L'horloge à hélice ? Comment ça marche ?

L'horloge à hélice est basée sur le phénomène de persistance rétinienne
(Persistance Of View=POV).
Selon wikipedia,"la **persistance rétinienne** est la capacité ou défaut de l'œil à conserver une image vue superposée aux images que l'on est en train de voir. Elle est plus forte et plus longue si l'image observée est lumineuse. Ceci est dû en partie au temps de traitement biochimique du signal optique. La persistance dure environ 50ms."
Ainsi lors du mouvement de notre barre de led, on peut faire apparaître un point fixe si on allume un bref instant une led et que l'on répète ceci à moins de 50ms d'intervalle (20Hz).
Donc la vitesse minimale d'apparition du phénomène serait donc de 1 tour en 50ms, soit 1t/50ms = 20 tours/seconde = 1200 tours par minute.
En pratique on constate que l'on peut tourner beaucoup moins vite, sans doute car la disparition de l'image n'est pas instantanée après 50ms. A une vitesse plus basse le phénomène est moins lumineux et si on tourne trop lentement un scintillement commence à apparaître. Si on tourne vraiment trop lentement on peut suivre la rotation des LEDs.
C'est pourquoi dans le programme l'affichage de l'horloge effectuée qu'à partir d'une vitesse minimale.
Les explications qui suivent aident à comprendre le fonctionnement général du logiciel mais ne remplacent pas une étude attentive de ce dernier.

Organisation du logiciel :
povS1.h : définitions générales utilisées dans tout le programme.
main.c : on trouve la boucle principale et de nombreuses fonctions de bases utilisées dans tout le programme
interrup.c : traitements réalisés en interruption :
haute priorité : timer 1 gestion du temps heure/minute/seconde (une IT par seconde)
INT0 une interruption par tour référence 0° (LED/phototransistor)
timer0 mesure de la durée d'un tour
timer3 une IT tout les 3° donc 120 IT par tour

basse priorité : réception de caractère sur la liaison série bluetooth

bluetooth.c : traitement des commandes reçu par la liaison bluetooth
temp_data.c : table de correspondance utilisée par la mesure de température

Lorsque le programme démarre il affiche la tension batterie (2V par led à partir de l'extrémité) et la température (18°C + 2°C par Led) si on appuie sur BP2. Si on appuie sur les deux boutons plus de 3 secondes on rentre dans le mode de réglage de l'heure (voir readme.txt).
Si la variable low_speed passe à 0, on passe au fonctionnement en horloge (dit POV).

La fonction BinToLeds() applique une valeur binaire (12bits) sur les LED. Chaque bit à 1 allume la LED de poids binaire correspondant. Cette fonction est systématiquement utilisée par la POV.
Par exemple BinToLeds(0xFFF) allume toutes les LEDs, BinToLeds(0x003) allume les deux LEDs extérieures, BinToLeds(0x801) allume les deux LEDs extrèmes,...

La POV utilise un tableau, LedsArray, de 60 cases qui reflète la valeur des LEDs tout les 6° (6°x60=360). Ce tableau est mis à jour par la partie gestion du temps du programme et il est appliqué sur LED "au bon moment".

Gestion du temps et du tableau LedsArray :

L'heure est stockée dans les variables hour,min et sec.

Le timer1 génère une interruption toutes les 50ms. On compte donc 20 interruptions pour avoir un déclenchement de la partie de programme correspondante toutes les secondes (1s=50msx20).

Chaque seconde on incrémente la variable sec (jusqu'à 59) puis le cas échéant on met à jour les variable hour et min. Puis on met à jour le tableau des LEDs :

```
LedsArray[hour*5]=0x0FF0;
LedsArray[min]=0x0FFE;
```

Les valeurs 0xFF0 et 0xFFE correspondent à la petite et à la grande aiguille.

L'angle de l'aiguille des minutes correspond à l'index dans le tableau (l'aiguille des minutes avance de 6° par minutes), par contre l'aiguille des heures avance de 30° par heure (360°/12h) ce qui correspond à 5 positions dans le tableau (30°=5x6°).

Pour remplir le tour des secondes avec les dernières LED, on ajoute 1 pour allumer la dernière LED : LedsArray[sec]|=1; La position des secondes correspond directement à l'index dans le tableau.

Pour faciliter le repérage, on allume également la LED extérieure toutes les cinq positions pour figurer un repère toute les 5 minutes (ou 5 secondes).

