Compteur horaire

Institut Gériatrique des Solex
Bernique

Abstract

Réalisation d'un petit compteur horaire sans grande précision pour comptabiliser le temps de mise sous tension d'un appareil électrique.

Keywords: ATtiny85, afficheur led 4 digits, TM1637

Table des matières

I. Opportunité et cahier des charges

Dans la cadre de mon installation d'électrolyse *dérouillante*, je souhaitais mesurer le temps de fonctionnement de l'appareil. En effet, une fois les opérations lancées et après plusieurs heures passées à s'occuper ailleurs, on ne se souvient pas toujours de l'horaire du démarrage et donc du temps total de l'électrolyse. Dommage, car cela ne permet pas d'accumuler l'expérience nécessaire pour planifier les futures utilisations de la machine (genre *ai-je le temps de faire l'électrolyse avant Casimir à la télé* ???… grosse angoisse ;-))

Bref, pas besoin d'une précision atomique… l'idée étant d'avoir un ordre de grandeur, cinq minutes d'écart après une dizaine d'heures de fonctionnement sont tout à fait acceptables.
De même, pas besoin de dépasser les 24h de comptage… et si ça doit dépasser, le compteur repars à zéro et on saura tout de même se rappeler, sauf énorme cuite, qu'on est passé par la case dodo entre temps ;-)

II. Composants et dimensionnement

1 Le cerveau du système

Bien connu des bidouilleurs de tous poils, le µ-processeur ATtiny85 de chez Atmel® est d'un accès très aisé. Il est disponible partout en version nue DIP-8.

Il bénéficie d'une immense communauté pour le support, est directement programmable en C et donne ainsi accès à toute sa puissance sans compromis sur l'efficacité dès lors que l'on se donne la peine de maîtriser ses registres intimes.
Il peut aussi être programmé en langage Arduino à l'aide de bibliothèques dédiées et restreintes adaptées à sa petite mémoire… il faut alors accepter de perdre un peu du contrôle strict de son comportement dicté par le contenu figé de ces mêmes bibliothèques !
J'ai choisi ici sa déclinaison ATtiny85-20PU pour ses dimensions restreintes et un nombre d'entrées/sorties largement suffisantes. On pourrait opter pour la version 10PU qui présente l'avantage de fonctionner à tension plus faible (jusqu'à 1,8v), plutôt intéressante dans des applications où les réserves d'énergie ne sont pas infinies!

Le programme à charger en mémoire est donné en annexe (les conditions de transfert sont données en en-tête du programme).

2 Alimentation

Fonction du lieu d'implantation, il faudra composer avec ce qui est disponible.
Dans le cas de mon application, j'ai à disposition une alimentation ATX qui délivre du 5V stabilisé de qualité et en quantité largement suffisante. Je n'ai pas cherché plus loin.
Veillez juste à ne pas dépasser les 5,5V fatidiques pour le µC… ou alors il faut passer par un LM7805 en guise d'alimentation avec les condensateurs de déparasitages qui vont bien (amont et aval).

3 Module afficheur 7 segments 4 chiffres

Pour afficher le temps passé, rien de plus simple qu'un petit module à led disposant de 4 chiffres avec le double point au centre… comme on en trouve pour moins de 2€ chez les vendeurs chinois du web (frais de port inclus… sic!).
Tout prêt à l'emploi avec son chip TM1637 intégré, il fonctionne selon un protocole affiché comme du I2C mais qui n'en est pas exactement!
Il se branche simplement sur deux entrées digitales du µC. J'ai opté pour PB3 et PB4. Il est alimenté avec le 5V déjà récupéré pour le processeur.

Tant que le temps écoulé sera inférieur à une heure, l'afficheur donnera les minutes et les secondes séparées par le double point qui clignote à chaque seconde.
Au delà de l'heure de fonctionnement, on affichera les heures et les minutes avec toujours le double point clignotant.

4 La base de temps

La base de temps sera donnée par l'oscillateur RC à 8MHz du µC (ramenés à 1MHz via un diviseur fixé à 8 par fuse au moment de la compilation et du transfert du programme). Aucune précision n'étant requise, nul besoin d'utiliser une horloge RTC additionnelle (compatible avec l'ATtiny85, mais juste overkill ici).

