Fabriquer un Power Router

Auteur : Association PTIWATT

avec Philippe de Craene dcphilippe@yahoo.fr
et Dominique Boucherie ptiwatt@mailoo.org

Date : juillet 2018 – avril 2019

Version du document : 1.9

Version du programme : 3.4

Un power router est un dispositif qui absorbe le trop d'énergie produit localement – soit par une éolienne, ou par panneaux photovoltaïques, ou tout autre équipement – afin d'éviter d'injecter cette énergie sur le réseau public.

A la fin du document il sera aussi question de pouvoir commander l'arrêt ou la mise en marche d'un appareil, non pas graduellement comme le fait le dispositif initial mais en « tout ou rien » afin de constituer en quelque sorte un système de délestage.

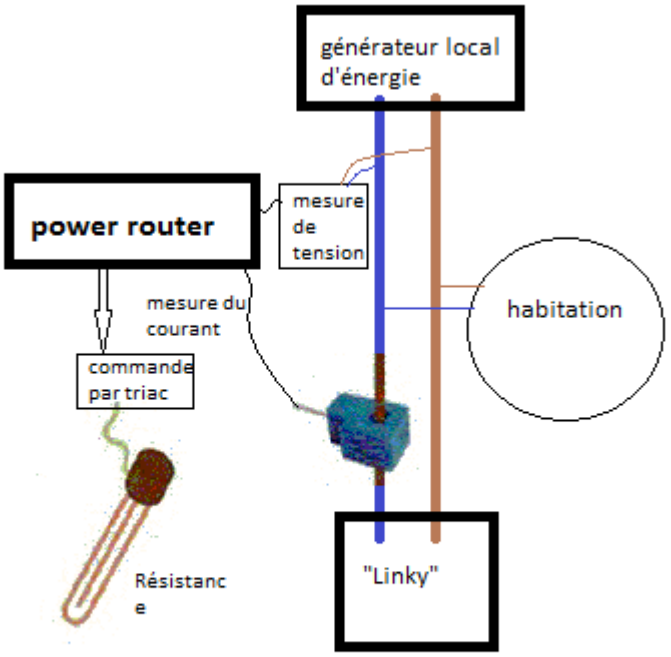
Ce document décrit comment réaliser cet appareil.

A noter : la réalisation de ce programme a demandé plusieurs dizaines d'heures de développement, apprendre à utiliser l'Arduino, comprendre son comportement, en cramer deux… apprendre son langage, faire des tests sur différents capteurs de courant, tester différents algorithmes, et imaginer tous les tests possibles pour fiabiliser au maximum l'appareil.
Toute contribution en vue de l'amélioration de l'appareil est la bienvenue ! Il vous est juste demandé de conserver mon nom et mon email dans l'entête du programme, et bien sûr de partager avec moi cette amélioration. Merci.

Table des matières

Introduction

Il s'agit de mesurer la tension et le courant après le compteur d'arrivée de l'énergie public : en mode de fonctionnement classique, nous avons plus ou moins (rappelez-vous des cours d'électricité au lycée et ce fameux cos φ) la tension en phase avec le courant. Or lorsqu'il y a injection, la tension et le courant seront en inversion de phase.



Il s'agit donc de détecter si le rapport courant/tension sont oui ou non en opposition de phase, auquel cas il s'agira d'alimenter un appareil composé d'une résistance pure telle qu'un chauffe-eau ou un halogène en proportion du courant produit afin de ne plus injecter dans le réseau public.

L'appareil est composé d'un élément de mesure du courant (mini pince ampèremétrique), d'un élément de mesure de tension (simple transformateur bobiné), d'un élément de commande par triac, le tout piloté par un programme dans un Arduino Uno R3.

La partie hardware est composée de 3 parties :

- La carte Arduino avec son câble USB et/ou son alimentation 5V
- La carte d'extension avec les 2 capteurs
- La carte du module triac

La partie logicielle est composée de 3 parties :

- La gestion des données tensions et courants pour détecter l'injection ou la consommation
- La gestion du triac
- La gestion de l'asservissement qui dose la conductivité dans la résistance afin d'empêcher l'injection sur le réseau public

L'appareil proposé ici convient pour quelques centaines de Watts à absorber. L'élément limitateur est le Triac qui dans notre cas accepte 5A maxi.

Liste des courses

1- Un arduino Uno R3 : https://fr.aliexpress.com/item/One-set-New-2016-UNO-R3-ATmega328P-CH340G-MicroUSB-Compatible-for-Arduino-UNO-Rev-3-0/32696412561.html?spm=a2g0s.9042311.0.0.27426c37rrsgNO

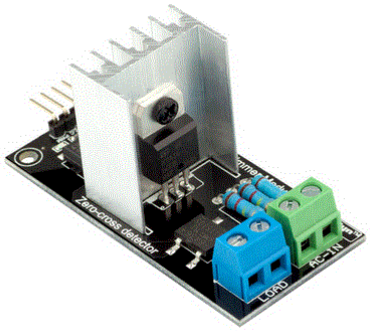
2- Une carte d'extension : https://fr.aliexpress.com/item/Free-Shipping-UNO-Proto-Shield-prototype-expansion-board-with-SYB-170-mini-bread-board-based-For/32502867722.html?spm=a2g0s.9042311.0.0.27426c37rrsgNO

3- Un capteur de courant sensible : https://fr.aliexpress.com/item/Free-shipping-0-30A-sensor-split-core-current-transformer-SCT006/32579590465.html?spm=a2g0s.9042311.0.0.27426c37zd6RJS

Le capteur de courant est une sorte de mini transformateur bobiné qui doit être connecté au secondaire sur une résistance (dite de Burden) de 100Ω. Inutile d'augmenter la valeur de cette résistance pour tenter d'obtenir un meilleur taux de transformation, ça sature très vite ces petits capteurs.
Au niveau de votre installation domestique, si vous êtes moderne durable et bio vous ne consommez pas plus de 4000W en instantané. Ce qui donne un courant I = 17A maxi.
Donc laissez tomber les gros capteurs de 30A qui sont perdus pour mesurer les faibles courants. Plus vous trouverez un capteur au ratio Itraversé/Isecondaire élevé, plus votre mesure sera sensible, et meilleurs sera votre PowerRouter.

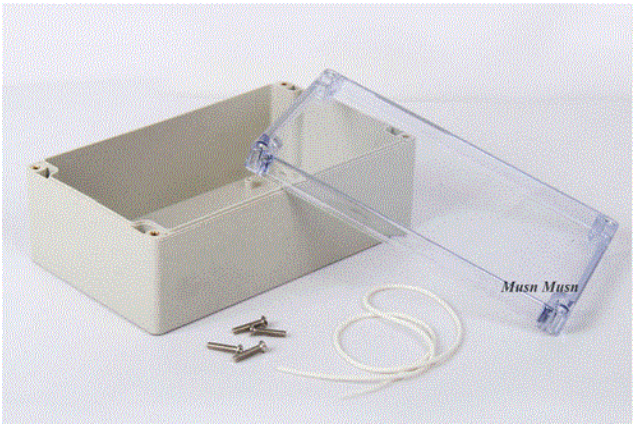
4- Un capteur de tension : un transformateur bobiné (il faut absolument une tension alternative) délivrant de l'ordre de 4V crête à crête au secondaire (généralement ce qui est disponible est bien plus, on calculera alors un pont diviseur de tension (cf plus bas).

5- Un module de commande de Triac : https://fr.aliexpress.com/item/AC-Light-Dimmer-Module-for-PWM-control-1-Channel-3-3V-5V-logic-AC-50-60hz/32802025086.html?spm=a2g0s.9042311.0.0.27426c37xh5Y5t
Ce module est très pratique : il comporte ET le détecteur de passage à zéro nécessaire à l'utilisation du triac, et le dispositif de commande du triac, tout cela par des optocoupleurs qui assurent l'isolation de la haute tension du secteur.
Ce modèle est un peu limité en puissance, 5A maxi, 2A nominal…
Cependant le triac est donné pour 16A. La limitation doit être dû à la faible épaisseur des pistes du circuit imprimé : il suffit dans ce cas de doubler les pistes de puissance par du fil électrique



6- Une alimentation 5V pour l'Arduino, c'est-à-dire un chargeur de téléphone portable de récupération, du câble Dupont (bof bof c'est plein de mauvais contacts, rien de vaut mieux que la soudure), une paire de prises Cinch (mâle + femelle châssis) pour connecter les capteurs, et une toute petite poignée de composants électriques dont la liste est un peu plus bas.

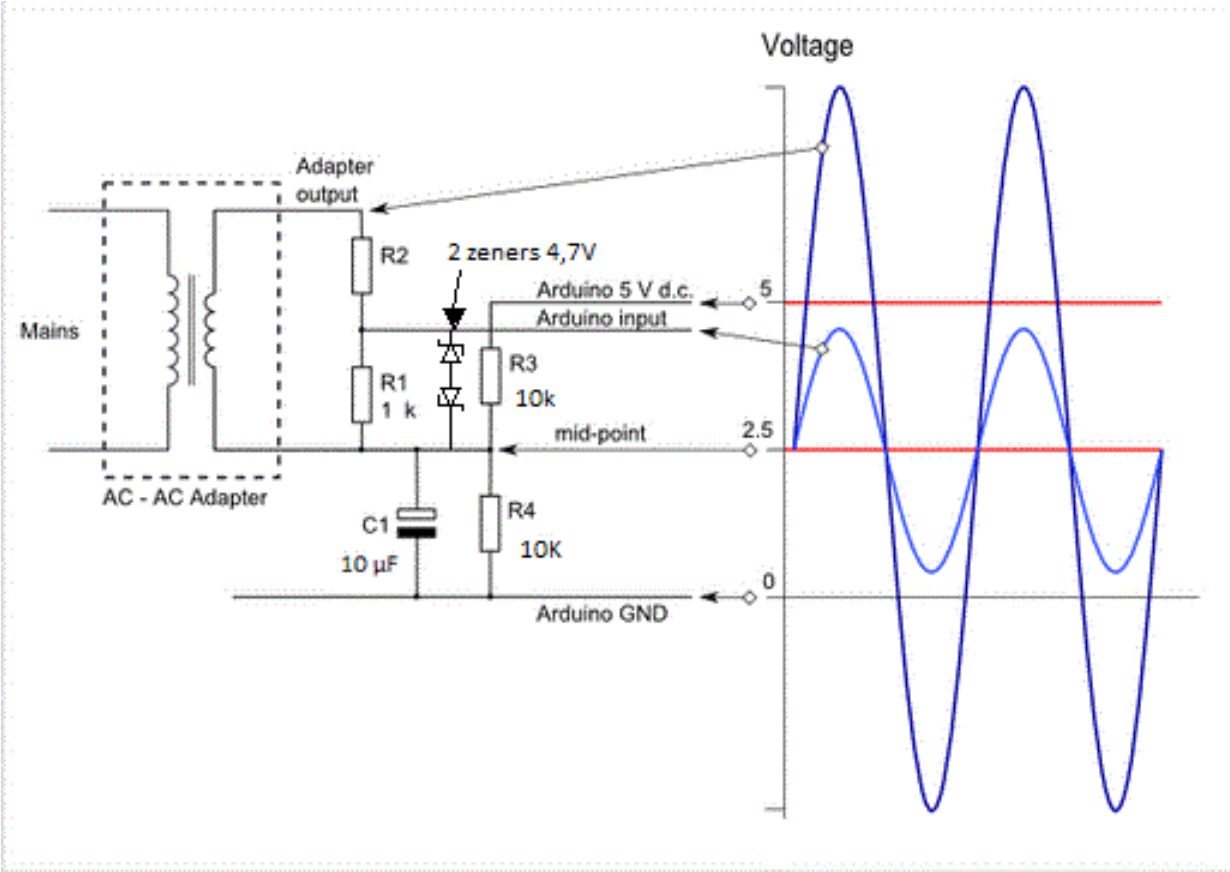
7- Une jolie boite pour intégrer durablement le montage (c'est d'ailleurs l'élément le très loin le plus cher)



Circuit électrique autour des capteurs

Les 2 capteurs fournissent chacun un signal sinusoïdal alternatif, qu'il faut adapter pour l'Arduino. En effet tout signal entrant sur l'Arduino doit rigoureusement être compris entre 0 et 5V. D'où l'idée de connecter les capteurs à mi-tension d'alimentation, soit 2,5V, en vérifiant que l'amplitude des signaux que délivreront les capteurs ne dépassent jamais le +5V ou deviennent négatives.

Mesure de tension

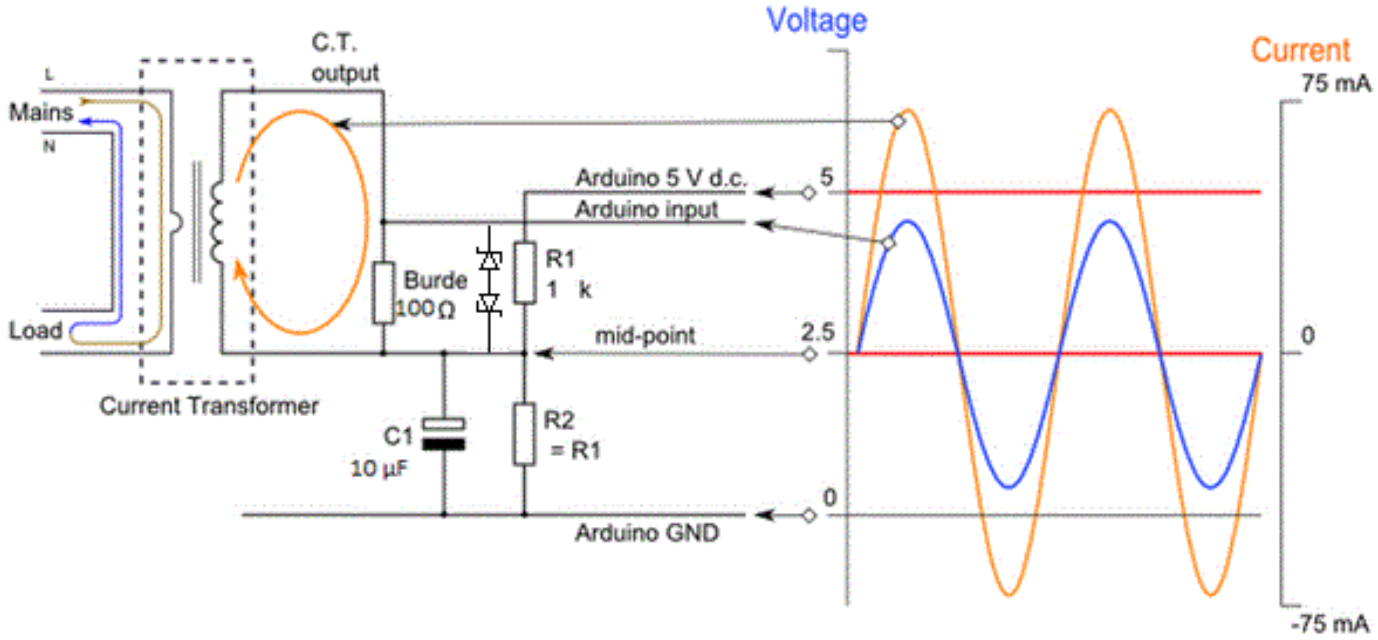


Il faut calculer R2 afin d'avoir R2 = Vtransfo eff *0,7 -1. Mesurer la tension à vide du transformateur avec un multimètre fournit la tension efficace.

R3 = R4, peu importe la valeur entre 1 et 47k
C1 entre 1 et 47uF.
2 zeners de 4,7V montées tête-bêche assurent une protection en cas de surtension.

La sortie de ce montage (Arduino input sur le schéma) est à relié à l'entrée A1 de l'Arduino .

