Le générateur d'ondes Schumann du crocodile
Version à microcontrôleur

Version : V1.0
Date : 4 sept 2014
Auteur : croco31

Résumé :
Ce document décrit la construction d'un générateur d'ondes magnétiques dites de « Schumann » de fréquence 7.83Hz, afin de tester les effets allégués souvent mentionnés sur l'écoute audiophile et le bien-être.

Avertissement :

Les informations données ici sont destinées à la réalisation expérimentale d'un montage électronique. L'auteur met en garde contre le risque électrique de la manipulation d'un circuit relié au secteur 240V et décline toute responsabilité suite à son usage.

1. Introduction aux ondes Schumann

Les ondes dites de Schumann sont en fait des oscillations de résonance de très très faible amplitude du champ magnétique terrestre, dues à la cavité Terre-Ionosphère (couche de l'atmosphère à 55km d'altitude environ) et entretenues par les décharges d'éclairs des orages qui se produisent en permanence sur la Terre.
Elles ont été prédites par le physicien Otto Schumann en 1952 et mesurées effectivement vers 1960.
La mesure en elle-même est assez difficile vu le faible niveau de ces harmonique (1picoT alors que le champ magnétique moyen vaut 47microT), et les perturbations électriques de notre environnement.

Contrairement à ce que est dit sur divers sites web, la fréquence de la résonance de Schumann n'a pas varié en fréquence (ce qui est logique car la cavité Terre/Ionosphère n'a géométriquement pas changé). Néanmoins l'activité du Soleil et l'alternance jour/nuit font varier de quelque 0.5Hz ces fréquences de résonance.

Les fréquences moyennes des résonances : 7.83, 14.3, 20.8, 27.3 et 33.8 Hz.

Le niveau et la fréquence de ces résonances est du même ordre que les champs magnétiques produits par le fonctionnement du cerveau humain, ce qui pourrait expliquer l'effet présumé de ces résonances de Schumann sur l'écoute audiophile et l'état de bien-être.

On conçoit aisément que vu leur faible niveau (de l'ordre du picoTesla soit 10^{-12} T), ces résonances puissent être masquées dans notre environnement électromagnétique de plus en plus perturbé de nos villes.

La mise en place d'un appareil qui récréé les fréquences manquantes pourrait avoir un effet biologique: c'est le but expérimental du générateur d'onde Schumann décrit ici.

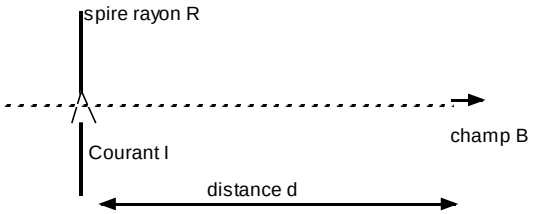
2. Choix de l'intensité du champ magnétique produit

L'objectif du montage est de produire un champ magnétique sinusoïdal à la fréquence de 7.83Hz, qui se superpose au champ magnétique ambiant et remplace la résonance Schumann déficiente.

Il faut donc produire un champ magnétique de l'ordre de 1pT dans un volume de type appartement ou maison, soit environ une distance de 10mètres autour de l'appareil.

Cela se fait avec une bobine parcourue par un courant alternatif.

Dans le cas d'une spire simple de rayon R dans l'air, le champ produit sur l'axe à une distance d est donné par la formule exacte suivante :



$$B = (\mu_0.I/2.R) . (1+d^2/R^2)^{-3/2}$$

La formule approchée suivante est suffisante quand l'angle R/d est faible (développement limité autour de 0) :

$$B \sim \mu_0.I .R^2/(2.d^3)$$

$\mu 0 = 4.\pi .10e\text{-}7$

I= courant en Ampères

R= rayon de la spire en mètres

d= distance sur l'axe au centre de la spire en mètres

B= champ magnétique en Teslas

En prenant un courant de 10mA facile à sortir de composants simples, et une spire de rayon R=2cm on obtient déjà un champ de 0,5pT à 2 mètres de la spire, et qui décroît comme le cube de la distance :

Distance	Champ B
2m	0,5pT
4m	0,05pT
10m	0,005pT

On voit donc que si on bobine 200 à 300 tours de fil sur une bobine de 2cm de rayon, on obtient largement 1pT à 10mètres, et beaucoup plus à quelques mètres (100pT à 2m).

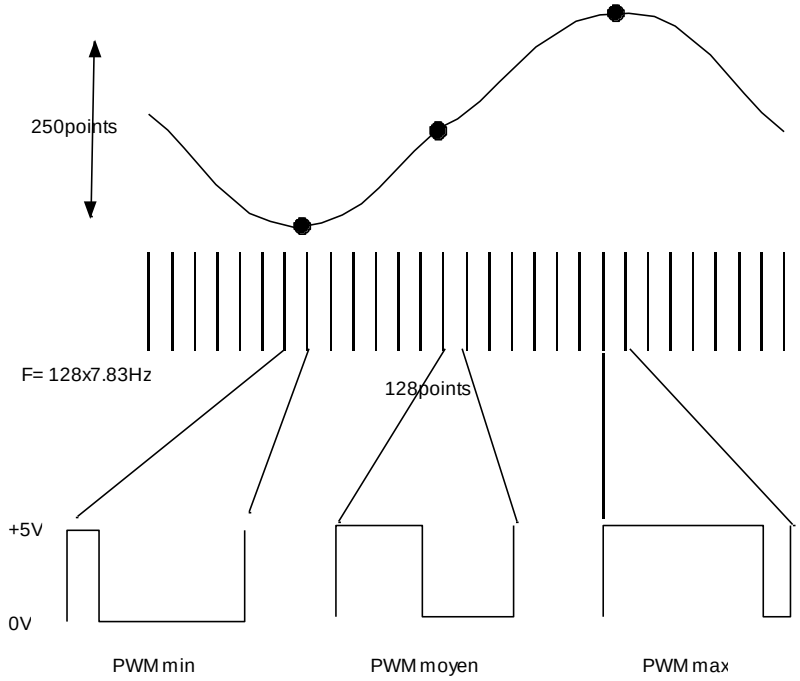
L'intérêt de viser 10mA est que ce faible courant est facile à produire sans distorsion par un ampli opérationnel courant type LM358.