En résumé, une interruption sur le dépassement du compteur du Timer1 permet de compter les micro-secondes qui s'écoulent, typiquement un µC calée à 1MHz et un registre qui déborde à 125 tics d'horloge.

Pour rappel, le tic d'horloge du Timer est défini par :

$$tic = \frac{2^{taille\ du\ compteur} \times diviseur_{Timer}}{fréquence\ \mu C}$$

Pour notre Timer1 (8 bits sur ATtiny85) avec un diviseur choisi à 8 (p'tain y-a que des 8 sur cette page!), on obtient une base de temps de :

$$tic = \frac{2^8 \times 8}{10^6} = 2048\,ms$$

En fixant un seuil de débordement à 125 tics, on obtient notre seconde fétiche :

$$\frac{2048 \times 125}{256} = 1000\,ms$$

Côté programme, à chaque débordement du Timer1, on incrémente le compteur des secondes… et une simple gestion des maxima permet d'incrémenter les minutes et les heures.

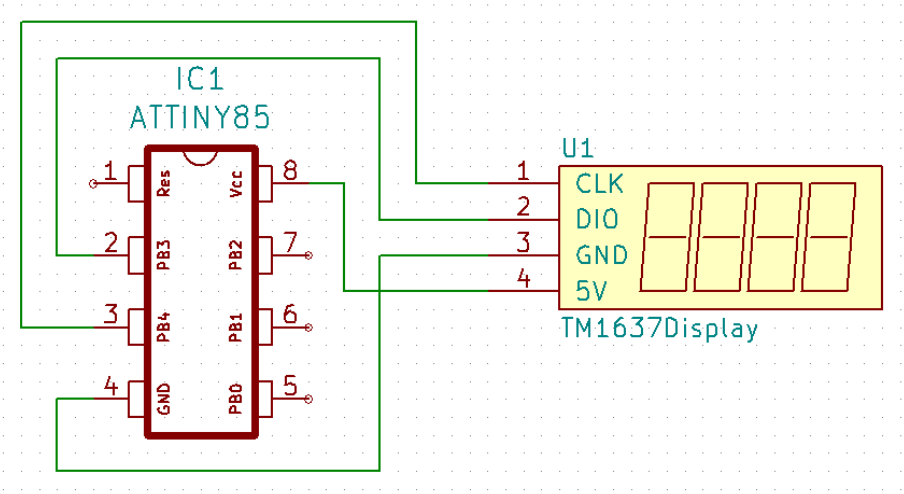
La suite est à lire dans le code "hybride" du programme en annexe et écrit comme suit :

- une armature en langage C pour compter le temps à l'aide du Timer1 et maîtriser au mieux le comportement du µC (j'ai volontairement écarté l'usage de la fonction *delay()* qui fige le programme et n'est pas très élégante!)
- un appel ponctuel à une fonction de la bibliothèque précédemment évoquée pour piloter l'écran lcd afin de contourner la difficulté d'une programmation directe en C (pas envie d'éplucher le fonctionnement détaillé du bazar).

Les puristes hurleront, la caravane est passée ;-)

III. Le circuit

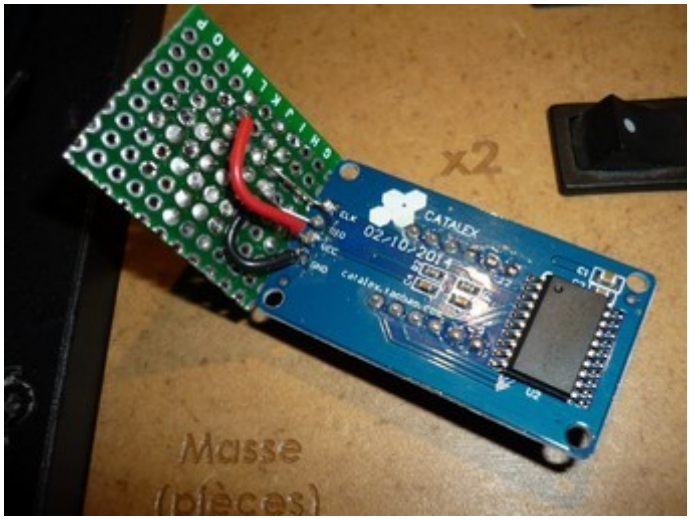
Plus simple, ça va être compliqué… un bon schéma valant tous les commentaires !