Mesure de courant

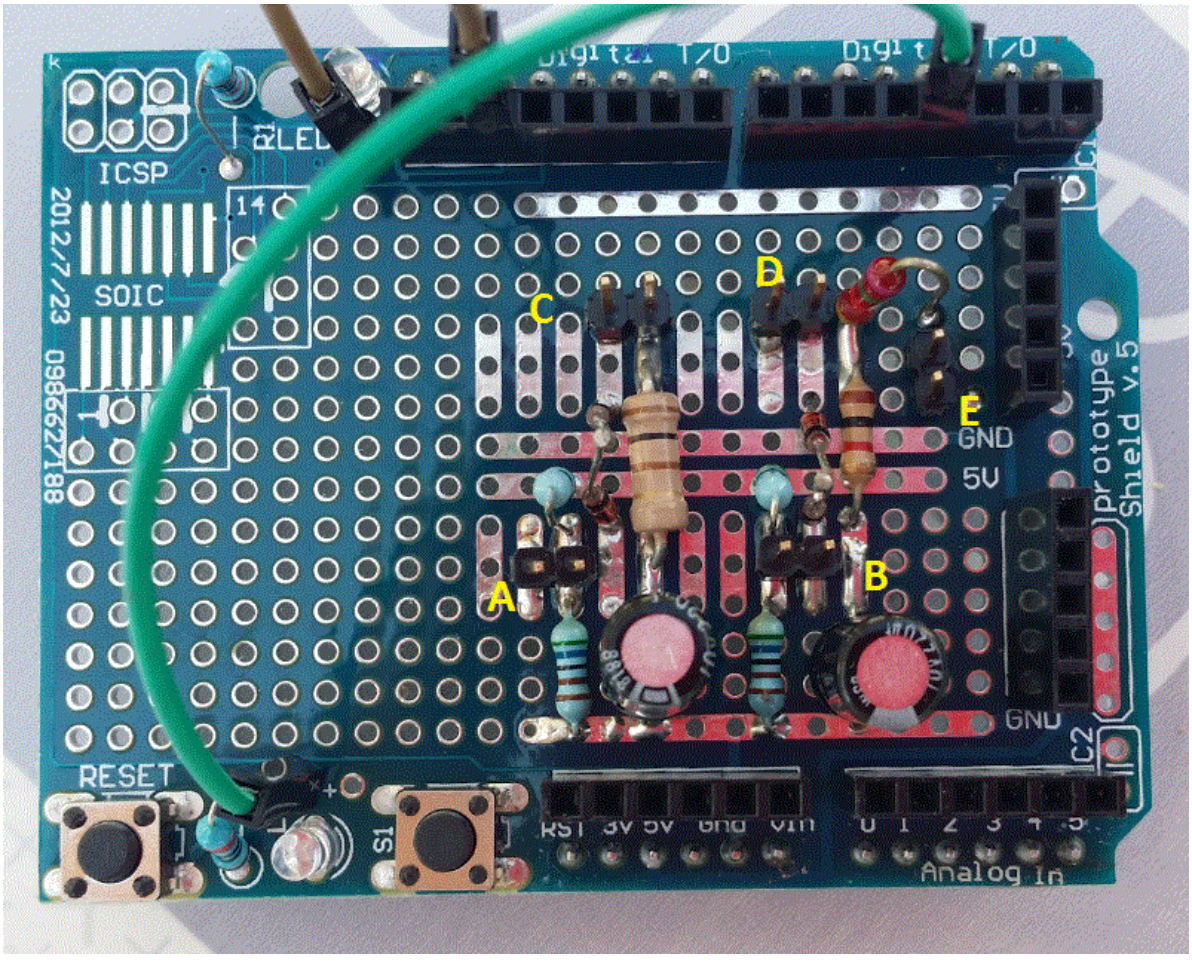


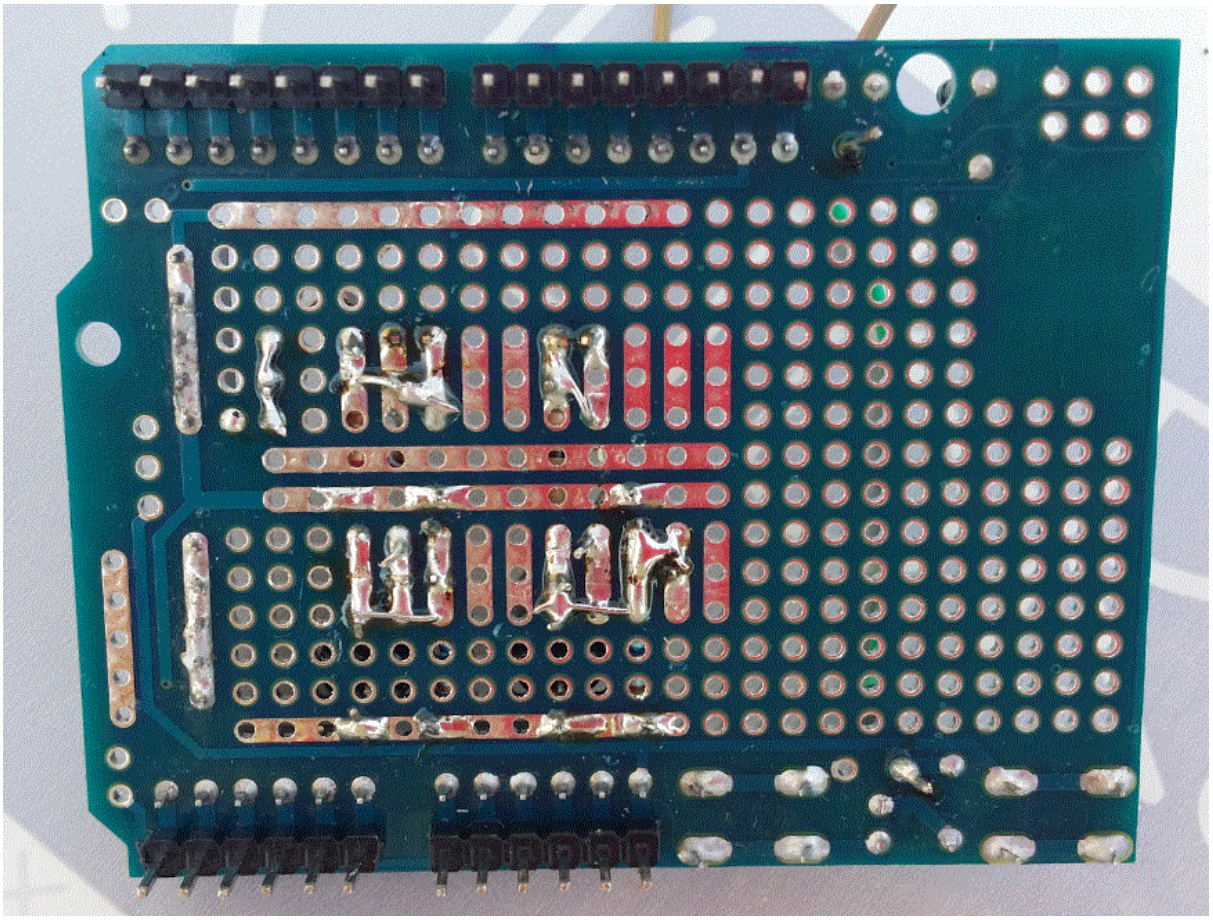
R1 = R2, peu importe la valeur entre 1 et 47k
C1 entre 1 et 47uF.
2 zeners de 4,7V montées tête-bêche assurent une protection en cas de surtension.

La sortie de ce montage (Arduino input sur le schéma) est à relié à l'entrée A0 de l'Arduino .

Quelques photos

On notera la présence de picots en A B C D E (en jaune sur la photo) pour le raccordement à l'aide de câbles Dupont





Premier test

1- Sans brancher les capteurs, enficher la carte d'extension dans l'Arduino, brancher l'Arduino sur le port USB du PC : Mesurer 2,5V sur les 2 bornes A et B (cf photo ci-dessus)

2- Brancher le capteur de courant entre A et C, et le capteur de tension entre B et E.

3- Connecter C en *entrée analogique 0*, et D en *entrée analogique 1*.

4- On connecte l'arduino au PC avec un câble USB.

5- Installer l'interface de programmation, le programme est téléchargeable ici : https://www.arduino.cc/en/Main/OldSoftwareReleases

6- On installe les drivers pour l'arduino UNO : http://283.mytrademe.info/ch340.html

7- On lance le programme Arduino :

1- Menu outils-> type de carte-> UNO

2- Menu outils-> PORT-> ComX ou X représente le port sur lequel est installé votre arduino.

3- On efface ce qu'il y a dans la fenêtre d'édition

4- On y colle le code ci-dessous :

```
// minMaxAndRangeChecker
// A simple tool to investigate the ADC values that are seen at the
// first four analogue inputs of an Atmega chip, as used on an emonTx
//
// Robin Emley (calypso_rae on the Open Energy Monitor forum)
//
// 20th April 2013
//

int val_a0, val_a1, val_a2, val_a3;
int minVal_a0, minVal_a1, minVal_a2, minVal_a3;
int maxVal_a0, maxVal_a1, maxVal_a2, maxVal_a3;

int loopCount = 0;
unsigned long timeAtLastDisplay = 0;
byte displayLineCounter = 0;
```

```
void setup(void) {
  Serial.begin(9600);
  Serial.print("ready ...");
  delay(700);
  Serial.println ();
  Serial.println(" The Min, Max and Range ADC values for analog inputs 0 to 3:");
}

void loop(void) {
  val_a0 = analogRead(0); //  CT2
  val_a1 = analogRead(1); //  CT3
  val_a2 = analogRead(2); //  Vsensor
  val_a3 = analogRead(3); //  CT1

  if (val_a0 < minVal_a0) { minVal_a0 = val_a0;}
  if (val_a0 > maxVal_a0) { maxVal_a0 = val_a0;}
  if (val_a1 < minVal_a1) { minVal_a1 = val_a1;}
  if (val_a1 > maxVal_a1) { maxVal_a1 = val_a1;}
  if (val_a2 < minVal_a2) { minVal_a2 = val_a2;}
  if (val_a2 > maxVal_a2) { maxVal_a2 = val_a2;}
  if (val_a3 < minVal_a3) { minVal_a3 = val_a3;}
  if (val_a3 > maxVal_a3) { maxVal_a3 = val_a3;}

  unsigned long timeNow = millis();
  if ((timeNow - timeAtLastDisplay) >= 3000) {
    timeAtLastDisplay = timeNow;
    displayVal(minVal_a0);
    displayVal(maxVal_a0);
    displayVal(maxVal_a0 - minVal_a0);
    Serial.print(";  ");

    displayVal(minVal_a1);
    displayVal(maxVal_a1);
    displayVal(maxVal_a1 - minVal_a1);
    Serial.print(";  ");

    displayVal(minVal_a2);
    displayVal(maxVal_a2);
    displayVal(maxVal_a2 - minVal_a2);
    Serial.print(";  ");

    displayVal(minVal_a3);
    displayVal(maxVal_a3);
    displayVal(maxVal_a3 - minVal_a3);
    Serial.println();

    resetMinAndMaxValues();
    displayLineCounter++;

    if (displayLineCounter >= 5) {
      Serial.println();
      displayLineCounter = 0;
      delay(2000); // to allow time for data to be accessed
    }
  }
}

void resetMinAndMaxValues() {
  minVal_a0 = 1023, minVal_a1 = 1023, minVal_a2 = 1023, minVal_a3 = 1023;
  maxVal_a0 = 0, maxVal_a1 = 0, maxVal_a2 = 0, maxVal_a3 = 0;
}
void displayVal(int intVal){
  char strVal[4];
  byte lenOfStrVal;
  itoa(intVal, strVal, 10); // decimal conversion to string
  lenOfStrVal = strlen(strVal); // determine length of string

  for (int i = 0; i < (4 - lenOfStrVal); i++)  {
    Serial.print(' ');
  }

  Serial.print(strVal);
}
```

Enregistrer ce programme sur le PC sous le nom de testminmax.ino par exemple.

Puis : Menu croquis -> téléverser.

Ouvrir le moniteur série : Menu outils -> moniteur série

Une autre fenêtre s'ouvre. Régler le débit (baud rate à 9600 si ce n'est pas le réglage par défaut)

Dans un premier temps on ne bascule pas l'inter 230V de la multiprise. L'arduino est uniquement alimenté (en 5V) par le port USB du PC et les valeurs de l'entrée tension et l'entrée courant sont donc égale à 0.
On voit apparaitre des valeurs dans le moniteur série :

Entrée A0

Première colonne = min en bits (de 0 à 1023)
Deuxième colonne = max (de 0 à 1023)
Troisième colonne = écart (max-min)

Entrée A1

Première colonne = min en bits (de 0 à 1023)
Deuxième colonne = max (de 0 à 1023)
Troisième colonne = écart (max-min)

Entrée A2 Etc etc

On doit lire des valeurs proches de 512 pour A0 at A1.

510 514 4 508 511 3

Les entrées analogiques passent par un convertisseur numérique qui fournit 0 à 1024 bits pour une valeur lue de 0 à 5V. A repos nous obtenons donc 512 bits, aux tolérances des valeurs de résistances près (5%).
Sinon c'est qu'il y a un problème sur le pont diviseur de tension de la voie concernée
A remarquer que le bruit des convertisseurs est de l'ordre de 4 bits.

Ensuite tester un appareil genre lampe halogène d'une centaine de Watts. Les valeurs doivent changer et se comporter à peu près comme ceci :

Sur A0 : 415 606 191

Sur A1 : 110 912 802

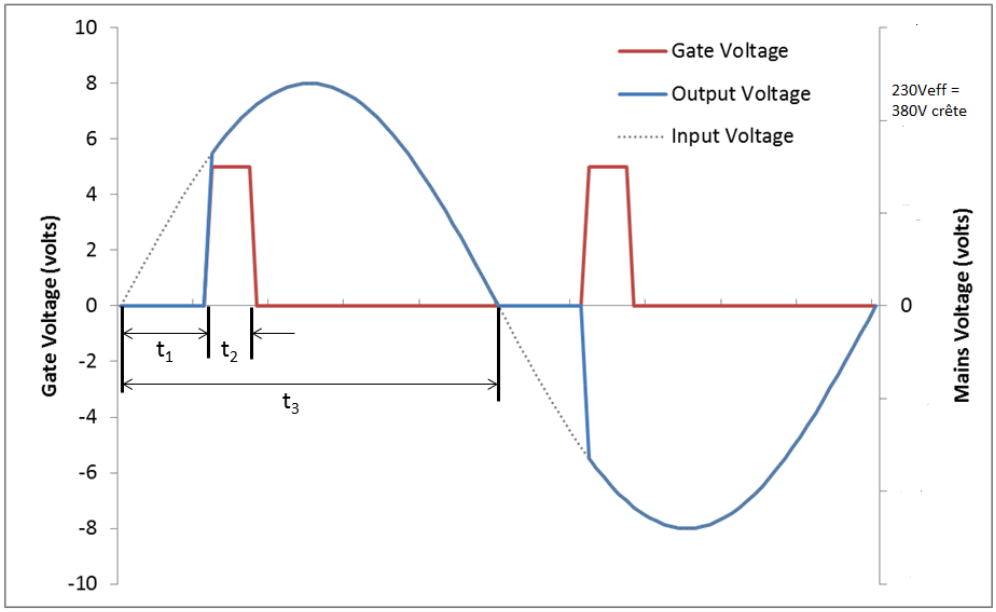
C'est tout bon ? parfait.

Le module triac

Le but du « jeu » est d'ouvrir plus ou moins un interrupteur pour déverser sur une résistance le trop d'énergie produit localement, afin de ne pas injecter sur le réseau public.

Principe de fonctionnement

L'usage d'un triac est idéal car il s'éteint tout seul à chaque passage à zéro de la sinusoïde de tension. Entre chaque passage à zéro il peut être déclenché (le fire) et plus c'est tôt plus il sera conducteur pour résistance, et inversement. Nous pouvons donc entrevoir qu'à chaque demi-alternance le programme va déclencher au bon moment pour juste délester le trop d'énergie produit, et surtout ne rien déverser lorsque ce n'est pas nécessaire.