3. La production du champ magnétique

Le champ magnétique est produit par une bobine à air de diamètre 4cm environ, parcourue par un courant sinusoïdal de 10mA crête environ, le nombre de tours de la bobine est de l'ordre de 200 à 300 tours.

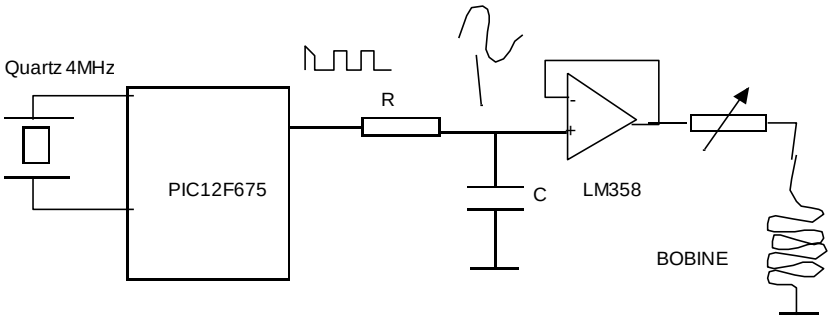
Le courant I sinusoïdal est produit par un microcontrôleur PIC12F675 par un montage PWM (modulation de largeur d'impulsion) dont la fréquence de référence est issue de l'oscillateur à quartz du PIC.
On aurait pu utiliser l'oscillateur interne du PIC pour économiser le quartz, mais il y aurait des dispersions entre les divers composants et le montage serait moins facilement reproductible au niveau de la fréquence 7.83Hz.

.Le PWM fonctionne sur 128 points par période du sinus à 7.83Hz grâce à une table de valeurs dans le programme. Chaque valeur va de 0 à 250, ce qui permet de coder finement la sinusoïde. Noter que 64 points auraient été suffisants en pratique tout en produisant une faible distorsion du sinus. Les 128 points produisent une deuxième harmonique à -30dB ce qui est largement suffisant comme faible distorsion. On pourrait utiliser ce type de table pour produire une autre forme d'onde en sortie.

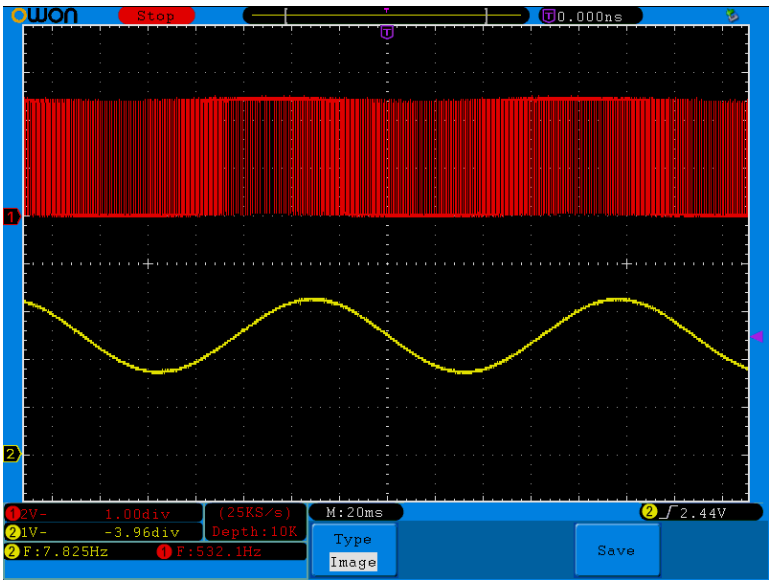


Génération du sinus en PWM

La sortie du PIC est tout ou rien soit 0V soit 5V (quasiment l'alimentation du PIC qui peut fonctionner sans problème de 3 à 5V), et le sinus est reconstitué par un filtrage analogique à travers un réseau RC (R=147K et C=1µF) ce qui donne une tension sinusoïdale de fréquence 7.83Hz aux bornes du condensateur de filtrage.