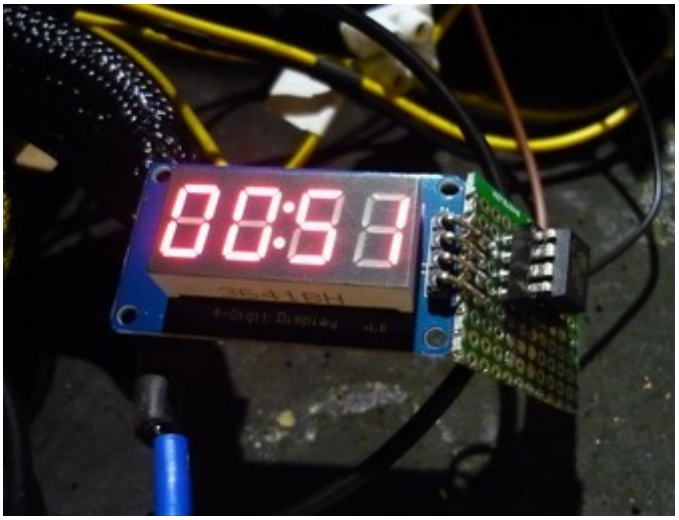


Comme expliqué auparavant, +5VDC et GND sont issus de l'alimentation ATX.

IV. Mise en œuvre

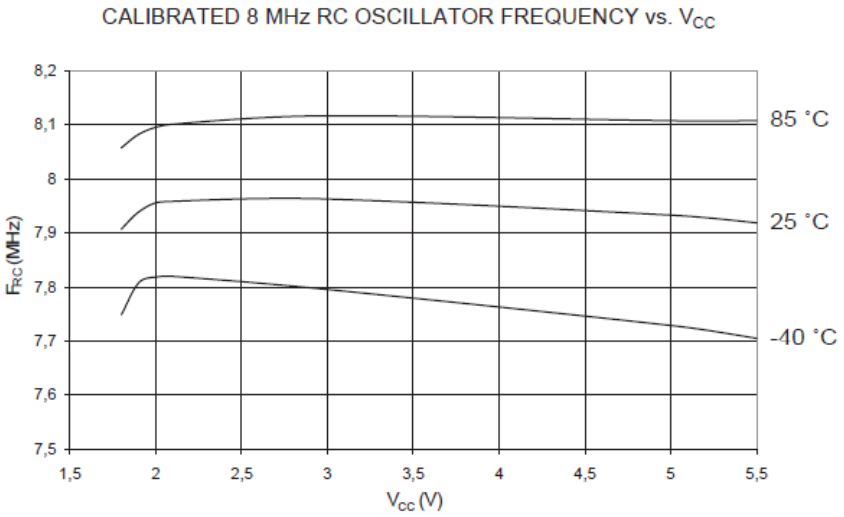
Et hop, le µC sur sa plaque à trous relié à l'écran en mode proto (ou barbare, c'est selon!), et le même durant un test de comptage…





… test nécessaire car la fréquence du µC est donnée par un oscillateur RC et non un quartz qui serait nettement plus précis. La fréquence est influencée par la tension d'alimentation, la température de fonctionnement et le temps de traitement de l'instruction de remise à zéro du compteur de débordement !

Figure 22-40. Calibrated 8 MHz RC Oscillator Frequency vs. V_{CC}



En résumé, chacun devra faire son apprentissage !

Pas compliqué pour autant : lancer un chrono sur un téléphone portable ou un PC, et lancer le montage en parallèle (il démarre à la mise sous tension après deux secondes d'affichage d'un 88:88). Puis comparer le résultat après quelques heures de fonctionnement ! La suite, c'est une règle de trois pour adapter le paramètre TEMPO du programme (1000 millisecondes théoriques) à la vraie valeur.