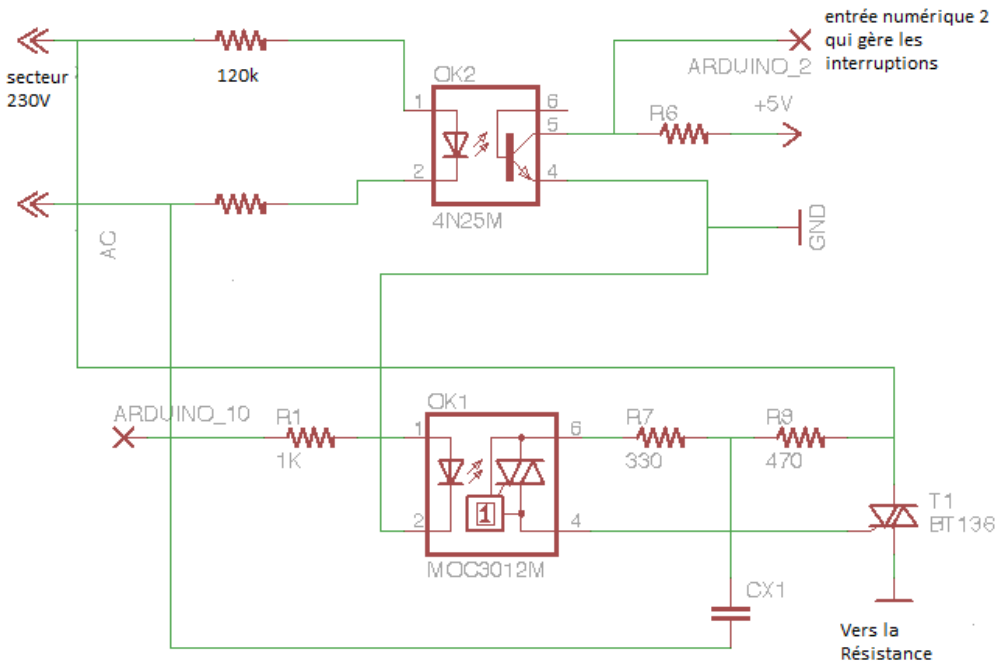


Par rapport au schéma, à chaque demi-alternance t3, à un instant t1 variable on procède au fire t2 qui rend le triac conducteur, jusqu'au prochain passage à zéro de la tension.

Ce dispositif oblige 2 contraintes :

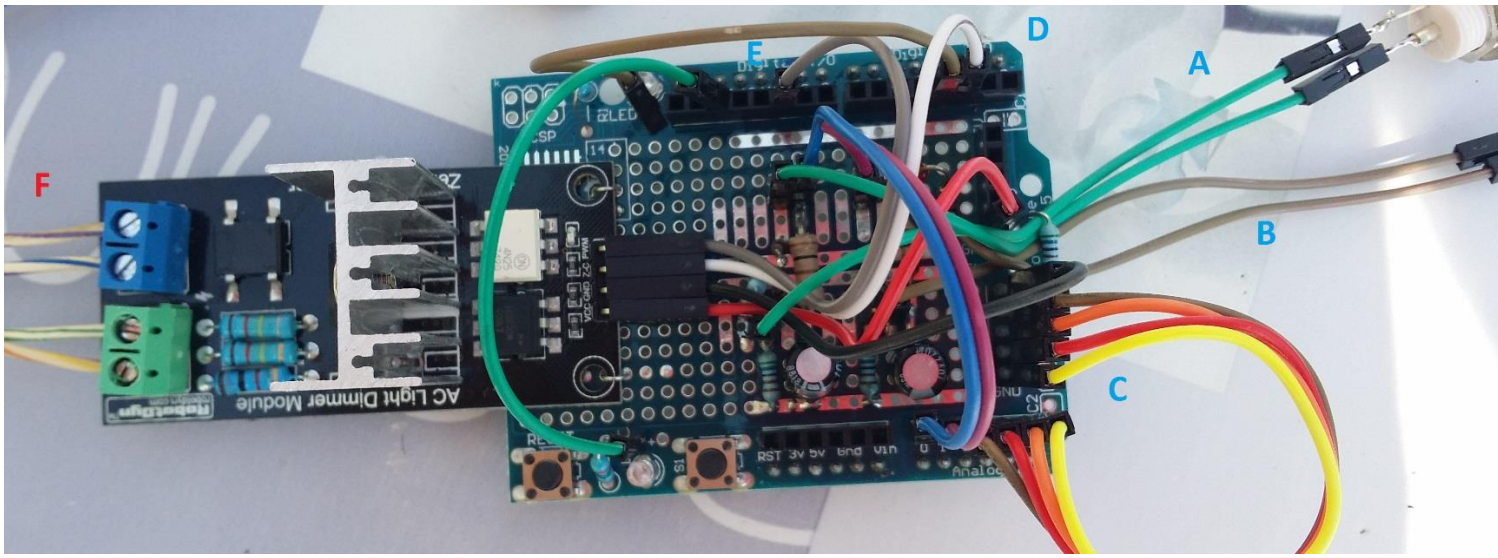
- Il faut un procédé pour détecter chaque passage à zéro. Pour l'instant sur l'Arduino nous n'avons comme mesure de tension qu'un nombre de bit de 0 à 1024. Il serait possible de créer un programme pour détecter chaque passage à zéro, mais ce genre de boucle prendrait beaucoup de ressources, ressources qui vont servir à bien autre chose….
Il va donc falloir un circuit supplémentaire pour détecter les passages à zéro.
- Le triac fonctionne sur le secteur, soit 230V, il est nécessaire de l'isoler pour que la phase ne se « balade » pas sur l'Arduino.

Le module triac va répondre à ces 2 besoins et reprend le schéma ci-dessous, avec l'usage d'optocoupleurs :



Il existe 2 familles d'optocoupleurs « simples » pour détecter le passage à zéro : les modèles avec 2 LEDs tête-bêche qui vont bien détecter chaque demi-alternance, et les modèles simple LED, auquel cas il faudra simplement ajouter en amont de la LED un pont redresseur de 4 diodes.

La photo du montage



On remarquera :

- A : la connexion pour la sonde de courant
- B : la connexion pour la sonde de tension
- C : les autres entrées analogiques sont reliées à la masse
- D : le fil du zéro-détecte sur l'entrée 2
- E : le fil de commande du triac sur la sortie 10
- F : bornier vert = arrivée 230V, bornier bleu = charge résistive

Deuxième test

L'astuce du programme est de gérer une interruption : le programme fait sa cuisine… et toutes les 10ms (pour un secteur à 50Hz) il reçoit une info sur l'entrée 2. Il arrête donc de touiller sa cuisine pour gérer cette interruption.

La 2ème astuce du programme est de définir un compteur système de 78us : 128 fois par 10ms (et donc par demi-cycle secteur) il sait qu'il va falloir s'occuper de quelque chose, à savoir d'un compteur temporel qui définie le moment du fire du triac.

Ces 2 astuces font appel à une librairie qu'il faut installer sur le PC : TimerOne.h
A voir ici : http://playground.arduino.cc/Code/Timer

1- Télécharger la librairie https://github.com/JChristensen/Timer/archive/v2.1.zip

2- La déclarer : (In the Arduino IDE) Sketch > Include Library > Add .ZIP Library > select the downloaded file > Open

3- Créer un nouveau programme avec le code ci-dessous :

```
/*
AC Light Control

  Philippe de Craene <dcphilippe@yahoo.fr> / May 2018
  merci à Ryan McLaughlin <ryanjmclaughlin@gmail.com>
  pour la partie commande du triac
  source : https://web.archive.org/web/20091121155320/http://www.arduino.cc:80/cgi-
bin/yabb2/YaBB.pl?num=1230333861/0

*/

#include <TimerOne.h>           // Avaiable from ttp://www.arduino.cc/playground/Code/Timer1

volatile int i=0;               // Variable to use as a counter
volatile boolean zero_cross=0;  // Boolean to store a "switch" to tell us if we have crossed
zero
int triac_pin = 10;             // Output to Opto Triac
int Dimmer_pin = 0;             // Pot for testing the dimming
int LED = 3;                    // LED for testing
int dim = 0, dimmax = 128;      // Dimming level (0-128)  0 = on, 128 = Off
int freqStep = 75;    // Set the delay for the frequency of power (65 for 60Hz, 78 for 50Hz) per
step (using 128 steps)
                      // freqStep may need some adjustment depending on your power the formula
                      // you need to us is (500000/AC_freq)/NumSteps = freqStep
                      // You could also write a seperate function to determine the freq

void setup() {                                          // Begin setup
 pinMode(triac_pin, OUTPUT);                            // Set the Triac pin as output
 pinMode(LED, OUTPUT);                                  // Set the LED pin as output
 attachInterrupt(0, zero_cross_detect, FALLING);   // Attach an Interupt to Pin 2 (interupt 0)
for Zero Cross Detection
 Timer1.initialize(freqStep);                           // Initialize TimerOne library for the freq
we need
 Timer1.attachInterrupt(dim_check, freqStep);           // Use the TimerOne Library to attach an
interrupt
                                                        // to the function we use to check to see if

it is
                                                        // the right time to fire the triac.  This

function
                                                        // will now run every freqStep in

microseconds.
 Serial.begin(9600);
 Serial.print("ready ...");
 delay(300);
 Serial.println ();
}                                                       // End setup

void zero_cross_detect() {          // function to be fired at the zero crossing
   zero_cross = 1;                  // set the boolean to true to tell our dimming function that
a zero cross has occured
}                                   // End zero_cross_detect

void dim_check() {                  // Function will fire the triac at the proper time
 if(zero_cross == 1) {   // First check to make sure the zero-cross has happened else do nothing
   if(i>=dim) {                     // Check and see if i has accumilated to the dimming value
we want
     digitalWrite(triac_pin, HIGH);    // Fire the Triac mid-phase
     delayMicroseconds(50);            // Pause briefly to ensure the triac turned on
     digitalWrite(triac_pin, LOW);     // Turn off the Triac gate
                                       //(Triac will not turn off until next zero cross)
     i = 0;                            // Reset the accumilator
```

```
     zero_cross = 0;                // Reset the zero_cross so it may be turned on again at the
next zero_cross_detect
   } else {
     i++;                     // If the dimming value has not been reached, incriment our counter
   }                          // End dim check
 }                            // End zero_cross check
}                             // End dim_check function

void loop() {                           // Main Loop
 while (dim> -1 ) {
       for (dim =0 ; dim <= dimmax ; dim++) {
           Serial.print(" valeur de dim = ");
           Serial.println(dim);
           analogWrite(LED, dim);             // Write the value to the LED for testing
           delay(100);
       }
       for (dim = dimmax ; dim > 0 ; dim--) {
           Serial.print(" valeur de dim = ");
           Serial.println(dim);
           analogWrite(LED, dim);             // Write the value to the LED for testing
           delay(100);
       }
 }
}
```

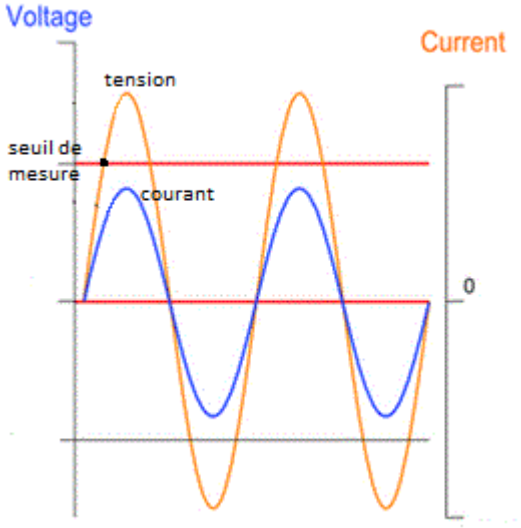
Enregistrer ce programme sur le PC sous le nom de testtriac.ino par exemple.
Puis : Menu croquis -> téléverser.
Ouvrir le moniteur série : Menu outils -> moniteur série
Une autre fenêtre s'ouvre. Il doit s'afficher la valeur de dim qui s'incrémente puis se décrémente de 0 à 128.

En branchant une ampoule à filament ou tout autre éclairage dimable celui-ci doit s'allumer puis s'éteindre progressivement, et infiniment.

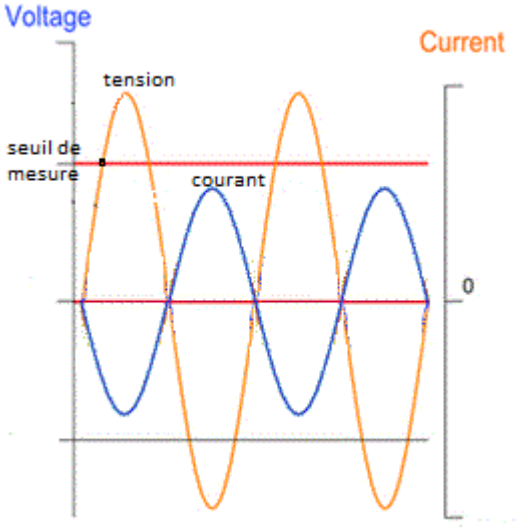
Ça marche ? on continue.

Mesure et traitement de la tension et du courant

Nous savons que la différence entre le mode injection et le mode consommation est la phase entre tension et courant, sachant que celle-ci se trouve en opposition de phase en mode injection.



mode consommation



mode injection

Le comment faire

Il serait possible de mesure cette phase avec un :
`valeurV = analogRead(voltageSensorPin)`
suivi d'un :
`valeurI = analogRead(currentSensorPin)`

Cependant il faut attendre un moment opportun pour réaliser cette opération, par exemple en effectuant la lecture du courant au moment où la tension atteint un seuil. Le problème est qu'il faut lire sans arrêt (en fait le plus vite possible) mesurer la tension dans l'espoir tomber sur le seuil qui autorisera la mesure de I. Tout cela oblige au recours d'une boucle de test qui prend du temps, temps qui doit servir à autre chose.

Ensuite il se pose 2 autres problèmes, les jolies sinusoïdes ci-dessus sont du monde des bisounours, au moins pour celle du courant :

- D'abord il y a toujours un déphasage entre courant et tension, ce fameux cos φ, parce que nos équipements ne sont que rarement des résistances pures, il y a des moteurs (selfique) des Leds ou du fluorescent (capacitif) qui font que le courant sera *toujours* en avance ou en retard de phase par rapport à la tension. Le résultat de la mesure ponctuelle donne donc une valeur de courant qui ne correspond pas à la réalité du courant consommé.
- Pire ! Il y a quantité d'appareils qui ne fonctionnent pas en régime linéaire, mais en régime impulsionnel : Il y a des alimentations à découpage (télévision, ordinateur) qui « s'amusent » à découper la tension – en donc le courant – en tronçons à des vitesses aussi rapides que variables, et les triacs (lave-linge, le Power Router) qui, lors du fire fait un appel en courant qui fait comme des cornes sur la courbe de courant, et ce à chaque demi-période secteur….

La mesure ponctuelle n'étant pas du tout adaptée, nous allons utiliser la même méthode que les programmes qui transforment l'Arduino en appareil de mesure de tensions alternatives : un procédé qui fait une acquisition sur un laps de temps (paramétrable) et produit un résultat cohérent après traitement.

La solution utilisée et le troisième test

La mesure du courant alternatif – et aussi de tous les autres paramètres : tension alternative, puissance utile, puissante apparente, facteur de puissance - fait appel à une librairie qu'il faut installer sur le PC : EmonLib.h
A voir ici : https://learn.openenergymonitor.org/electricity-monitoring/ctac/how-to-build-an-arduino-energy-monitor

1- Télécharger la librairie https://github.com/openenergymonitor/EmonLib/archive/master.zip
2- La déclarer : (In the Arduino IDE) Sketch > Include Library > Add .ZIP Library > select the downloaded file > Open
3- Créer un nouveau programme avec le code ci-dessous :

```
// EmonLibrary examples openenergymonitor.org, Licence GNU GPL V3

#include "EmonLib.h"             // Include Emon Library
EnergyMonitor emon1;             // Create an instance

void setup()
{
  Serial.begin(9600);

  emon1.voltage(1, 155, 1.7);  // Voltage: input pin, calibration, phase_shift
  emon1.current(0, 105);       // Current: input pin, calibration.


 Serial.println("ready ...");
 Serial.println ();
 delay(500);
 Serial.println("   V   /   I   /  Peff   /   Pi    (en Volts/Ampères/Watts/VA/)");
 Serial.println();
}

void loop()
{
  emon1.calcVI(20,200);         // Calculate all. No.of half wavelengths (crossings), time-out
//  emon1.serialprint();           // Print out all variables (realpower, apparent power, Vrms,
Irms, power factor)

  float realPower       = emon1.realPower;        //extract Real Power into variable
  float apparentPower   = emon1.apparentPower;    //extract Apparent Power into variable
  float powerFactor     = emon1.powerFactor;      //extract Power Factor into Variable
  float supplyVoltage   = emon1.Vrms;             //extract Vrms into Variable
  float Irms            = emon1.Irms;             //extract Irms into Variable

    Serial.print("   ");
    Serial.print(supplyVoltage);
    Serial.print("  |  ");
    Serial.print(Irms);
    Serial.print("  |  ");
    Serial.print(realPower);
    Serial.print("  |  ");
    Serial.print(apparentPower);
    Serial.print("  |  ");
    Serial.println(powerFactor);
}
```

Enregistrer ce programme sur le PC sous le nom de testmesuresAC.ino par exemple.
Puis : Menu croquis -> téléverser.
Ouvrir le moniteur série : Menu outils -> moniteur série
Une autre fenêtre s'ouvre. Il doit s'afficher la valeur de la tension, du courant, de la puissance utile, de la puissance apparente, du facteur de puissance.