Filtrage RC



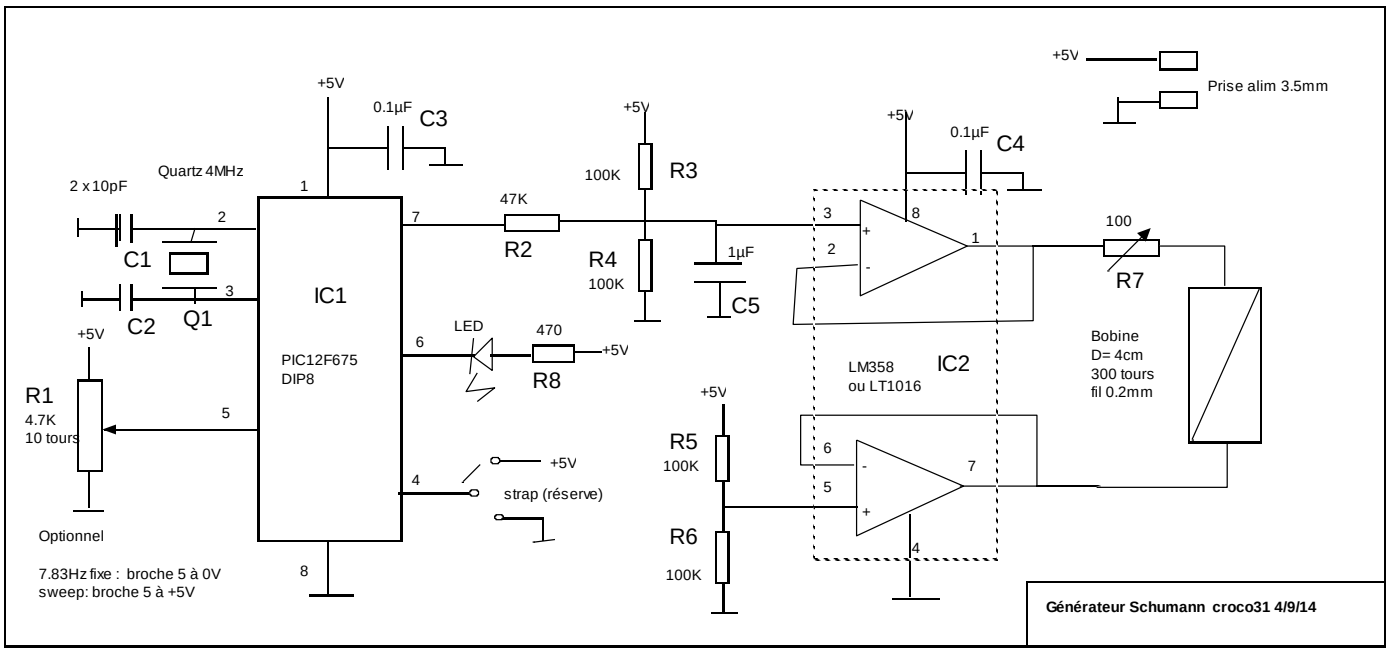
Rouge : sortie du PIC Jaune : sinus filtré en sortie

Comme le réseau RC ne peut pas alimenter directement une bobine (car cela l'amortirait complètement), un suiveur de tension à base d'AOP double LM358 permet de fournir le courant nécessaire de 10mA, sans charger le réseau RC grâce à la grande impédance de l'AOP.
Comme l'AOP est double, l'autre côté de la bobine est connectée sur un point milieu artificiel créé par le deuxième AOP disponible dans le LM358. Ce n'est pas obligatoire mais cela évite une composante continue du champ magnétique produit.
Enfin une résistance série avec la bobine, éventuellement ajustable, permet de limiter le courant dans la bobine, et éviter de saturer l'AOP qui ne peut fournir que 20mA entre 0.5V et 4V environ.
C'est pourquoi l'amplitude de la tension sinus produite est volontairement limitée à 1V crête et centrée sur le milieu de la tension d'alimentation, afin de ne pas saturer l'AOP. Ceci suffit à produire le courant de 10mA voire plus en jouant sur la résistance série de la bobine. La résistance totale de celle-ci peut aller jusqu'à 100ohms avec 1V, si le fil utilisé est très fin. La résistance série permet aussi de vérifier au scope la forme et l'amplitude du courant sinus traversant la bobine.

En soi le schéma précédent suffirait pour tester les effets du générateur, mais comme il y avait des broches disponibles parmi les 8 broches du PIC12F675, des fonctions ont été rajoutées :

- Un potentiomètre d'ajustage de la fréquence autour de la valeur 7.83Hz : la position du potentiomètre permet de choisir 3 modes de fonctionnement :
  - Fréquence fixe 7.83Hz quand le potentiomètre est à 0
  - Balayage automatique de la période du sinus autour de 7.83Hz par pas de 128µs de -0.04Hz à +0.04Hz en 10 minutes environ quand le potentiomètre est au maximum
  - Fréquence ajustable entre -0.2 Hz et +0.2Hz autour de la valeur 7.83Hz dans les autres cas.
  - On peut ne pas monter ce potentiomètre en connectant l'entrée du PIC à 0 ou +5V si on veut une fréquence fixe ou un balayage respectivement.
- Une LED indiquant le fonctionnement du PIC et les modes choisis
- Une entrée libre du PIC est en réserve pour choisir un autre mode futur (autre fréquence par exemple) en changeant seulement le PIC.

