assembleur le soin de faire les calculs, est naturellement préférable, même s'il faut réfléchir un peu plus la première fois et vérifier que l'assembleur accepte les expressions données. L'assembleur CALM calcule en nombres entiers 32 bits; Mais si ValPeriod est supérieur à 255, il n'acceptera pas l'instruction "Move #ValPeriod,W" (message "Expression trop grande").

En modifiant Freq et en réassemblant/téléchargeant, on peut après quelques itérations trouver la fréquence de résonance du buzzer (intensité maximale), trouver votre limite d'audition en fréquence, voir si votre chien est sensible aux ultrasons, etc. Il faut chaque fois changer la constante et réassembler; si on avait 8 interrupteurs sur le port B, on pourrait directement lire le port comme variable donnant la période. Puisque nous avons seulement deux poussoirs sur le port C, on peut les utiliser pour incrémenter et décrémenter une variable VarPer, que l'on peut afficher sur le port B pour connaître sa valeur. Si vous voulez que cette valeur change lentement, il faut encore une astuce de retard au bon endroit (on ne lira les poussoirs que toutes les 1000 oscillations du buzzer, ce qui supprime les rebonds de contact par la même occasion). Bravo si vous savez déjà inventer et déverminer un programme de cette complexité! Si vous n'y arrivez pas, une solution est donnée dans le programme PicgT1b.asm

3.2. Sirène

On a vu précédemment comment faire un son continu. Pour une sirène, il faut mettre la période en variable, et modifier cette variable, par exemple toutes les 20 ou 100 périodes, suivant la vitesse de variation voulue. Le programme suivant n'a pas de butées; il passe dans les ultrasons et revient à une période maximale =0 puis =255, etc. La durée de chaque note dépend de la période, puisque l'on joue un nombre fixe de périodes. La variation de fréquence n'est donc pas régulière.

```

Program PicgT4 Sirène
.Proc 16F870
.Ref 16F870
.Loc DebVar

Variables Variables
C1: .Blk.16 1 ; Décompteur fixant la duré
Period: .Blk.16 1 ; Période variable
CRep: .Blk.16 1 ; Compteur de répétition d

Constant Ports
DirC = 0 ; Tout en sortie

Constant Vitesse d'évolution
NbCycles = 20 ; Nombre de cycles avant répétition de l

.Loc 0
Program Début
Debut: Move #DirC,W
Move W,TrisC
Clr Period ; On part avec la période

Boucle:
; On diminue la période, à 0 on recommence
Dec Period
A$: Move #NbCycles,W
Move W,CRep
; On répète NbCycles la même période
R$: Move Period,W
Move W,C1
Not PortA
B$: DecSkip,EQ C1
Jump B$
DecSkip,EQ CRep
Jump R$
Jump Boucle

.End
```

Pour éviter les valeurs qui génèrent des ultrasons (le passage est très bref, car la période est courte), il faut comparer avec une fréquence maximum, donc une période minimale. Puisque l'on décrémente la variable Period, une comparaison d'égalité peut faire sortir de la boucle et réinitialiser avec une valeur correspondant par exemple à la fréquence la plus basse, si une valeur de période inférieure à 256 (codé par un 0) est souhaitée. La

comparaison d'égalité se fait avec une soustraction ou de préférence un ou-exclusif; le résultat est nul s'il y a égalité bit par bit. On écrira par exemple:

```
Dec      Period
Move     #PeriodMin,W
Xor      Period,W ; Egalité?
Skip,NE
Jump     Reinitialise
```

A\$: ...

L'étiquette "Reinitialise:" est en dessus de l'étiquette Boucle: de PicgT4, avec entre deux les instructions "Move #PeriodMax,W" et "Move W,Period"

3.3. Clignotement de lampes

Pour abaisser la fréquence, il suffit de mettre une boucle d'attente dans la boucle d'attente, en utilisant deux variables C1 et C2:

```
B1$:
B2$: DecSkip,EQ C2
      Jump     B2$
      DecSkip,EQ C1
      Jump     B1$
```

Cette boucle va durer $((3 \mu s \times 256) + 3 \mu s) \times 256 \approx 197'000 \mu s$, soit 0.2 secondes. En ajoutant trois Nop dans la boucle centrale, on double le temps d'exécution. Pour des délais longs, on peut ajouter une 3e boucle imbriquée, etc. Evidemment, avec ce petit jeu, le processeur ne fait rien d'autre que décompter, avec très rarement une instruction utile; avec un processeur coûtant quelques francs, ce type de sous-utilisation est toutefois acceptable dans beaucoup d'applications. Mais il nous faudra quand même apprendre à programmer plus intelligemment.

Si la fréquence est basse, on entend un clic dans le buzzer lorsqu'il change d'état, et on peut voir les LEDs clignoter. A une fréquence supérieure à 30Hz, une LED qui clignote apparaît en demi-intensité.