En effectuant la mesure sur un appareil en marche, il est possible de régler les 2 paramètres de calibration pour :

`emon1.voltage(1, 155, 1.7)`
ici c'est 155, choisir une valeur pour s'approcher d'une lecture de 230V

`emon1.current(0, 105)`
ici c'est 105, idem choisir une valeur pour que la valeur de puissance utile s'approche de la puissance réelle consommée.

Le programme fonctionnel

Il suffit de rassembler les morceaux dédiés au fonctionnement du triac et de la lecture du courant, puis d'ajouter le traitement du délai de déclenchement du triac en fonction de la puissance injectée.

```
/*

Power_Router est un système qui permet d'utiliser l'excédent d'énergie autoproduit par
l'allumage d'un appareil résistif (facteur de puissance proche de 1) ce qui évite l'injection
au réseau public de distribution d'électricité.

Le principe de fonctionnement est le suivant :
- détection de phase entre courant et tension permet de savoir si on consomme ou bien on injecte
- en cas d'injection il se produit la mise en route progressive d'un dispositif d'absorption
d'excédent de puissance
- la mesure du courant permet d'ajuster au mieux le niveau d'absorption de cet excédent.

le programme prévoit :
- une sonde de tension : simple transfo 230V/5V Crête à Crête sur mi-tension (2.5V)
- une sonde de courant : 20A/25mA sur mi-tension (2.5V)
- un module de commande par triac
- un dispositif de détection de passage à zéro de la sinusoïde de tension secteur (par exemple
l'optocoupleur H11AA1)
- la bibliothèque TimeOne.h à installer et disponible là :
http://www.arduino.cc/playground/Code/Timer
- la bibliothèque EmonLib.h à installer et disponible là :
https://github.com/openenergymonitor/EmonLib/archive/master.zip

La gamme de puissances testée est va de 300 à 1000W

merci à Ryan McLaughlin <ryanjmclaughlin@gmail.com> pour avoir étudié et mis au point la partie
commande du triac
il y a quelques années et que j'ai repris dans ce programme :)
source : https://web.archive.org/web/20091212193047/http://www.arduino.cc:80/cgi-
bin/yabb2/YaBB.pl?num=1230333861/15

_____________________________________________________________
|                                                           |
|       auteur : Philippe de Craene <dcphilippe@yahoo.fr    |
|           pour l' Association P'TIWATT                    |
_____________________________________________________________

Toute contribution en vue de l'amélioration de l'appareil est la bienvenue ! Il vous est juste
demandé de conserver mon nom et mon email dans l'entête du programme, et bien sûr de partager
avec moi cette amélioration. Merci.

chronologie des versions :
version 0.5 - 3 mai 2018     - boucle de décrémentation dim --
version 0.8 - 5 juil. 2018   - 1ère version fonctionnelle, pb du pic de courant du triac
version 1   - 6 juil. 2018   - ajout de la bibliothèque EmonLib.h pour mesure du secteur
version 1.4 - 7 juil. 2018   - simplification des tests sur sPower et dim. Version film youtube
version 1.6 - 8 juil. 2018   - ajout LED d'overflow + optimisation des paramètres + seuilPoff
version 1.8 - 24 sept 2018   - ajout du pas variable sur dim avec dimstep

*/

#include <TimerOne.h>           // librairie à installer depuis
http://www.arduino.cc/playground/Code/Timer1
#include <EmonLib.h>            // librairie à installer depuis
https://github.com/openenergymonitor/EmonLib/archive/master.zip

EnergyMonitor emon1;            // Create an instance

// détermination des entrées / sorties :

byte triac_pin         = 10;    // sortie numérique pour le Triac
byte triacLED          = 3;     // sortie numérique pour la LED de test du triac
byte limiteLED         = 13;    // sortie numérique pour la LED d'overflow
byte voltageSensorPin  = 1;     // détecteur de tension entrée analogique 1
byte currentSensorPin  = 0;     // détecteur de courant entrée analogique 0
byte zeroCrossPin      = 2;     // détecteur de phase entrée digitale 2

// variables de gestion des interruptions (zero-crossing) :

int dimmax = 128;          // valeur max de dim pour inhiber le triac
int dim = dimmax;          // Dimming level (0-128)  0 = on, 128 = Off
```

```
char periodStep = 75;      // détermine la période du timer (65 pour 60Hz, 78 pour 50Hz, en µs)
                           // suivant la formule (500000/AC_freq)/NumSteps = periodStep
                           // 78*128=10ms=1/2 période 50Hz mais aux tests 76 marche mieux

volatile int i = 0;             // Variable to use as a counter
volatile boolean zero_cross=0;  // Boolean to store a "switch" for crossed zero

// variables de gestion de la mesure de puissance (seuls valeurI et rPower nous intéressent) :

long valeurI;                   // extract Irms into Variable (long supprime la « , »)
long rPower = 0;                // extract Real Power into variable
int dimstep;                    // valeur de l'écart de dim


// variable de calibration
                                // valeurs en mW (milliwatts) qui déterminent les seuils :
int seuilP = 3000;              // l'hystérésis d'asservissement : 3000 = 3W => hystérésis à 6W
int seuilPoff = 25000;          // seuil d'arrêt instantané en cas de chute de production


//
// SETUP
//_____________________________________________________________________________________________

void setup() {                  // Begin setup
 pinMode(triac_pin, OUTPUT);    // Set the Triac pin as output
 pinMode(triacLED, OUTPUT);     // Set the LED pin as output
 pinMode(limiteLED, OUTPUT);

attachInterrupt(digitalPinToInterrupt(zeroCrossPin), zero_cross_detect, RISING);
                            // en cas d'interruption lance 'zero_cross_detect' en mode RISING
                            // à chaque changement de LOW à HIGH de zeroCrossPin.
// documentation : https://www.arduino.cc/reference/en/language/functions/external-
interrupts/attachinterrupt/

 Timer1.initialize(periodStep);                    // initialisation de la librairie TimerOne
 Timer1.attachInterrupt(dim_check, periodStep);    // Use the TimerOne Library to attach an
interrupt
                                          // to the function we use to check to see if it is
                                          // the right time to fire the triac.  This function
                                          // will now run every periodStep in microseconds.

emon1.voltage(voltageSensorPin, 200, 1.7);      // Voltage: input pin, calibration, phase_shift
emon1.current(currentSensorPin, 8000);          // Current: input pin, calibration

// comparer l'affichage de la console avec celui d'un wattmètre pour s'approcher de la réalité.
// Sinon faire le test avec une charge connue, par exemple une ampoule tungstène de 60W
// Les valeurs ne sont pas critiques, l'ordre de grandeur est 200 pour la tension et
// 8000 pour avoir le courant en mA (milliampères)

 Serial.begin(9600);
 Serial.println ();
 Serial.println("ready ...");
 Serial.println ();
 delay(500);
 Serial.print("   I (mA)  |   Pu (mw)  ||   dimstep   |    dim    )");
 Serial.println();
}               // End setup


//
// ZERO CROSS DETECT : gestion du passage à zéro par interruption
//_____________________________________________________________________________________________

void zero_cross_detect() {       // function to be fired at the zero crossing
   zero_cross = 1;               // set the boolean to true to tell our dimming function that a
zero cross has occured
}                                // End zero_cross_detect


//
// DIM CHECK : gestion du triac par interruption
//_____________________________________________________________________________________________

void dim_check() {                          // Function will fire the triac at the proper time
 if(zero_cross == 1 && dim < dimmax) {      // First check to make sure the zero-cross has
                                            // happened else do nothing
                                            // ET inutile de compter le fire ça doit être éteint
     if(i>dim) {            // i est un compteur qui détermine le retard au fire. plus dim
                            // est élevé, plus de temps prendra le compteur i et plus tard
       digitalWrite(triac_pin, HIGH);       // se fera le fire du triac
       delayMicroseconds(50);               // Pause briefly to ensure the triac turned on
       digitalWrite(triac_pin, LOW);        // Turn off the Triac gate, le triac reste conducteur
       i = 0;                               // Reset the accumilator
       zero_cross = 0;                      // Reset the zero_cross so it may be turned on again
at the next zero_cross_detect
       }
```

```
    else { i++; }           // If the dimming value has not been reached, incriment our counter
  }                         // End zero_cross check
}                           // End dim_check function


//
// LOOP : programme principal (qui tourne en boucle)
//_____________________________________________________________________________________________

void loop() {                   // Main Loop

// Ci-dessous est le relevé des mesures du courant et de la puissance utile
// rPower indique une polarité et donc permet de savoir si on consomme ou si on injecte.
 valeurI      = emon1.Irms;              //extract Irms into Variable
 rPower       = emon1.realPower;         //extract Real Power into variable

 emon1.calcVI(20,200);          // Calculate all. No.of half wavelengths (crossings), time-out
                                // d'origine emon1.calcVI(20,2000)

 dimstep = abs(rPower/20000)+1;           // le pas dépend de la puissance en jeu

 if( rPower < -seuilPoff ) { dim = dimmax; }   // arrêt en cas de chutte brusque de production

 else {                         // test de polarité => puissance positive en mode injection
   if( rPower > seuilP ) {      // l'injection augmente, on diminue le délai d'allumage du triac
      if( dim > dimstep ) { dim -= dimstep; } else { dim = 0;}}
   else if( rPower < -seuilP ) {               // moins de prod : on baisse la charge
           if( dim - dimstep < dimmax ) { dim += dimstep; } else { dim = dimmax; }}
 }

  if (dim < 1) { digitalWrite(limiteLED, HIGH); }    // led témoin de surcharge
  else { digitalWrite(limiteLED, LOW); }
  analogWrite(triacLED, dimmax-dim);                 // Write the value to the LED for testing

 // affichage de ce qui se passe / à décommenter dans le cadre du debugging
    Serial.print("   ");
    Serial.print(valeurI);
    Serial.print("  |  ");
    Serial.print(rPower);
    Serial.print("  ||  ");
    Serial.print(dimstep);
    Serial.print("  |  ");
    Serial.println(dim);

}                               // fin de Main Loop
```

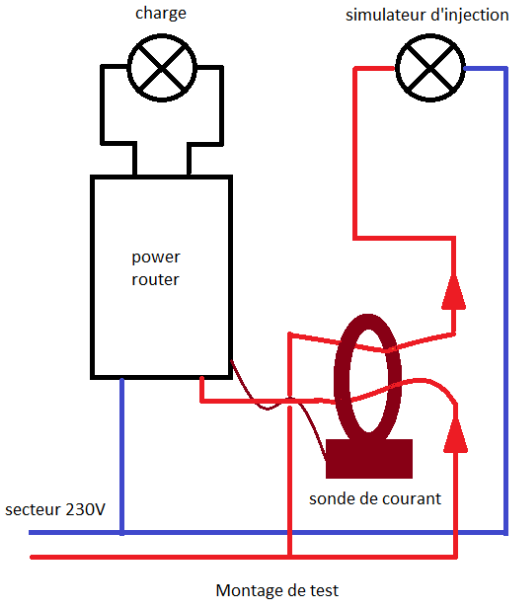
Illustration des essais

Les essais ont été réalisés avec :

- seuilP = 3000, ce qui implique une variabilité de 6000 de P (6Watts)
- emon1.calcVI(20,200) soit acquisition sur 10 périodes, et mesure toutes les 200ms

Remarque : il y a un compromis entre rapidité de réaction au changement de charge et/ou de production et stabilité des mesures. En choisissant un temps d'acquisition plus long, le temps de réaction est proportionnellement plus lent, mais les données récoltées moins sujettes à variations.

Schéma de câblage



Montage de test

Tout est question de phase de courant : pour simuler une injection, il faut que le sens du courant soit inverse du sens du courant de consommation. Ainsi le champ magnétique recueilli par la sonde sera la différence entre le champ magnétique crée par la consommation et le simulateur d'injection.

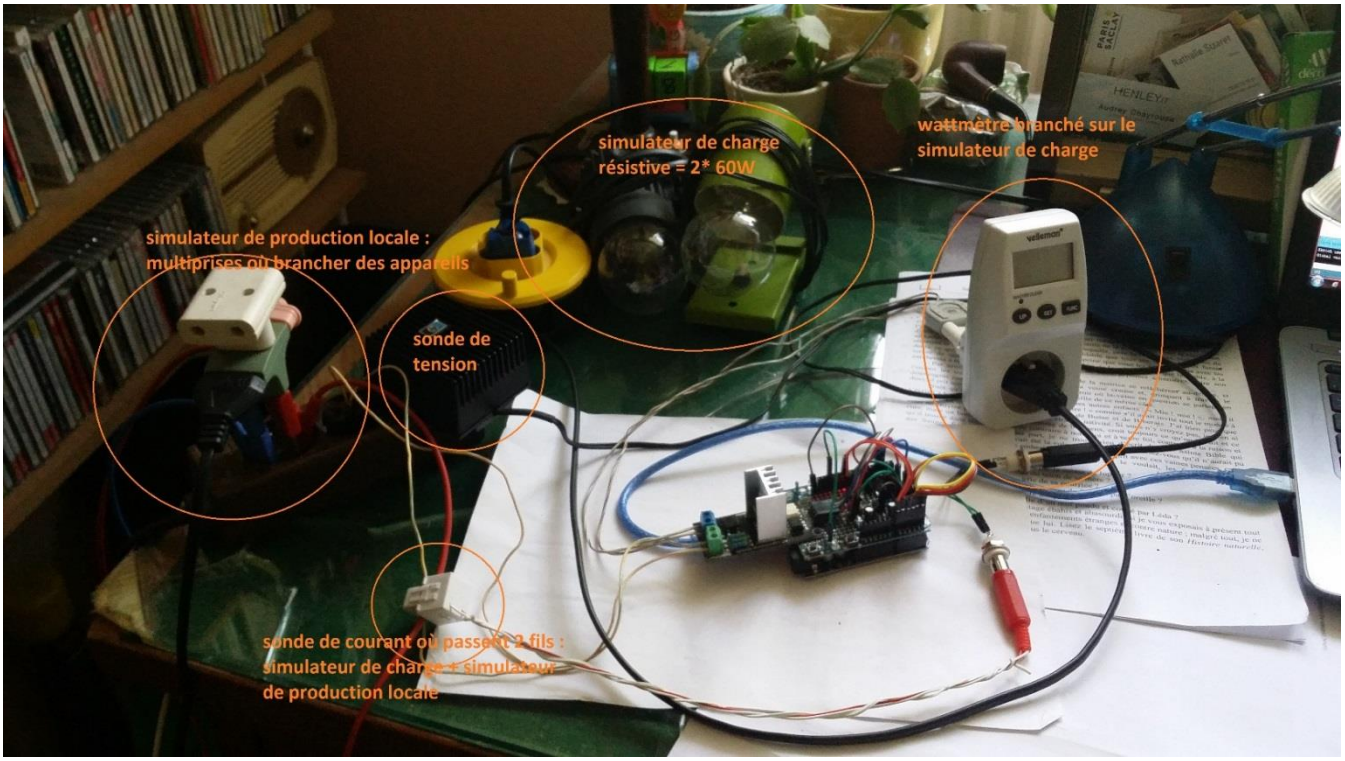
Dans le montage, la consommation est le power router avec sa charge. La simulation d'injection est une ou plusieurs ampoules électriques à filaments.