4. Le schéma électrique

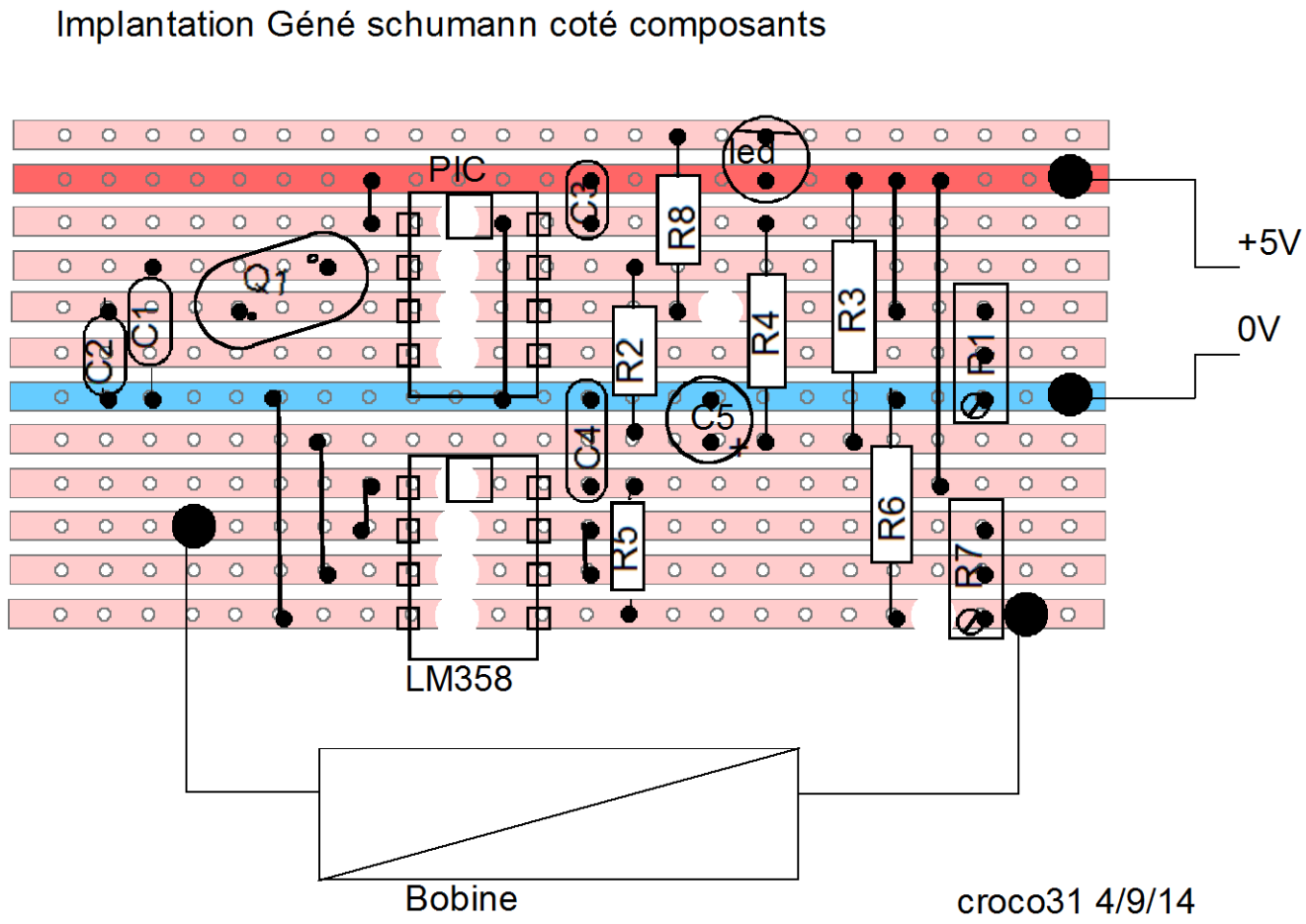


5. Alimentation du circuit

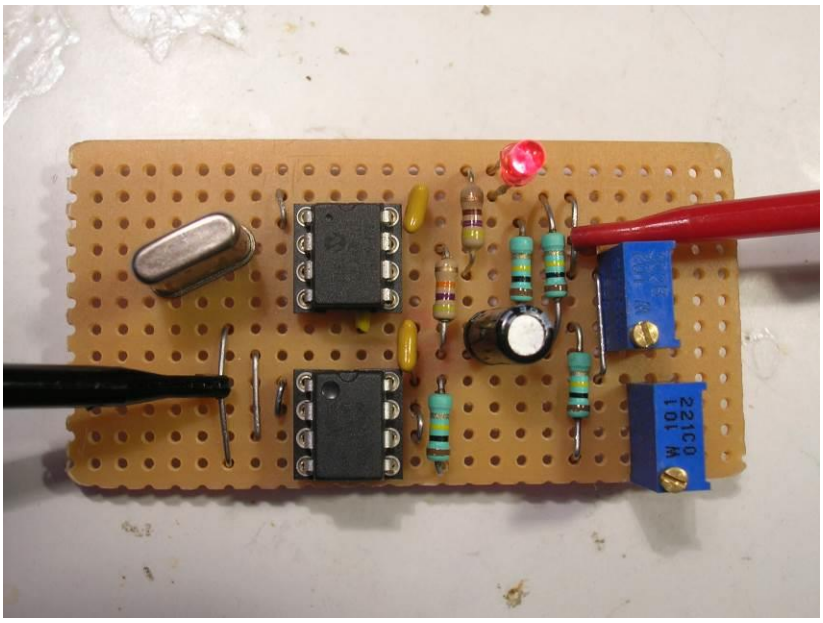
L'alimentation du circuit est en 5V, ce qui permet de l'alimenter sur une prise USB de PC (avec le câble adéquat) ou directement avec une petite alimentation à découpage 220V/5V que l'on trouve sur ebay pour 2 ou 3 euros.
Le circuit demande seulement 20mA environ, et la tension peut descendre à 4V sans problème, par contre ne pas dépasser 5.5V sous peine de destruction du PIC.
Si on veut alimenter en 12V il faudra rajouter un régulateur 78L05 pour produire du 5V pour le PIC et l'AOP.

6. Réalisation et mise au point

La réalisation du prototype se satisfait d'une petite plaque pastillée 2.54 à trous ou à bandes Veroboard sur laquelle sont montés les 2 composants 8 broches sur support (PIC12F675 et LM358) pour pouvoir les remplacer facilement, les autres composants sont soudés et reliés par fils coté cuivre.