Faisons clignoter les LEDs du port B avec une demi-période de 1 seconde exactement. Pour cela, écrivons d'abord une routine qui dure N fois 10 millisecondes. N est un paramètre qui sera transmis par le registre W. Pour une attente d'une seconde exactement, il suffira donc d'écrire:

```
Move     #100,W
Call     Att10ms
```

L'instruction Call saute à l'adresse indiquée, en mettant sur une pile l'adresse de retour (adresse de l'instruction suivante). Avec le PIC, la pile n'a pas besoin d'être initialisée comme avec d'autres processeurs.

La routine Att10ms doit être formée de deux boucles imbriquées. Elle pourrait appeler 100 fois une boucle de 100 microsecondes, ce que nous savons faire. Ecrivons plutôt cette routine de façon à pouvoir la transformer en une routine qui donne le dixième de seconde, donc des attentes jusqu'à 25 secondes. Une boucle appelle 25 fois une boucle intérieure qui doit donc durer $10000 \mu s / 25$, donc 400 microsecondes. Le coeur de cette boucle durant $3 \mu s$, il faut la répéter $400/3 = 133$ fois:

```
Routine Att10ms Attente 0,01 à 2,55 s
      in: W durée en unités de 10ms
      mod: C1,C2,C3
Att10ms:
      Move     W,C3
      Test     C3          ; Si C3 contient 0, il ne faut pas
      Skip,EQ          ; attendre 2,56 s, mais 7  $\mu s \approx 0$ 
      Ret
C$:   Move     #25,W
      Move     W,C2
B$:   Move     #133,W      ; ajuster pour 10ms précis
      ; si l'oscillateur n'est pas à 4MHz exactement
      Move     W,C1
A$:   DecSkip,EQ C1
      Jump     A$
      DecSkip,EQ C2
      Jump     B$
      DecSkip,EQ C3
      Jump     C$
      Ret
```

La routine Att100ms (attente de 0.1 à 25s) ne diffère que par la ligne "C2\$: Move #250,W".

L'instruction Ret renvoie au programme principal. Une pile spéciale de 8 positions seulement est prévue dans le PIC, et c'est bien assez: on ne fait jamais des appels de routines compliqués en cascade comme avec d'autres processeurs. Contrairement à la majorité des processeurs, on ne peut pas mettre de variables sur la pile.

Remarquons encore que si la durée donnée dans W est zéro, le temps maximal de 2,56 ms sera atteint. Pour avoir une attente minimale si W=0, il faut tester W en entrée de la routine, en écrivant

```
Att10ms:
    Move    W,C3
    Skip,NE
    Ret     ; Retour avec 4 µs de perdues si W=0
    Move    #133,W
    ....
```

Ecrivons un programme qui clignote les diodes avec une période de 1 seconde, mais avec une durée d'allumage en paramètre fixe, que l'on pourra modifier en réassemblant.

```
Program PicGT2 Clignote tout le PortB à 1 Hz,
                durée à 1 selon Duree
.Proc    16F870
.Ref     16F870

Variables Décompteurs
.Loc    DebVar ; Zone libre pour les variables
C1:    .16    1 ; Compteur pour boucles d'
C2:    .16    1
C3:    .16    1

Constant Ports
DirB    = 0
ToutEteint = 2'11111111

Constant Durées impulsions
Periode = 100 ; 1 seconde
Duree   = Periode/10 ; Si on veut raisonner
        ; en pourcentage de temps et non pas en valeur absolue
.Loc    0

Program  
Debut:  Move    #DirB,W ; Tout en sortie
        Move    W,TrisB
        Move    #ToutEteint,W
        Move    W,PortB ; Diodes éteintes

Boucle:
    Not     PortB ; Passe à 0, allume les di
    Move    #Duree,W
    Call    Att10ms
    Not     PortB ; Repasse à un
    Move    #{Periode-Duree},W
    Call    Att10ms
    Jump    Boucle

Routine Att10ms Attente 0,01 à 2,55 s
in: W durée en unités de 10ms
mod: C1,C2,C3
Att10ms:
    Move    W,C3
    Skip,NE
    Ret
C$: Move    #25,W
    Move    W,C2
B$: Move    #133,W ; ajuster pour 10ms précis
    Move    W,C1
A$: DecSkip,EQ C1
    Jump    A$
    DecSkip,EQ C2
    Jump    B$
    DecSkip,EQ C3
    Jump    C$
    Ret
.End
```

En changeant les constantes Periode et Durée, on peut vérifier leur effet.

3.4. Un chenillard

L'instruction "RLC Registre" décale à gauche le registre ou port mentionné. RRC décale à droite. Le décalage est en fait une rotation à travers le bit de "Carry"; on retrouve l'information en place après 9 décalages. En association avec les instructions qui préparent ou testent le Carry, on peut entre autre multiplier ou diviser par 2.

Pour un chenillard, remplacer dans le programme PicGT2.asm l'instruction Not PortB par RRC PortB, après avoir initialisé le port B à la valeur 2'10100000 par exemple. Pour compliquer, on peut tous les 8 décalages, prendre une autre valeur, forcer la valeur du Carry, etc. Comme exemple, le programme PicGT2b remplit et vide le registre d'affichage.