Il y a une chance sur deux pour que la sonde soit branchée dans le bon sens : si après mise en marche et présence d'une simulation d'injection la charge devient totale en permanence, il suffit d'inverser la sonde pour que le montage fonctionne correctement.

Cas initial : Il n'y a rien côté production, charge de 160W

La console indique :

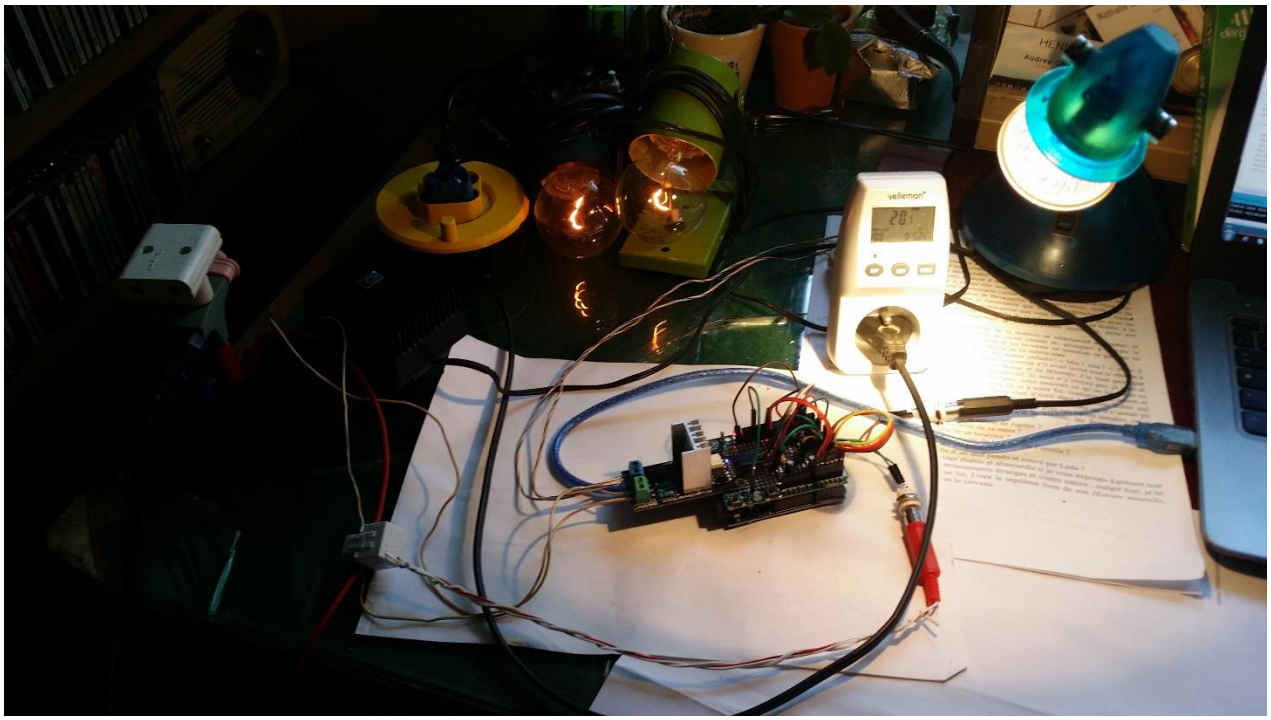
I entre 33 et 40 P entre -2000 et +2000 dim = 128
Le wattmètre est branché sur la charge : là bien sûr il indique 0



Cas 1 : on simule une production de 25W, charge de 160W

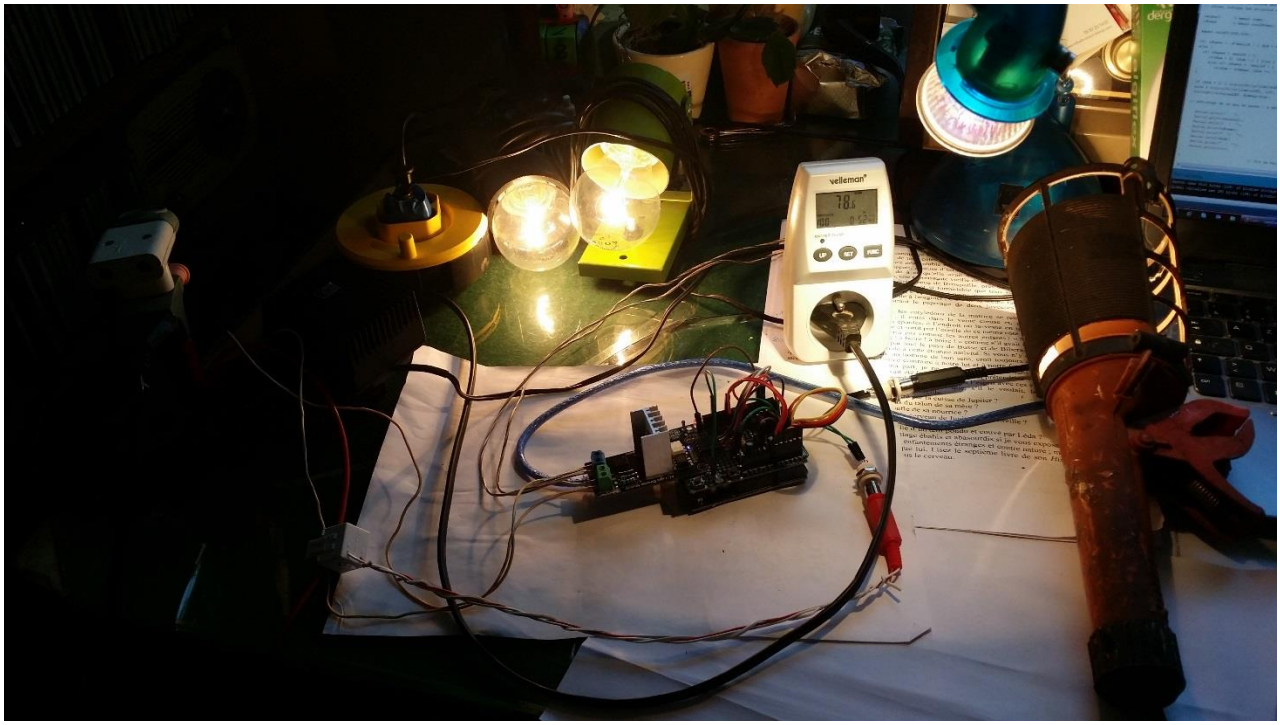
I entre 240 et 400 P entre -4000 et +4000 dim = 97
Wattmètre = 22W

Remarque : le sens de la sonde de courant détermine la polarité : si la charge « monte » à fond, il suffit de retourner la sonde de courant…



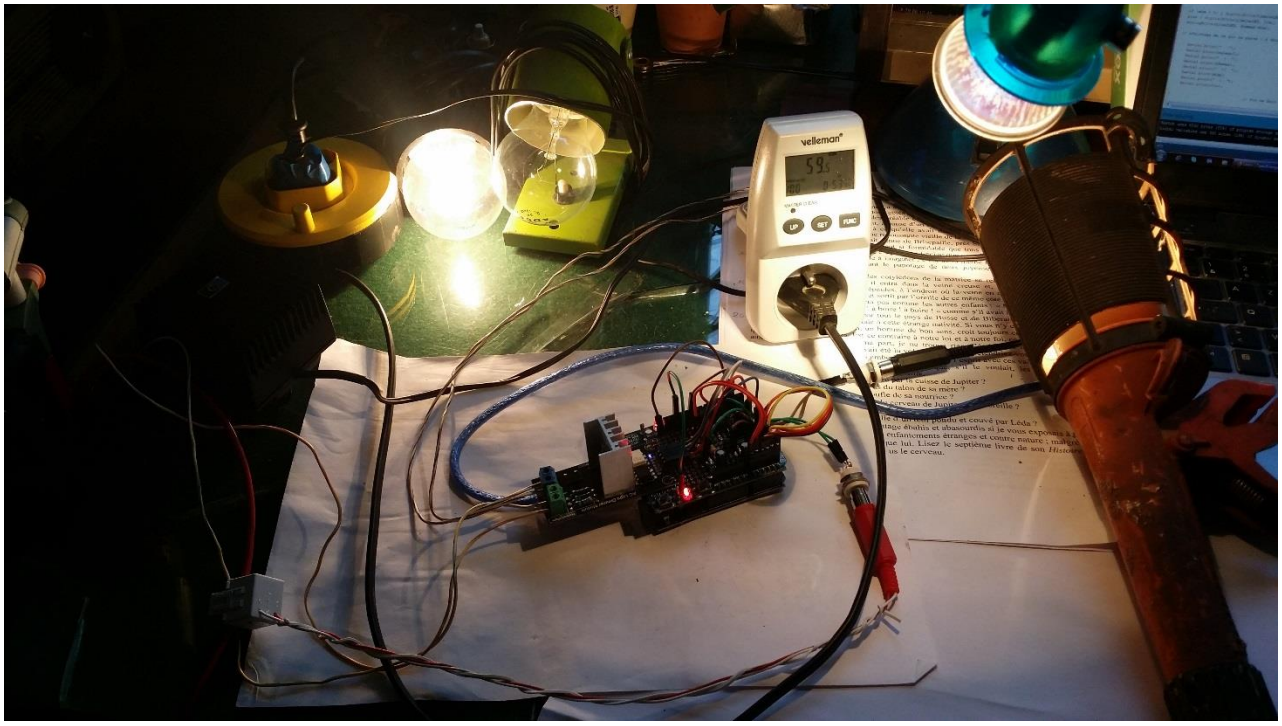
Cas 2 : on simule une production de 85W, charge de 160W

I entre 260 et 290 P entre -10000 et +9000 dim = 56
Wattmètre = 85W



Cas 3 : on simule une production de 85W, charge de 60W

I entre 140 et 160 P entre -27000 et +30000 dim = 0
Wattmètre = 59W
On remarque la LED d'overflow qui est allumée.
On remarque également que la variation des mesures est bien plus faible que dans les 2 cas précédents, ici le triac étant déclenché dès le début, le pic de courant est quasi inexistant.



Cas 4 : on simule une production de 25W, charge de 60W

I entre 160 et 300 P entre -4000 et +4000 dim = 78
Wattmètre = 26W

Cas similaire au cas 1 sauf que la charge est réduite de moitié. La dim fait déclencher plus tôt pour compenser une résistance de charge plus élevée.