Carte verobaord 10 coupures de bandes, 8 straps



Circuit proto 65x35mm

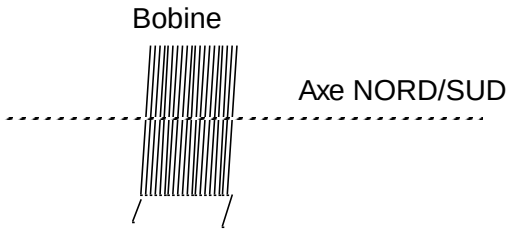
Pour la mise au point, monter d'abord le PIC, le quartz et la LED et sa résistance : dès la mise sous tension la LED clignote plusieurs fois pour indiquer que le logiciel s'est initialisé puis pulse à la fréquence produite en sortie (7.83Hz) avec une intensité dépendant de la position du potentiomètre de réglage s'il est présent:

- Si mode FIXE : la LED est lumineuse
- Si mode AJUSTABLE par potentiomètre : LED de faible intensité
- Si mode SWEEP : la LED a une intensité moyenne et fait un flash à chaque changement de fréquence à chaque minute environ.

Monter ensuite le LM358 et ses composants associés et la bobine : on peut vérifier au scope la forme du courant sinus produit aux bornes de R7, mais on peut aussi écouter le son à l'oreille avec un casque pour vérifier qu'il n'est pas distordu. Le broche 7 de l'AOP est polarisée à la moitié de la tension d'alimentation soit 2.5V continu , vérifiable avec un multimètre.

7. Utilisation

Le montage (au moins la bobine) doit être monté dans un boîtier non magnétique (plastique ou bois) et l'axe de la bobine doit être orienté sur la ligne Nord/Sud :



D'après ce que j'ai trouvé sur le Web , il est préférable de mettre la bobine en hauteur (en haut d'une armoire non métallique), à l'étage par exemple pour couvrir un volume de 10 mètres environ.
A confirmer par expérimentation.

8. Logiciel

Le logiciel du PIC12F675 est développé en C (compilateur MPLAB-XC8 dont une version Lite gratuite peut être téléchargée sur www.microchip.com)
Le PIC 12F675 est configuré pour utiliser un quartz externe 4MHz, et utiliser toutes ses broches en broches IOs.

Il se programme avec un programmateur de PIC à partir du fichier .HEX qui contient aussi les bits de configuration du PIC.

En voici une copie (c'est du texte) :

```
:020000003A289C
:10000800A00003088301B0000A08B1008A01640057
:1000180083120C1C2A28A114051605120C108B1229
:10002800900125088F0024088E0001309000220AD4
:100038007F39A2000B1101308A0022080921143EE1
:1000480081008B160508FC39013885000B1D322804
:100058008B120B11FC3085050C180C288312310803
:100068008A0030088300A00E200E09002130840089
:10007800303046208316FF23900083014B28040666
:1000880080018 40A0406031D4328640000346400C8
:10009800831605100515051285148312851 0D130B5
:1000A800831681008312AA01831685100A308312F1
:1000B8008A2183168514643083128A21AA0A0330A0
:1000C8002A02031C58281430A3003530A400FC3041
:1000D800A5002408AB002508AC0083168E148312F3
:1000E800A110900125088F0024088E00013090008F
:1000F8008B128101A20183160C140B178B17831224
:10010800A801A9010430AA00A7012110FE30A90006
:100118006400A11C8D28A110143083169F0009309B
:1001280083129F009F149F1897281E08A600A608F0
:10013800031DAA283530A400FC30A50022083F3949
:10014800031D83288316023085068328FB30260288
:10015800A208831603198510831203 1CED282208B0
:10016800033A03192118BA28831685148312A208A2
:10017800031D8C282110A80A0830280203 1C8C288B
:10018800A801A90A0A302902031C8C28A901831690
:10019800851083122114A708031DDA28AA0A0A3039
:1001A8002A02031CDD28A701A70ADD28AA0BDD28DF
```

```
:1001B800A701A110A11CDE28A1103030DE00FC3000
:1001C800DF002A085E07A4005F0803185F0AA5007D
:1001D8008C28220BF128831685141530DE00FC309C
:1001E800DF00831226088312AD000310AD0C031044
:1001F8002D0C3F395E07AB005F0803185F0AAC009F
:100208002B08A4002C08A5008428820773347934AD
:100218007E34843489348F3494349A349F34A434AB
:10022800A934AE34B334B834BC34C034C434C8345C
:10023800CC34CF34D334D634D834DB34DD34DF3463
:10024800E134E334E434E534E534E634E634E634E2
:10025800E534E534E434E334E134DF34DD34DB34ED
:10026800D834D634D334CF34CC34C834C434C0347E
:10027800BC34B834B334AE34A934A4349F349A347B
:1002880094348F34893484347E34793473346D34BF
:10029800683462345D34573452344C344734423411
:1002A8003D34383433342E342A34263422341E3440
:1002B8001A341734133410340E340B340934073419
:1002C800053403340234013401340034003400347A
:1002D80001340134023403340534073409340B344F
:1002E8000E341034133417341A341E34223426349E
:1002F8002A342E34333438343D34423447344C3481
:10030800523457345D34623468346D348312AE002D
:10031800FA30AF0090296400AF032F0F8F29AE0B7E
:040328008C29080014
:02400E00893FE8
:10420000FF00FF00FF00FF00FF00FF00FF00FF00B6
:00000001FF
```

Le logiciel est simple :

- Le main initialise les registres de configuration et le timer1 pour l'interruption de fréquence 128x7.83Hz
- Puis boucle de lecture synchronisée sur l'interruption pour conversion de la tension sur la broche 5 et exécution du mode choisi
- L'interruption Timer1 qui monte pPWM=1 s'entretient en rechargeant la valeur du compteur Timer1 : on doit compenser la période pour tenir compte du temps d'exécution de la routine d'IT qui se cumule à la période voulue (ajoute des cycles).
- Le Timer0 est utilisé pour remettre le signal pPWM à 0 au bout de temps pris dans la table des valeurs du sinus en µS.
- le Watchdog interne est actif et provoquera un reset du PIC en cas de problème.