Par exemple pour 3h 37m et 23s le tableau contient :

```
00   000000000001
01   000000000001
02   000000000001
03   000000000001
04   000000000001
05   000000000001
06   000000000001
07   000000000001
08   000000000001
09   000000000001
10   000000000001
11   000000000001
12   000000000001
13   000000000001
14   000000000001
15   111111110001   <-- 3h
16   000000000001
17   000000000001
18   000000000001
19   000000000001
20   000000000001
21   000000000001
22   000000000001
23   000000000001
24   000000000000
25   000000000001
26   000000000000
27   000000000000
28   000000000000
29   000000000000
```

23sec

```
30   000000000001
31   000000000000
32   000000000000
33   000000000000
34   000000000000
35   000000000001
36   000000000000
37   111111111110   <-- 37 min
38   000000000000
39   000000000000
40   000000000001
41   000000000000
42   000000000000
43   000000000000
44   000000000000
45   000000000001
46   000000000000
47   000000000000
48   000000000000
49   000000000000
50   000000000001
51   000000000000
52   000000000000
53   000000000000
54   000000000000
55   000000000001
56   000000000000
57   000000000000
58   000000000000
59   000000000000
```

Affichage du tableau à une position fixe dans l'espace :

Maintenant qu'on a vu comment ce tableau est généré, il s'agit de comprendre comment il est appliqué sur les LED "au bon moment".
On utilise un autre timer, le timer0 pour mesurer la durée d'un tour. Ce timer compte sur 16 bits (donc de 0 à 65535 puis repasse à 0) toutes les 3.2 µs. On obtient donc le débordement au bout de 209,72ms (65536x3.2µs).
A chaque passage du phototransistor devant la LED infrarouge du support, on obtient une interruption, on lit alors la valeur du timer0 puis on le remet à 0. Ainsi à chaque passage la valeur lue correspond au nombre de fronts d'horloge à 3.2µs durant le tour précédent et donc à une mesure de la durée du tour :

durée du tour = (valeur dans le timer0 au moment du passage devant la LED IR) x 3.2µs

On notera n0 cette valeur dans le timer0.
Pour 500tr/min par exemple on a 120ms/tour et donc n0 = 37500.
Si n0 est trop élevée, ou si le timer déborde avant la fin d'un tour, on tourne tourne lentement et la POV n'est pas activée (variable low_speed à 1).
Si n0 a une valeur correcte, on passe en mode POV (low_speed passe à 0).
On utilise alors un troisième timer, le timer3 qui va servir à générer une interruption tout les 3° donc 120 fois par tour (360/3). Pour avoir une meilleure précision, ce timer s'incrémente 32 fois plus vite que le timer0 donc pour 3° il doit avoir compté n3=(32xn0)/120 impulsions. On précharge donc ce timer à (65536-n3) pour avoir un débordement à bout de n3 impulsions.
A chaque interruption de débordement du timer 3, les LEDs ont donc parcouru un angle de 3°.
On a donc 120 interruptions par tour. Pour les IT n° 0 , 2 ,4 , 6 …., soit une IT sur deux ce qui correspond à 6° (2x3°), on applique la valeur du tableau en position 0,1,2,3... sur les LEDs.
A chaque IT impaire on éteint toutes les LED. Donc une LED n'est allumée au maximum que durant un temps correspondant à 3°. Ce qui correspond à "allumer la LED un bref instant" pour avoir l'illusion d'un point fixe. On pourrait diminuer ce temps pour avoir une meilleure résolution spatiale mais c'est un compromis avec la complexité du programme, la luminosité souhaitée et la taille mémoire occupée par le tableau. Dans les futures versions, il est possible que le tableau ait directement 120 cases voire 240.
En réalité, c'est légèrement plus compliqué. Il est absolument nécessaire qu'à la fin d'un tour on soit arrivé exactement à l'index 59 dans le tableau. Si ce n'est pas le cas les dernières valeurs, entre 55 et 59 minutes (ou secondes) par exemple, ne seront pas affichées car au début du tour suivant on reprend toujours à l'index 0. L'index 59 correspond à 119 interruptions à 3° (2x59 =118 plus 1 interruption pour éteindre les LEDs de la position 59) . Le moment de la fin du tour correspond à la 120ième interruption. Or les temps de calcul entre les mises à jour des timers ne sont pas négligeables et induisent un léger décalage que l'on doit compenser. En effet le microcontrôleur travaille à 40MHz et chaque instruction élémentaire dure 0.1µs (4 cycles à 40MHz). Le timer 3 s'incrémentant toutes les 0.1µs (3.2µs/32), chaque instruction exécutée entre les rechargements (calculs, test, etc..) impacte directement la précision (on pourrait utiliser le module CCP du PIC en mode output compare par pallier à ce problème). Ici, pour être certain d'être toujours en phase, on incrémente ou décrémente une variable (delta) à chaque fois que l'on est pas à 119 interruptions à la fin d'un tour. Si on a moins de 119 interruptions, on tourne trop lentement, on ajoute 10 à delta, si on en a plus on tourne trop vite et on retranche 10 à delta.
La valeur de delta est retranchée de n3 pour s'ajuster exactement à 3°.
Le choix d'un pas de 10 pour delta est empirique. Un pas plus fin conduira à une stabilisation plus lente mais plus précise, un pas élevé peut rendre l'affichage instable.
Dans la version actuelle du programme, on constate que delta se stabilise à environ 420. C'est à dire que l'on obtient la valeur exacte au bout d'au moins 42 tours, c'est à dire au bout de 5 à 6 secondes à 500tr/min. On pourrait aussi calculer/mesurer la correction et la coder "en dur" dans le programme en retranchant une valeur constante à la valeur théorique de n3 calculée par 32xn0/120. Mais avec cette méthode, il faudrait refaire les calculs/mesures à la moindre modification du programme.