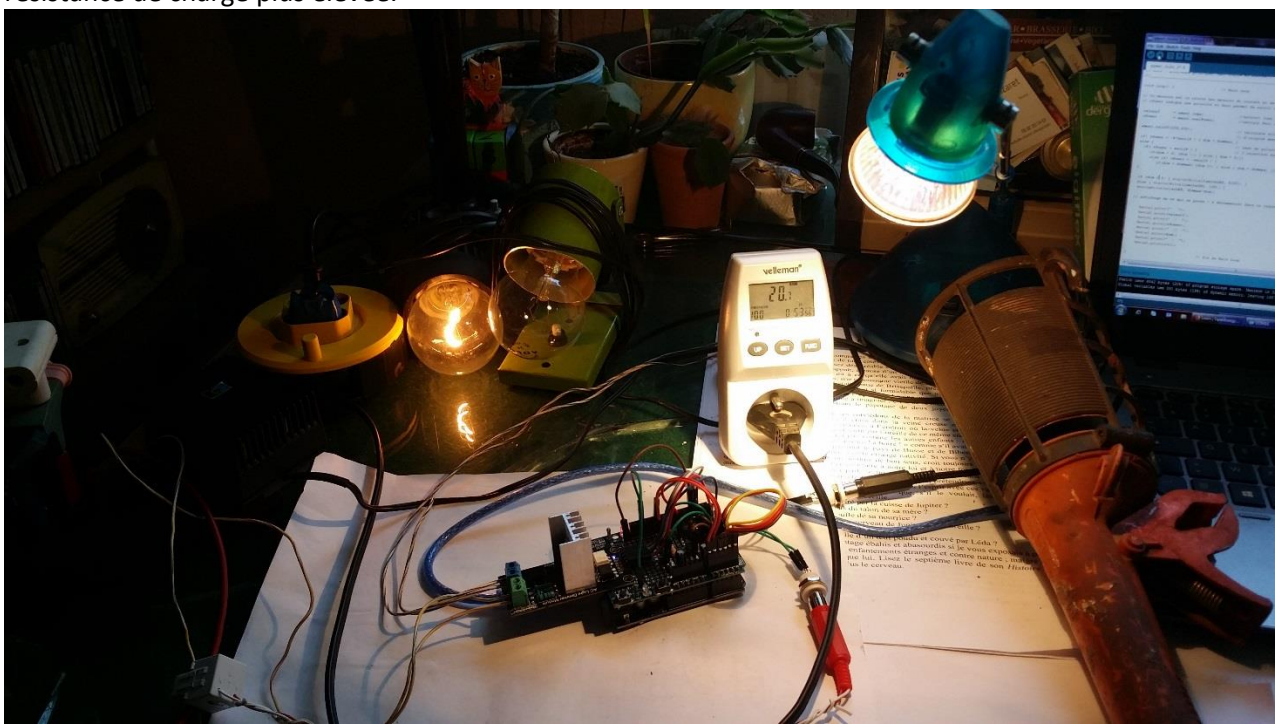
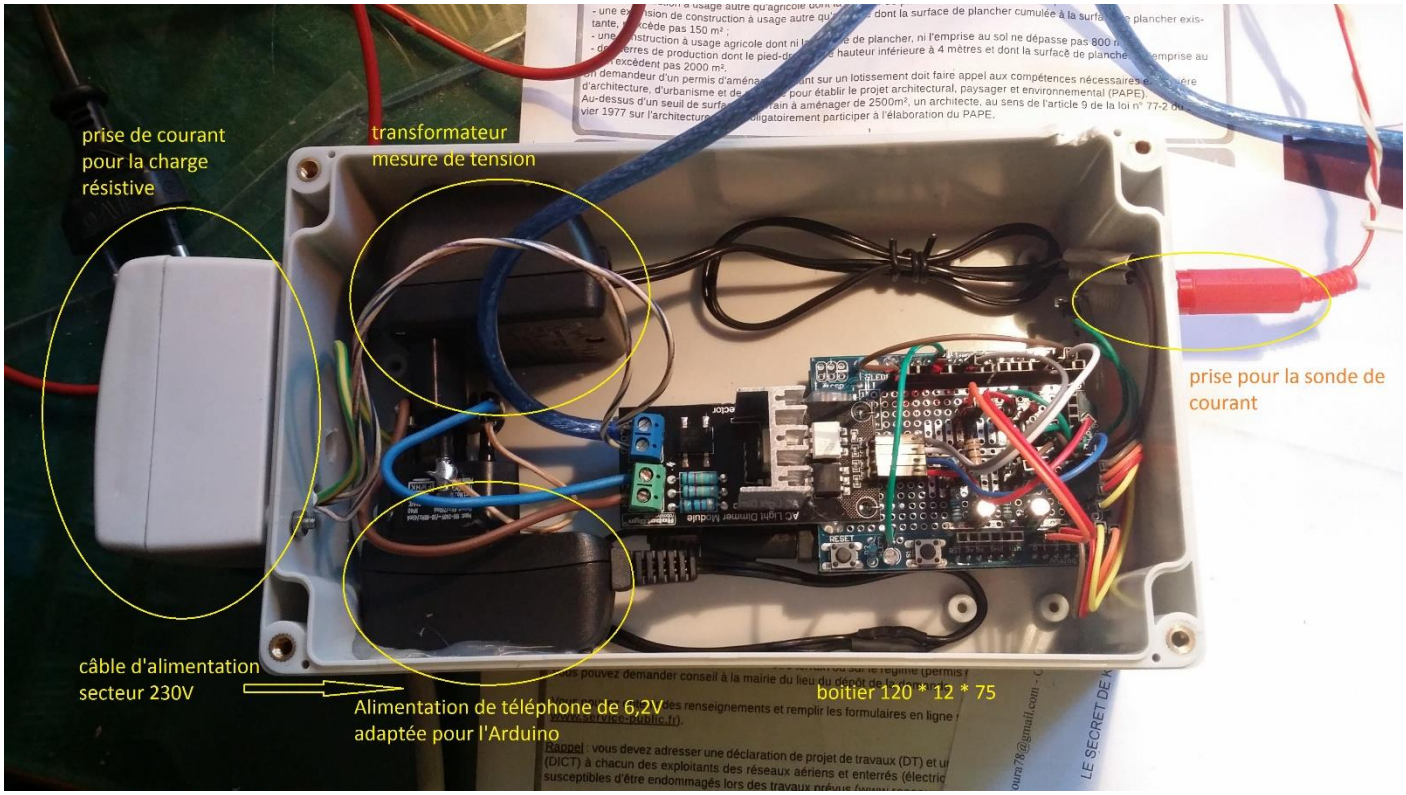
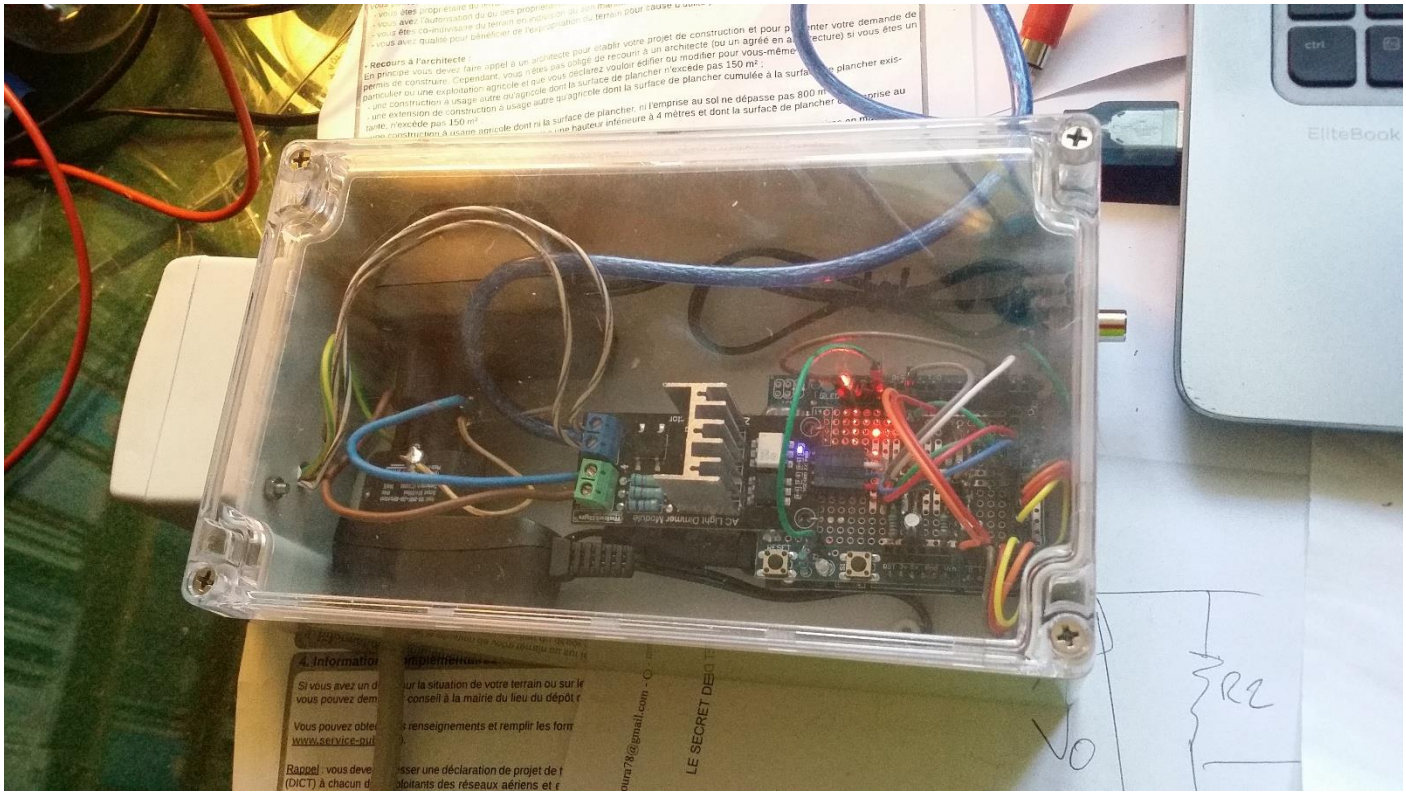
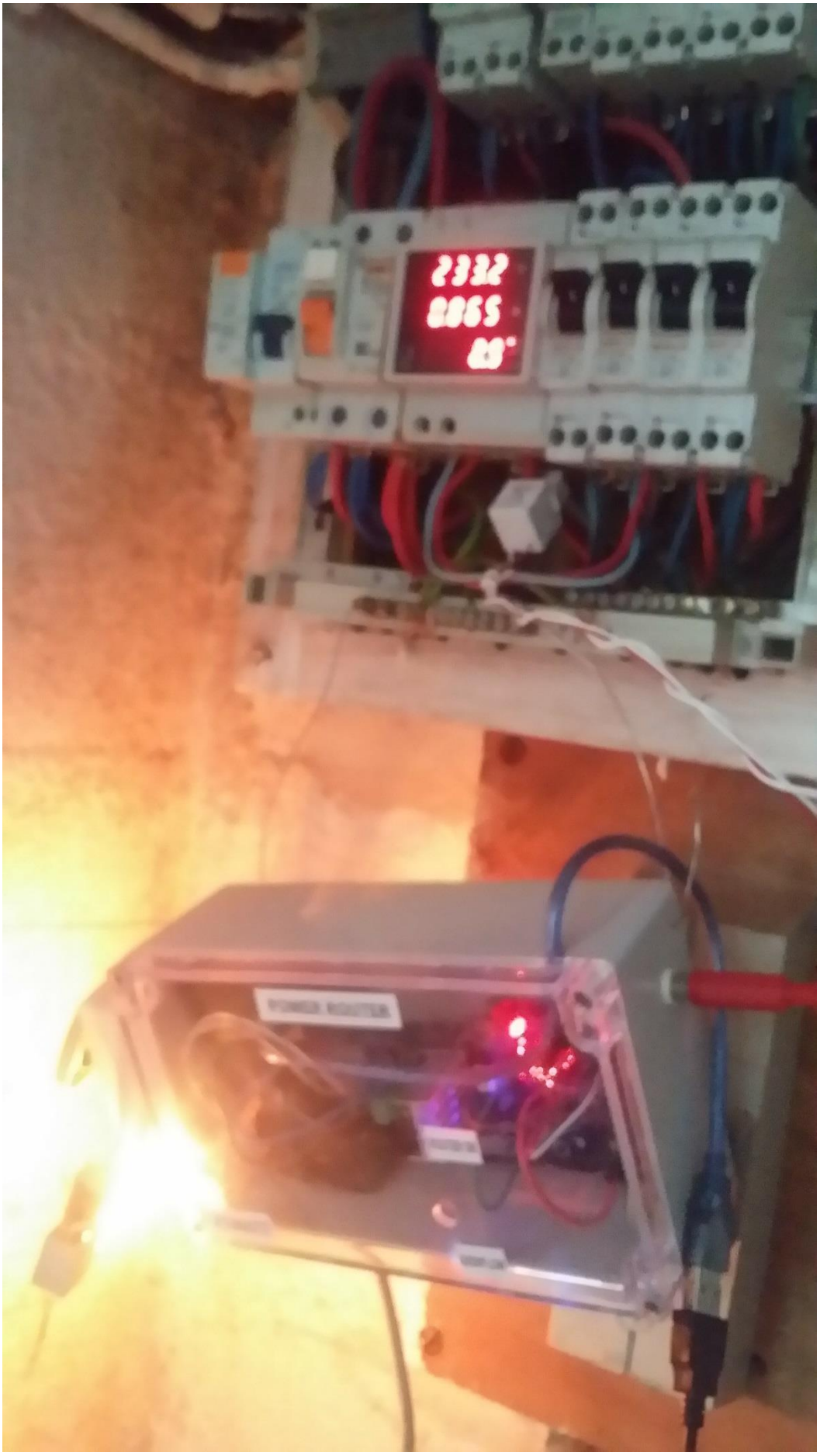


Illustration de mise en boite





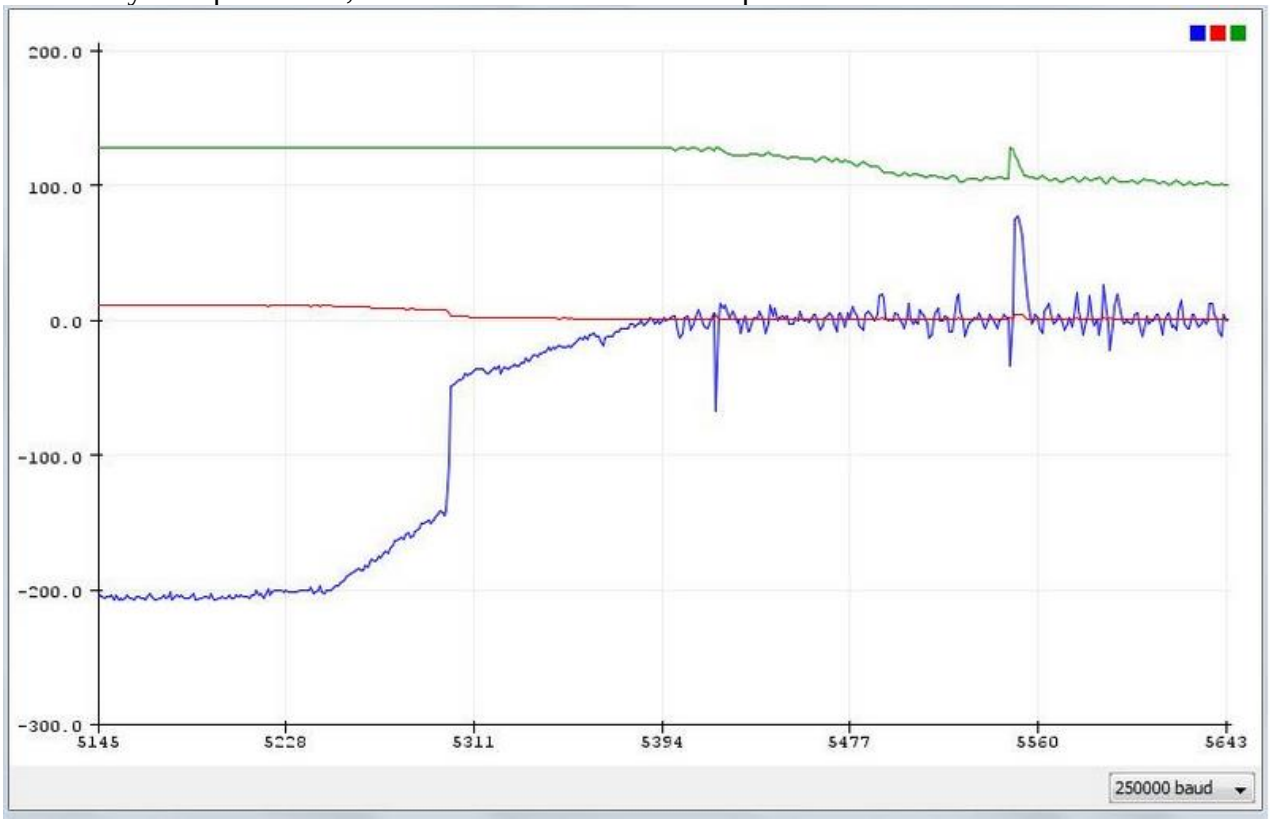
Tests concrets



Voici ci-après 2 relevés de mesures de respectivement rpower/1000 , dimstep et dim :

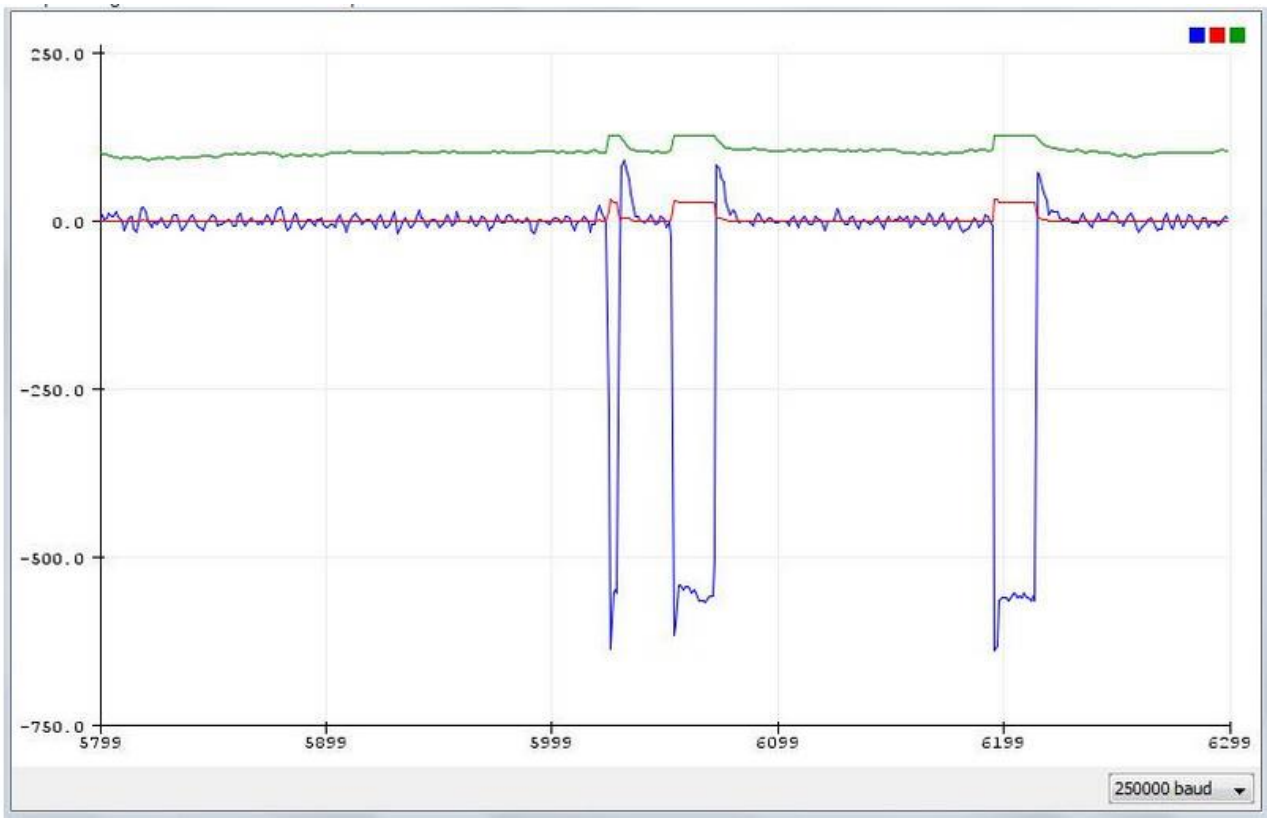
1er exemple : augmentation progressive de l'injection

- branchement de l'injecteur éolien à 5220
- Avant 5311 je débranche tout ce que je peux pour diminuer la consommation
- l'halogène commence à clignoter à 5394
- Il y a un pic à 5560, on voit dim évoluer en conséquence



2ème exemple : perturbations sur injection permanente

- L'injection est en régime permanent
- Il y a des pics de consommation : c'est la machine à pain….



Ajout d'une sortie de commande de délestage

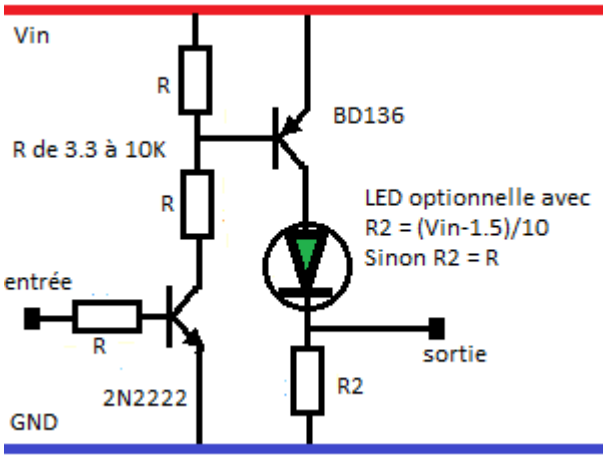
Pourquoi faire ?