Attention : dans le cas où une recompilation est effectuée, il se pourrait que le compilateur utilisé optimise différemment la routine d'interruption, ce qui va modifier la fréquence de l'interruption du Timer1 : dans ce cas il faudra ajuster la constante qui fixe la période des ITs Timer1.

Le source C :

```
/* ------------------------------------------------------------------*/
/* Prog de generation d'une sinusoide onde Schumannn a 7.83Hz        */
/* Pour PIC12F675     Compilateur HiTech C  PICC V8.02               */
/* Auteur: croco31                                                   */
/* ------------------------------------------------------------------*/
/* Modifs:
   27/08/2014  JLC creation
   03/09/14    JLC passage en mode oscillateur a quartz
 */


#include <htc.h>


/*
Configuration bits of PIC12F675
 - external quartz oscillator 4MHz
 - GP4 and GP5 are used as CLKIN/CLKOUT
 - WDG active
 - Power Up Timer active
 - MCLR used as IO
 - brown out disable pour focntionner de 3 a 5V
*/

// il faut HS si quartz de plus de 4MHz
__CONFIG(XT & UNPROTECT & WDTEN & PWRTEN & MCLRDIS & BORDIS);

/* clear EEPROM si usage comme parametre */
__EEPROM_DATA(0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF);


/*
 Pin usage of PIC12F675:
 -----------------------
  VSS  (8): GND
  GP0  (7): out PWM+ sortie PWM vers la bobine
  GP1  (6): out controle de la LED
  GP2  (5): Analog in AN2  entree du trimmer de reglage de frequence et selection de mode
  GP3  (4): INPUT ONLY: spare
  GP4  (3): IO CLKOUT  quartz 4MHz
  GP5  (2): IO/CLKIN quartz 4MHz
  VDD  (1): power supply +5V

*/

/* definition usage des ports       */
/* -------------------------------- */
#define pTRIMIN  (GPIO2)
#define tTRIMIN  (TRIS2)
#define pPWM   (GPIO0)
#define tPWM   (TRIS0)
#define VAL_PWMON 0x01
#define VAL_PWMOFF 0x00
#define MASK_PWM 0xFC  // pLED doit toujours etre a 0 pour allumer la LED
#define pREFOUT (GPIO4) // non utilise si quartz externe
#define tREFOUT (TRIS4)
#define pLED (GPIO1)
#define tLED (TRIS1)
/* GP3 input is used for mode selection  0= normal 1= calibration */
#define pMODE   (GPIO3)
#define tMODE  (TRIS3)



/* Table LUT pour la generation du sinus par PWM */
/* tiree de http://www.daycounter.com/Calculators/Sine-Generator-Calculator2.phtml */
/* nombre de points  = 128 chaque echantillon dure 498.8uS */
/* valeur max = 230 chaque valeur vaut 4uS sur Timer0   */

/* table sinus de 128 points MAX=230 */
const unsigned char LUT_PWM[128]= {
115,121,126,132,137,143,148,154,
159,164,169,174,179,184,188,192,
196,200,204,207,211,214,216,219,
221,223,225,227,228,229,229,230,
230,230,229,229,228,227,225,223,
221,219,216,214,211,207,204,200,
196,192,188,184,179,174,169,164,
159,154,148,143,137,132,126,121,
115,109,104,98,93,87,82,76,
71,66,61,56,51,46,42,38,
34,30,26,23,19,16,14,11,
9,7,5,3,2,1,1,0,
0,0,1,1,2,3,5,7,
9,11,14,16,19,23,26,30,
34,38,42,46,51,56,61,66,
71,76,82,87,93,98,104,109
};
#if 0
/* table sinus de 128 points MAX=255 */
const unsigned char LUT_PWM[128]= {
```

```
128,134,140,146,152,158,165,170,
176,182,188,193,198,203,208,213,
218,222,226,230,234,237,240,243,
245,248,250,251,253,254,254,255,
255,255,254,254,253,251,250,248,
245,243,240,237,234,230,226,222,
218,213,208,203,198,193,188,182,
176,170,165,158,152,146,140,134,
128,121,115,109,103,97,90,85,
79,73,67,62,57,52,47,42,
37,33,29,25,21,18,15,12,
10,7,5,4,2,1,1,0,
0,0,1,1,2,4,5,7,
10,12,15,18,21,25,29,33,
37,42,47,52,57,62,67,73,
79,85,90,97,103,109,115,121
};
#endif


/* calage TIMER1 16bits pour avoir une periode de 998uS soit (65535-998)  */
/* TIMER1 reboucle de 0xFFFF a 0 */
#define PERIODE_THEORIQUE (65535-998)
#define PERIODE_COMPENSEE (PERIODE_THEORIQUE+28) // compensation de la duree de la routine interruption +:augmente Freq
#define OFFSET_THEORIQUE (20)

unsigned char index_cycle_pwm=0; // index du cycle de 0 a 255
unsigned int periode_cycle; // valeur courante de la periode TIMER1 pour reglage de frequence
unsigned char periode_offset; // decalage offset dans la periode
bit flag_synchro,flag_sweep; // synchro du main sur l'interruption

/* ------------------------------ */
/* Interruption Timer1 et Timer0 */
/* ------------------------------ */
/* ATTENTION: toute modification du code interruption change la periode du TIMER1 */
#pragma interrupt_level 1
void interrupt Interruption(void)
{
 asm("clrwdt"); // clear du watchdog

reboucle:
 // -- IT Timer1: toutes les 1ms
 if( TMR1IF) {
  flag_synchro= 1;
  pREFOUT= 1; // pulse TEST a la frequence du timer1 pour reglage
  pREFOUT= 0;
  TMR1IF= 0;