Dans mon cas, j'ai terminé à 968ms pour une minute de retard sur la référence après 11 heures de fonctionnement (alimenté en 5,12v et sous 22°C)! Good enough ;-)

V. Conclusion

Woulà... un p'tit montage à deux composants pour un résultat sympa.
Difficile de faire plus simple ;-)

Annexe : le code

```
/*******************************************************************************
*  Compteur horaire                                                            *
*  Affiche minutes:secondes si temps passé inférieur à 1 heure                 *
*  Affiche heures:minutes si temps passé supérieur à 1 heure                   *
*  utilisation d'un afficheur 7segment 4 chiffres en pseudo I2C                *
*                                                                              *
*  Processor: ATTiny85 @ 1MHz                                                  *
*  Compiler Arduno ISP v1.8.9                                                  *
*  Programmer: USBasp                                                          *
*                                                                              *
*                 ___                                                          *
*              PB5  *|  +  |*  VCC                                             *
*       CLK ← PB3  *|     |*  PB2                                             *
*       DIO ← PB4  *|     |*  PB                                              *
*              GND  *|____|*  PB0                                             *
*                                                                              *
* Timer1:    millis function                                                   *
* Interrupt on COMPA vector on Timer1                                          *
*                                                                              *
* Lfuse: 0x62 / Hfuse: 0xDF / Efuse: 0xFF @1MHz                                *
*                                                                              *
* Bernique BGY#051                                                             *
* Common Creative BY-NC-SA                                                     *
* 2019/09/24                                                                   *
*                                                                              *
* Status: OK                                                                   *
*******************************************************************************/

#define F_CPU 1000000L                  // 1Mhz (8Mhz, prescaled by fuse = 8)

#include <util/delay.h>
#include <avr/interrupt.h>
#include <TM1637Display.h>

#define CLK PB3
#define DIO PB4

int tempo=968;                          // Adjust value to fit perfect 1 second gap (influence of Vin, T°...)
int TEST_DELAY=2000;
int ss=0;                               // seconds
int mm=0;                               // minutes
int hh=0;                               // hours
int point = 0;                          // controlling led double dot behavior
int timer = 0;                          // millis() function like!
int k = 3;                              // led brightness, from 0 to 15

TM1637Display display(CLK, DIO);        // starting instance
```

```
void setup() {
 // setup
 // I/O configuration
 DDRB |=   (1<<CLK) | (1<<DIO);                    // Pin configuration as output

 // definition of test matrix
 uint8_t data[] = { 0xff, 0xff, 0xff, 0xff };
 uint8_t blank[] = { 0x00, 0x00, 0x00, 0x00 };
 display.setBrightness(k);

 // All segments on @ start time
 display.setSegments(data);
 delay(TEST_DELAY);
 display.setSegments(blank);

 //Timer1 setup
 /** Target is 1ms per Compare match to ensure proper millis incremental value
  *  CPU 1Mhz; Timer prescal 8; OCR1A=125 => 1ms per Compare match
 */
 TCCR1 = 0x04;                                    // set prescaler to CK/8
 TIMSK = 0x40;                                    // enable Timer1 Compare Match A Interrupt
 OCR1A = 0x7D;                                    // setup max value for compare match, i.e. 125

 sei();                                           // make interrupt active

}                                                 // end of setup

void loop(){
 // showing minutes:seconds as long as duration is < 1 hour
 while (mm<6000){
  while (ss<60){
   display.showNumberDecEx(ss+mm, (0x80 >> point), true);
  }
  ss=0;
  mm=mm+100;
 }

 // showing hours:minutes as soon as duration is > 1 hour
 hh=100;
 mm=0;
 ss=0;
 while (hh<2359){
  while (mm<60){
   while (ss<60){
    display.showNumberDecEx(hh+mm, (0x80 >> point), true);
   }
   ss=0;
   mm ++;
  }
  mm=0;
```

```
    hh=hh+100;
   }
   // reset after 24h and restart @zero
   ss=0;
   mm=0;
   hh=0;
  }

  ISR (TIMER1_COMPA_vect){                    // Interrupt on COMPA vector
   TCNT1 = 0x00;
   timer++;
   if (timer>tempo) {                         // 1 second has been reached !
    timer=0;
    ss++;
    point = 1-point;                          // inversion of Point between 0 and 1 or 1 and 0
   }

  }
```