A l'usage il peut s'avérer que le Power Router se déclenche trop souvent : il y a gâchis d'énergie. Dans ce cas il peut être utile de brancher un appareil en « tout ou rien » c'est-à-dire non pas de façon proportionnelle comme la commande de la charge, mais de façon permanente, pour une durée qui dépendra soit de la consommation qui devra croître, soit de la production qui devra chuter.
Ou bien au contraire il peut être souhaitable de débrancher un des injecteurs, ou bien de passer de l'injection à la charge de batteries. Bref nous allons ajouter un circuit de délestage.

Pour réaliser cette nouvelle fonctionnalité on utilise une sortie numérique supplémentaire, qui, lorsque la mesure de la puissance atteint un seuil – delestON - passe à l'état « haut », le délestage s'implémente, jusqu'à l'atteinte d'un seuil de consommation – delestOFF - , à l'issue duquel cette sortie numérique repasse à l'état « bas » - ou réciproquement.

La sortie numérique doit pouvoir commander un appareil, pour cela elle sera connectée à un SSR. Soit ce SSR se trouve dans le même boitier que l'Arduino, soit i se trouve à une dizaine de mètres comme c'est le cas pour mon installation. Dans ce cas il est nécessaire de réaliser un petit ampli dont le schéma est ci-contre :

Au repos les 2 transistors sont bloqués, la tension en sortie est nulle.



Le signal haut de la sortie 11 de l'Arduino - soit 5V - arrive à la base du transistor 2N2222 qui sature. La tension à la base du BD136 chute, celui-ci sature à son tour et la tension de sortie vaut Vin.

Pour le 2N2222 n'importe quel transistor NPN petits signaux fait l'affaire. Juste faire attention au brochage.

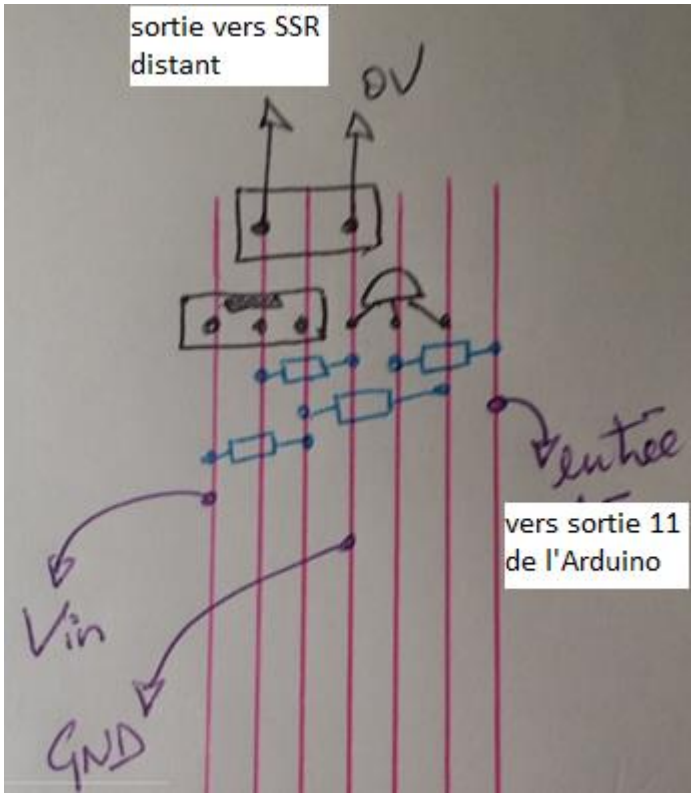
Idem pour le BD136, n'importe quel transistor PNP d'une puissance de dissipation de quelques watts fait l'affaire. En fait il est surdimensionné pour ne pas cramer en cas de court-circuit…

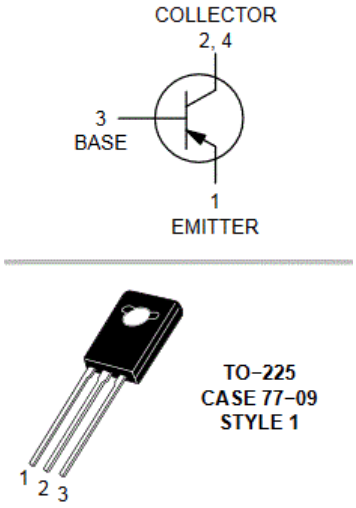
R vaut entre 3,3K et 10K

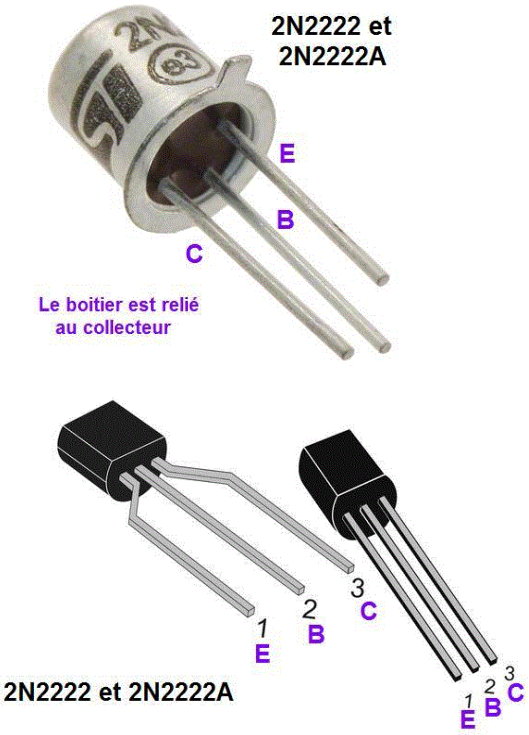
La base du 2N2222 est relié au port 11 de l'Arduino

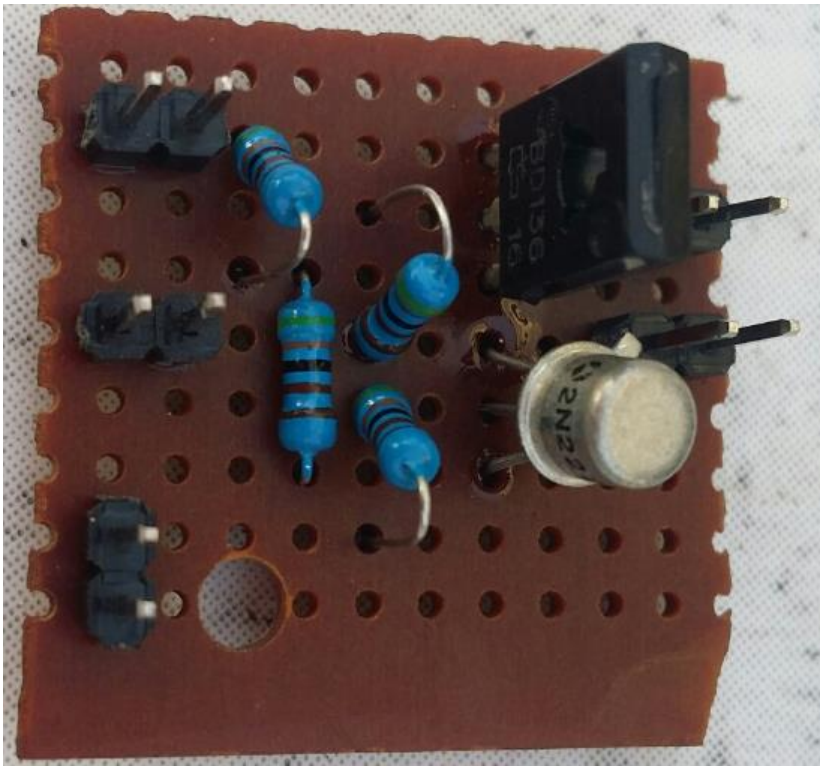
L'implantation des composants se fera sur la plaquette de montage, comme décrit avec le croquis ci-contre (vue côté composants).

Brochage du BD136 :