  T0IE=0; // masque l'IT timer0 si trop proche

  // -- relance de l'IT Timer1 en rechargeant TMR1H et TMR1L pour 998 uS
  // 128 cycles x 997uS  donnent 127.616 ms soit 7.836 Hz
  // 128 cycles x 998uS  donnent 127.744 ms soit 7.828 Hz
  // 128 cycles x 999uS  donnent 127.872 ms soit 7.820 Hz
  //
  // timer1 reboucle a 0xFFFF et avance a chaque cycle 1uS
  T1CON= 0;
  // ******   ICI ajustage du nombre de cycles de la periode du Timer1 ******************
  // periode_cycle est chargee par la tache de fond main()
  TMR1H= (periode_cycle>>8) & 0xFF;
  TMR1L= periode_cycle & 0xFF;

  T1CON= 0x01; // active timer1

  // -- activation du Timer0 pour le cycle PWM courant (0 a 127)
  index_cycle_pwm = (index_cycle_pwm +1) & 0x7F ; // reboucle tout seul de 127 a 0
  T0IF= 0; // clear en cas de debordement
  TMR0= OFFSET_THEORIQUE + LUT_PWM[index_cycle_pwm]; //  ici LUT varie de 0 a 230 centrage autour du milieu de periode
  // Timer0 montera T0IF au bout de (255-TMR0) x 4uS
  T0IE= 1; // enable IT Timer0
  //pPWM= 1;
  // force toujours pLED=0
  GPIO = (GPIO & MASK_PWM) | VAL_PWMON;
 }

 // -- IT Timer0: au bout du cycle PWM courant
 // elle peut arriver dans l'IT timer1 pour les faibles cycles PWM
 // prevoir une marge en fond si l'IT timer1 arrive juste
 if( T0IF) {
  T0IE= 0; // masque IT Timer0
  T0IF= 0;
  // -- retombe la sortie PWM
  // pPWM= 0;
  GPIO = (GPIO & MASK_PWM) | VAL_PWMOFF;
  if( TMR1IF) goto reboucle; // evite un jitter de phase sur l'horloge Timer1
 }

} /* return d'IT */

void
Delayms (unsigned char nbmilli)
{
unsigned char count;
         do {
          count=250;
```

```
         while(count--) {asm("clrwdt");}
        } while (--nbmilli);
}


/*****************************************************************************************/
/* Main program                                                                          */
/*****************************************************************************************/

main (void)
{
unsigned char val_potar;
unsigned int new_periode;
unsigned char count_sweep;
unsigned char count_sec;
unsigned char val_sweep,dir_sweep;



 asm("clrwdt"); /* clearing internal watchdog (periode defaut = 18ms) */

/* PORTs initialisation */
 /* -------------------- */
 tPWM   =      0; /* output */
 tTRIMIN=      1; /* input pour entree analogique sur GP2 = AN2)
 tMODE  =      1; /* input MODE select */
 tREFOUT=      0; /* output test */
 tLED=       1; /* output pilotage LED en output a 0 */
 pLED=       0; /* prepare pour allumage de la LED */

 /* configure OPTION REG               */
 /* ---------------------------------- */
 /* GPPU=    1  GPIO Pullups disable
    INTEDG=  1  edge
    T0CS=    0  TIMER0 source = CLKOUT (FOSC/4) = 1uS
    T0SE=    1
    PSA=     0  PRESCALER assigned to TIMER0
    PS2:PS0= 001  prescaler 1:4 soit 4uS par pas de Timer0
  */
 OPTION= 0xD1;

/* clignote la LED 3 fois au demarrage */
for(val_sweep=0;val_sweep<3;val_sweep++) {
 tLED=0;
 Delayms(10);
 tLED=1;
 Delayms(100);
}


/* valeurs par defaut */
/* ------------------ */
periode_offset= OFFSET_THEORIQUE; // pour le timer0, depend de l'amplitude de la table du sinus
periode_cycle= PERIODE_COMPENSEE; // recalage de la periode suivant le code de la routine interruption
new_periode= periode_cycle; // sauve la valeur pour le mode sweep

 /*  POWER ON    */
 POR=1; /* mark power up is done */
 flag_synchro= 0;



 /* TIMER1 (16bits): configuration pour une periode de 498.8uS */
 /* ------------------------------------------------*/
 /* 128 cycles de TIMER1 donnent la periode du sinus */
 /* le sinus est genere par une table de 128 valeurs de PWM */
 /* Timer1 fait une IT quand reboucle de 0xFFFF a 0x0000  a chaque cycle (prescaler=1:1 ) */
 /* on doit tenir compte de la duree de rechargement du timer1 dans la routine d'IT */

 /* TMR1GE= 0 pas de timer1 gate externe
    T1CKPS1:0 = 00 1:1 prescale 1:1 =  pas de 1uS
    T1OSCEN=0  LP oscillator disable
    T1SYNC=0
    TMR1CS= 0 internal clock
    TMR1ON = 1
 */

 T1CON= 0;
 TMR1H= (periode_cycle>>8) & 0xFF;
 TMR1L= periode_cycle & 0xFF;
 T1CON= 0x01; // active timer1

 /* TIMER0 (8bits): sera active a chaque IT TIMER1 en chargeant la valeur depuis  la table LUT_PWM */
 /* ------------------------------------------------------------------------------*/
 T0IE= 0; // pas d'IT timer0 ici
 TMR0= 0;

 /* active les ITs Timer1           */
 /* ------------------------------- */
 index_cycle_pwm = 0;
 TMR1IE= 1;
 PEIE= 1;
 ei();
```

```
/* boucle en tache de fond  */
/* ------------------------ */
/* - prepare la valeur de periode PWM pour le cycle suivant */
/* 3 modes:
    - reglage fixe par valeur constante (depend de la precision de l'oscillateur interne)
    - reglage par potentiometre
    - balayage autour de la valeur fixe
 */

count_sweep=0;
count_sec=0;
val_sweep= 4; // correspondra a 0 uS apres +1
dir_sweep= 0; // sens du balayage
flag_sweep= 0;
count_sec= 254; // pour changer immediatement en mode sweep (find e course du potar)

while(1) {
 asm("clrwdt");
 // -- synchro sur l'IT pour le sweep de frequence et comptage des periodes
 // cela evite des perturbations entre la commande de la lED et la sortie PWM
 while(flag_synchro==0) continue; // attente de chaque IT de periode PWM (l'IT est passee juste avant)
 flag_synchro=0;
 /* NOTA: ne pas manipuler les PORTS ici car utilises sous IT */
 /* capture du potentiometre pour choix du mode */
 /* 0= mode fixe 255=mode sweep autre= reglage */
 /* test du mode calibration */
  /* activation de l'ADC dans ce mode pour lire le potar de reglage */
  /* Tosc/8= 2uS AN2 Vref=Vdd */
  ANSEL=0x14; /* ADCS=001 AN2=analog */
  ADCON0= 0x09; /* ADON et GO=0 */
  /* attente de la valeur convertie : ne doit pas depasser l'echeance de la periode PWM suivante */
  /* la conversion dure 22uS ici 11xTad */
  GODONE= 1;
  while(GODONE) continue;
  /* recupere la valeur 8 bits seulement = MSB des 10 bits produits par ADC */
  val_potar= ADRESH;

  if( val_potar==0) {
    /* ----------------    MODE FREQ FIXE (calibree 7.83Hz) -----------------------*/
    /* clignotement de la LED a 7.83Hz theorique */
    periode_cycle = PERIODE_COMPENSEE;
    if( (index_cycle_pwm & 0x3F ) ==0) tLED= !tLED; // toggle du tri-state , pLED=0
    /* init sur frequence centrale si on repasse en mode sweep */
    count_sweep=0; /* init pour mode sweep */
    count_sec=0;
    val_sweep= 4; // correspond a 0 uS
    dir_sweep= 0; // sens du balayage
    flag_sweep= 0;
    count_sec= 254; // pour changer immediatement en mode sweep (find e course du potar)
  } else if ( val_potar>250) {
    /* -------- MODE SWEEP DE FREQUENCE  ---------------------------*/
    /* balayage entre 7.7 et 7.9 Hz en 10 minutes environ */
    /* index_cycle_pwm va de 0 a 255 pour chaque periode du 7.83Hz */
    /* modif de la periode de 1uS chaque minute */
    if( (index_cycle_pwm ) == 0 ) tLED= 0; // tLED=0 , pLED=0 LED allumee
    if( ((index_cycle_pwm ) == 3 ) && !flag_sweep) {
      tLED= 1; // LED restera allumee un cycle a chaque changement de freq
    }
    if( index_cycle_pwm==0) { // debut nouvelle periode 7.83Hz
       flag_sweep= 0;
       count_sweep++;
       if(count_sweep>=8) { // top seconde au bout de 8 cycles 7.83Hz environ
        count_sweep=0;
        count_sec++;
        if( count_sec>=60) { /* 1 minute environ */
         count_sec=0;
         /* chaque minute environ */
         tLED= 0; // marque le changement
         flag_sweep= 1; // pour laisser la LED allumee un cycle 7.83Hz
         if( dir_sweep==0) {
          // augmente la periode
          val_sweep++;
          if( val_sweep>=10) dir_sweep=1; // change de sens
         } else {
          // diminue la periode
          val_sweep--;
          if(val_sweep==0) dir_sweep=0; // change de sens
         }
         /* on modifie la periode courante */
         flag_synchro=0;
         while(!flag_synchro) continue;
         flag_synchro=0;
         periode_cycle= PERIODE_COMPENSEE -5 + val_sweep; // -5 a +5 uS par periode PWM
        }
       }
    }
  } else {
    /* ---------------      MODE POTENTIOMETRE ----------------------------------*/
    /* signale ce mode par flashs de la LED  a chaque cycle 7.83Hz             */
    /* la frequence est donnee par un potar de reglage externe : autour de 7.83Hz */
    if( (index_cycle_pwm ) == 0 ) tLED= 0; // tLED=0 , pLED=0 LED allumee
    if( (index_cycle_pwm ) == 1 ) tLED= 1; // LED eteinte
```

```
     /* calcule la nouvelle periode PWM */
     /* plage de reglage de + ou - 31 autour de la valeur par defaut theorique pour 7.83Hz */
     /* val_potar varie de 0 a 255 : au milieu 0x80 = pas de correction */
     /* 0= -32  128= 0 255= +31 */
     new_periode= PERIODE_COMPENSEE -32 + ((val_potar>>2) & 0x3F);
     /* mise a jour periode courante pour effet sur la routine d'ITs */
     periode_cycle= new_periode;
    /* ---------- */
     count_sweep=0;
     count_sec=0;
     val_sweep= 4; // correspond a 0 uS
     dir_sweep= 0; // sens du balayage
     flag_sweep= 0;
     count_sec= 254; // pour changer immediatement en mode sweep (fin de course du potar)
   }
 }
}
```