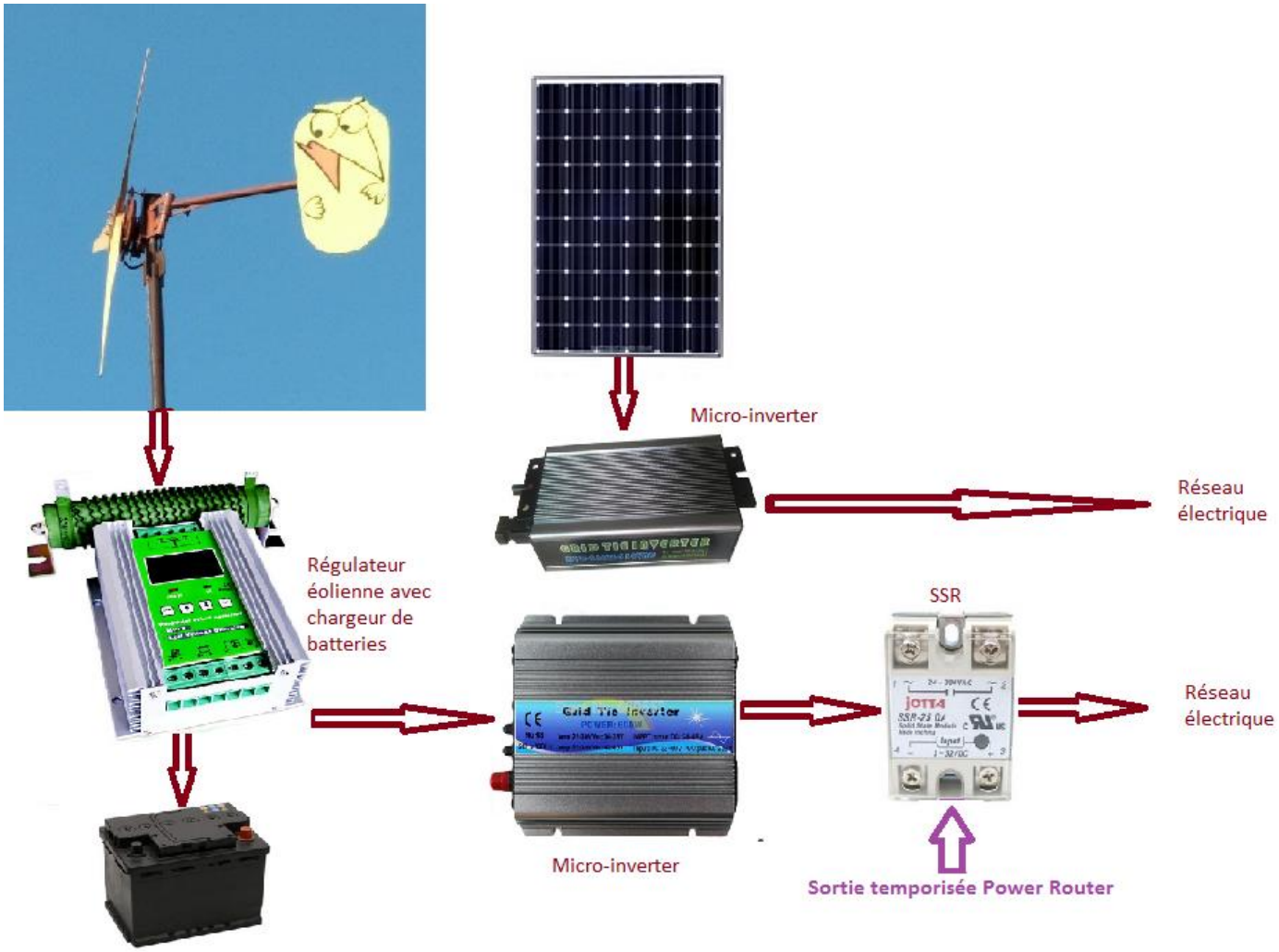








Mise en œuvre globale



Mise à jour du programme

Par rapport à la version précédente il a été en outre rajouté un watchdog avec comptage des redémarrages.

```
/*

Power_Router est un système qui permet d'utiliser l'excédent d'énergie autoproduit par
l'allumage d'un appareil résistif (facteur de puissance proche de 1) ce qui évite l'injection
au réseau public de distribution d'électricité.

Le principe de fonctionnement est le suivant :
- détection de phase entre courant et tension permet de savoir si on consomme ou bien on injecte
- en cas d'injection il se produit la mise en route progressive d'un dispositif d'absorption
d'excédent de puissance
- la mesure du courant permet d'ajuster au mieux le niveau d'absorption de cet excédent.
- Par ailleurs il est prévu une sortie temporisée de 1 minute lorsque le seuil d'injection est
proche de 20W permettant par exemple de couper l'injection d'une éolienne au profit de la charge
de batteries.

le programme prévoit :
- une sonde de tension : simple transfo 230V/5V Crête à Crête sur mi-tension (2.5V)
- une sonde de courant : 20A/25mA sur mi-tension (2.5V)
- un module de commande par triac
- un dispositif de détection de passage à zéro de la sinusoïde de tension secteur (par exemple
l'optocoupleur H11AA1)
- la bibliothèque TimeOne.h à installer et disponible là :
http://www.arduino.cc/playground/Code/Timer
- la bibliothèque EmonLib.h à installer et disponible là :
https://github.com/openenergymonitor/EmonLib/archive/master.zip

La gamme de puissances testée est va de 300 à 1000W

merci à Ryan McLaughlin <ryanjmclaughlin@gmail.com> pour avoir étudié et mis au point la partie
commande du triac
il y a quelques années et que j'ai repris dans ce programme :)
source : https://web.archive.org/web/20091212193047/http://www.arduino.cc:80/cgi-
bin/yabb2/YaBB.pl?num=1230333861/15
```

```
_____________________________________________________________
|                                                             |
|       auteur : Philippe de Craene <dcphilippe@yahoo.fr      |
|           pour l' Association P'TITWATT                     |
_____________________________________________________________

Toute contribution en vue de l'amélioration de l'appareil est la bienvenue ! Il vous est juste
demandé de conserver mon nom et mon email dans l'entête du programme, et bien sûr de partager
avec moi cette amélioration. Merci.

chronologie des versions :
version 0.5 - 3 mai 2018     - boucle de décrémentation dim --
version 0.8 - 5 juil. 2018   - 1ère version fonctionnelle, pb du pic de courant du triac
version 1   - 6 juil. 2018   - ajout de la bibliothèque EmonLib.h pour mesure du secteur
version 1.4 - 7 juil. 2018   - simplification des tests sur sPower et dim. Version film youtube
version 1.6 - 8 juil. 2018   - ajout LED d'overflow + optimisation des paramètres + seuilPoff
version 1.8 - 24 sept 2018   - ajout du pas variable sur dim avec dimstep
version 1.9 - 12 oct. 2018   - ajout d'une sortie temporisée de 5min à seuilPoff (25W) du seuil
d'injection
version 2.0 - 4 nov. 2018    - ajout d'un watchdog avec comptage de reset en EEPROM
version 2.2 - 7 nov. 2018    - seuilPtempo variable à part entière pour le délestage +
correction coquille
version 2.3 - 16 dec 2018    - réaménagement des messages console pour gagner du temps

*/

#include <EEPROM.h>
#include <avr/wdt.h>    // documentation : https://tushev.org/articles/arduino/5/arduino-and-
watchdog-timer
#include <TimerOne.h>   // librairie à installer depuis
http://www.arduino.cc/playground/Code/Timer1
#include <EmonLib.h>    // librairie à installer depuis
https://github.com/openenergymonitor/EmonLib/archive/master.zip

EnergyMonitor emon1;    // Create an instance

// détermination des entrées / sorties :

const byte triac_pin         = 10;    // sortie numérique de commande de charge résistive
const byte tempo_pin         = 11;    // sortie temporisée à 5min à 20W du seuil
const byte triacLED          = 3;     // sortie numérique pour la LED de test du triac
const byte limiteLED         = 13;    // sortie numérique pour la LED d'overflow
const byte voltageSensorPin  = 1;     // détecteur de tension entrée analogique 1
const byte currentSensorPin  = 0;     // détecteur de courant entrée analogique 0
const byte zeroCrossPin      = 2;     // détecteur de phase entrée digitale 2

// variables de gestion des interruptions (zero-crossing) :

int dimmax = 128;          // valeur max de dim pour inhiber le triac
int dim = dimmax;          // Dimming level (0-128)  0 = on, 128 = Off
char periodStep = 75;      // détermine la période du timer (65 pour 60Hz, 78 pour 50Hz, en µs)
                           // suivant la formule (500000/AC_freq)/NumSteps = periodStep
                           // 78*128=10ms=1/2 période 50Hz mais aux tests 76 marche mieux

volatile int i = 0;             // Variable to use as a counter
volatile boolean zero_cross=0;  // Boolean to store a "switch" for crossed zero

// variables de gestion de la mesure de puissance (seuls valeurI et rPower nous intéressent) :

long valeurI;                   // extract Irms into Variable (long supprime la « , »)
long rPower = 0;                // extract Real Power into variable
int dimstep;                    // valeur de l'écart de dim

// variables de calibration
                                // valeurs en mW (milliwatts) qui déterminent les seuils :
int seuilP = 3000;              // l'hystérésis d'asservissement : 3000 = 3W => hystérésis à 6W
int seuilPoff = 25000;          // seuil d'arrêt instantané en cas de chute de production
int valtempo = 30;              // temporisation du délestage temporisé : 30 secondes
int seuilPtempo = 3000;         // seuil de puissance pour démarrage du délestage
bool etat_tempo_repos = HIGH;   // état de la sortie temporisée au repos : HIGH ou LOW

// autres variables

unsigned long decompte;                 // décompte de la temporisation
unsigned int memo_temps = 0;
byte tempoON = LOW;

//
// SETUP
//_____________________________________________________________________________________________

void setup() {                  // Begin setup
 pinMode(triac_pin, OUTPUT);    // Set the Triac pin as output
 pinMode(tempo_pin, OUTPUT);
 pinMode(triacLED, OUTPUT);     // Set the LED pin as output
 pinMode(limiteLED, OUTPUT);
```

```
 attachInterrupt(digitalPinToInterrupt(zeroCrossPin), zero_cross_detect, RISING);
                            // en cas d'interruption lance 'zero_cross_detect' en mode RISING
                            // à chaque changement de LOW à HIGH de zeroCrossPin.
// documentation : https://www.arduino.cc/reference/en/language/functions/external-
interrupts/attachinterrupt/

 Timer1.initialize(periodStep);                  // initialisation de la librairie TimerOne
 Timer1.attachInterrupt(dim_check, periodStep);  // Use the TimerOne Library to attach an
                                      // interrupt to the function we use to check to see if
                                      // it is the right time to fire the triac.  This function
                                      // will now run every periodStep in microseconds.

 emon1.voltage(voltageSensorPin, 200, 1.7);     // Voltage: input pin, calibration, phase_shift
 emon1.current(currentSensorPin, 8000);         // Current: input pin, calibration

// comparer l'affichage de la console avec celui d'un wattmètre pour s'approcher de la réalité.
// Sinon faire le test avec une charge connue, par exemple une ampoule tungstène de 60W
// Les valeurs ne sont pas critiques, l'ordre de grandeur est 200 pour la tension et
// 8000 pour avoir le courant en mA (milliampères)

// Utilisation de EEPROM pour récupérer et mettre à jour le nombre de redémarrage du programme
// reconstruction du nombre redemarage, pour rappel EEPROM ne gère que le type unsigned char
 unsigned char redemarrage_high = EEPROM.read(0);  // récupère la 1ère moitié du nombre
 unsigned char redemarrage_low  = EEPROM.read(1);  // récupère la 2ème moitié du nombre
 unsigned int redemarrage = (redemarrage_high << 8) + redemarrage_low;
 redemarrage++;
 EEPROM.update(0, highByte(redemarrage));
 EEPROM.update(1, lowByte(redemarrage));

 Serial.begin(250000);
 Serial.println ();
 Serial.print("A NOTER : ");
 Serial.print(redemarrage);
 Serial.println(" ème redémarrage du programme");
 Serial.println();
 Serial.println("Prêt à démarrer ...");
 Serial.println ();
 delay(500);
 Serial.print(" I (A)| Pu (W) || dimstep |  dim ||  sortie temporisée");
 Serial.println();

 wdt_enable(WDTO_500MS);         // watchdog = reset après inaction suppérieure à 500ms

}                // End setup


//
// ZERO CROSS DETECT : gestion du passage à zéro par interruption
//_____________________________________________________________________________________

void zero_cross_detect() {       // function to be fired at the zero crossing
   zero_cross = 1;               // set the boolean to true to tell our dimming function
                                 // that a zero cross has occured
}                                // End zero_cross_detect


//
// DIM CHECK : gestion du triac par interruption
//_____________________________________________________________________________________

void dim_check() {                           // Function will fire the triac at the proper time
 if(zero_cross == 1 && dim < dimmax) {       // First check to make sure the zero-cross has
                                             // happened else do nothing
                                             // ET inutile de compter le fire ça doit être éteint
     if(i>dim) {             // i est un compteur qui détermine le retard au fire. plus dim
                             // est élevé, plus de temps prendra le compteur i et plus tard
       digitalWrite(triac_pin, HIGH);        // se fera le fire du triac
       delayMicroseconds(50);                // Pause briefly to ensure the triac turned on
       digitalWrite(triac_pin, LOW);         // Turn off the Triac gate, le triac reste conducteur
       i = 0;                                // Reset the accumilator
       zero_cross = 0;                       // Reset the zero_cross so it may be turned on again
at the next zero_cross_detect
       }
    else { i++; }            // If the dimming value has not been reached, incriment our counter
 }                           // End zero_cross check
}                            // End dim_check function


//
// LOOP : programme principal (qui tourne en boucle)
//_____________________________________________________________________________________

void loop() {                    // Main Loop

// Ci-dessous est le relevé des mesures du courant et de la puissance utile
// rPower indique une polarité et donc permet de savoir si on consomme ou si on injecte.
```

```
 valeurI      = emon1.Irms;              //extract Irms into Variable
 rPower       = emon1.realPower;         //extract Real Power into variable

 emon1.calcVI(20,200);         // Calculate all. No.of half wavelengths (crossings), time-out
                               // d'origine emon1.calcVI(20,2000)

 dimstep = abs(rPower/20000)+1;           // le pas dépend de la puissance en jeu

 if( rPower < -seuilPoff ) { dim = dimmax; }   // arrêt en cas de chutte brusque de production

 else {                        // test de polarité => puissance positive en mode injection
   if( rPower > seuilP ) {     // l'injection augmente, on diminue le délai d'allumage du triac
      if( dim > dimstep ) { dim -= dimstep; } else { dim = 0;}}
   else if( rPower < -seuilP ) {               // moins de prod : on baisse la charge
           if( dim + dimstep < dimmax ) { dim += dimstep; } else { dim = dimmax; }}
 }

  if(dim < 1) { digitalWrite(limiteLED, HIGH); }    // led témoin de surcharge
  else { digitalWrite(limiteLED, LOW); }
  analogWrite(triacLED, dimmax-dim);                 // Write the value to the LED for testing

// Sortie temporisée quand le seuil seuilPtempo est atteint
 unsigned int temps_actuel = millis()/1000;    // mesure du temps en secondes


 if( rPower > -seuilPtempo) {
   tempoON = HIGH;                             // détection seuil atteint
   decompte = temps_actuel;                    // initialisation du compteur
 }

 if( tempoON = HIGH ) {
   if( temps_actuel - decompte < valtempo) {
     digitalWrite(tempo_pin, !etat_tempo_repos);     // sortie temporisée active
   }
   else {
     digitalWrite(tempo_pin, etat_tempo_repos);
     tempoON = LOW;                             // fin de temporisation
   }
 }

// affichage de ce qui se passe toutes les 2 secondes car ça bouffe du temps....
  if( temps_actuel == memo_temps +2 ) {
    Serial.print("    ");
    Serial.print(valeurI/1000);
    Serial.print("  |  ");
    Serial.print(rPower/1000);
    Serial.print("  ||      ");
    Serial.print(dimstep);
    Serial.print("  |  ");
    Serial.print(dim);
    Serial.print(" ||  ");
    Serial.print(" état tempo : ");
    Serial.print(tempoON);
    Serial.print(" décompte : ");
    Serial.println(temps_actuel - decompte);
    memo_temps = temps_actuel;
  }

 wdt_reset();                  // raz de la tempo du watchdog

}                              // fin de Main Loop
```

Bilan à l'usage et améliorations apportées

Le montage précédent a fonctionné durant 6 mois et il joue son rôle. Néanmoins le temps de réaction n'est pas négligeable et doit pouvoir être optimisé.

En jouant sur la valeur du dénominateur « 2000 » de la ligne d'instruction :

```
dimstep = abs(rPower/20000)+1;            // le pas dépend de la puissance en jeu
```

on peut faire un pas plus ou moins dynamique en fonction de la puissance mesurée – plus la valeur est élevée (disons 50000) plus dimstep est amortie et donc plus il faudra de cycles du programme `loop()` pour que la valeur de consommation rejoigne celle de l'injection. Au contraire une valeur plus faible va donner des pas d'incrémentation du triac plus important, et donc réduire le nombre de cycles nécessaires, cependant au risque de faire entrer le montage en oscillation par surcompensation.

En cas de bug : ajout d'un « watchdog »

Le watchdog est un dispositif qui réinitialise le programme lorsqu'il ne tourne plus rond. En fait il s'agit d'un petit compteur, lorsque l'instruction `wdt_reset();` n'a pas été exécuté après un certain temps, l'Arduino se réinitialise.

Le certain temps est réglé dans le setup par l'instruction :

```
wdt_enable(WDTO_500MS);          // watchdog = reset après inaction supérieure à 500ms
```

Les curieux peuvent consulter la page de documentation : https://tushev.org/articles/arduino/5/arduino-and-watchdog-timer

Par ailleurs, si le watchdog tire l'alarme, il est bon de le savoir. Il a donc été ajouté dans le setup du programme une partie qui récupère une donnée en EEPROM, l'incrémente, et la remet en EEPROM. L'EEPROM est une mémoire non volatile, c'est-à-dire qui est perpétuelle même en l'absence de tension d'alimentation.

La donnée sauvegardée en EEPROM est de type «caractère », c'est-à-dire de 0 à 254, soit 1 byte. Pour aller plus loin en valeurs numériques nous en utiliserons 2, l'un de byte de poids fort d'un entier positif, l'autre de poids faible :

```
unsigned char redemarrage_high = EEPROM.read(0);  // récupère la 1ère moitié du nombre
unsigned char redemarrage_low  = EEPROM.read(1);  // récupère la 2ème moitié du nombre
unsigned int redemarrage = (redemarrage_high << 8) + redemarrage_low;
redemarrage++;
EEPROM.update(0, highByte(redemarrage));
EEPROM.update(1, lowByte(redemarrage));
```

L'usage du Watchdog et de l'EEPROM font appels à 2 librairies livrées en standard.

Supprimer les messages Console

Une solution simple qui améliore énormément l'efficacité du montage est de supprimer les messages envoyés à la Console. Ainsi l'ajout de la variable `VERBOSE = true` permet le debugging et l'affichage à la Console, en état normal d'utilisation on préfèrera `VERBOSE = false` qui supprime tout messages hors ceux au démarrage.

Ajouter un LCD 1602

Néanmoins comme on aime bien savoir un peu ce qui se passe au niveau du fonctionnement nous allons ajouter un petit afficheur LCD.

Nous allons utiliser l'afficheur le plus classique qui soit : le LCD 1602 pour 16 caractères sur 2 lignes, avec l'extension I2C qui réduit considérablement la complexité du montage qui se résume à 2 fils de data. Toutes les infos sont décrites là : https://andrologiciels.wordpress.com/arduino/lcd/lcd-1602-i2c/

La bibliothèque est ici : https://app.box.com/s/czde88f5b9vpulhf8z56

Attention ! L'installation de la bibliothèque met à disposition une nouvelle bibliothèque *LiquidChrystal.h* ; il faut donc absolument supprimer – ou renommer – tous les fichiers de même nom : *LiquidCrystal.h* et *LiquidCrystal.cpp*, qui se trouvent quelque part sous le répertoire *C:\Program Files (x86)\Arduino\librairies*. Heureusement il n'y a rien de destructifs, les nouvelles librairies apportant les fonctionnalités compatibles avec les LCD classiques.

Afin de ne pas ruiner le bénéfice de la rapidité, le programme prévoira une mise à jour de l'affichage toutes les 2 secondes.

Supprimer la librairie *EmonLib.h*

Pour celui qui débute en programmation, comme ça a été mon cas il y a quelques mois, une librairie est une sorte de « boite noire » qui permet de commander un périphérique – par exemple le LCD - , ou un organe de fonctionnement – par exemple les interruptions -. Et il est toujours très intéressant – mais pas toujours facile d'accès – d'analyser le contenu des librairies. En l'occurrence pour les 2 exemples précédemment cités la librairie ajoute en simplicité et en clarté du programme en résumant en quelques sortes des accès aux registres du processeur ou bien des séquences d'un protocole de communication en une bibliothèque de nouvelles instructions qui leur sont propres.

Concernant *EmonLib.h* il s'agit d'une « boite noire » qui nous sert à calculer la puissance réelle instantanée. Or il n'y a ni accès aux registres, ni protocole de communication, juste un algorithme de calcul. Le côté amusant pour la petite histoire est qu'il m'a été fait reproche sur un forum de ne pas citer avec grande emphase Energie Monitor qui commercialise des capteurs de mesures de puissance – que nous n'utilisons pas ici - et offre sa librairie pour leur usage – que nous n'allons bientôt plus utiliser. Or ce malotru aurait été bien plus malin d'essayer mon programme et de le critiquer quant à son fonctionnement, au lieu de me traiter de copieur pour l'usage d'une librairie qui contient trois calculs qu'on apprend à l'école... Bref cette librairie montre qu'il faut 200ms pour effectuer une mesure, ce qui est long, mesure précédée par une forme de détection de passage à zéro, ce qui est un doublon car nous avons déjà l'information par ailleurs.

Supprimer la consigne *seuilPoff*

Cette consigne seuilPoff assurait la fonction de couper immédiatement toute alimentation dans la charge lorsqu'il se produit une brusque baisse d'injection, à savoir quand la puissance instantanée chute en dessous de la consigne.

A l'usage cette fonctionnalité génère de l'instabilité alors qu'il est déjà possible d'arriver presque au même résultat en calibrant judicieusement le *dimstep* avec *coefdeReaction*.

Le programme final

```
/*

Power_Router est un système qui permet d'utiliser l'excédent d'énergie autoproduit par
l'allumage d'un appareil résistif (facteur de puissance proche de 1) ce qui évite l'injection
au réseau public de distribution d'électricité.

Le principe de fonctionnement est le suivant :
- détection de phase entre courant et tension permet de savoir si on consomme ou bien on injecte
- en cas d'injection il se produit la mise en route progressive d'un dispositif d'absorption
d'excédent de puissance
- la mesure du courant permet d'ajuster au mieux le niveau d'absorption de cet excédent.
```

- Par ailleurs il est prévu une sortie temporisée de 30 secondes (paramétrable) lorsque le seuil d'injection est proche de 3W (paramétrable) permettant par exemple de couper l'injection d'une éolienne au profit de la charge de batteries.

le programme prévoit :
- une sonde de tension : simple transfo 230V/5V Crête à Crête sur mi-tension (2.5V)
- une sonde de courant : 20A/25mA sur mi-tension (2.5V)
- un module de commande par triac
- un dispositif de détection de passage à zéro de la sinusoïde de tension secteur (par exemple l'optocoupleur H11AA1)
- la bibliothèque TimeOne.h à installer et disponible là :
http://www.arduino.cc/playground/Code/Timer
- en option un afficheur LCD 1602 avec extension I2C

La gamme de puissances testée est va de 300 à 1000W

merci à Ryan McLaughlin <ryanjmclaughlin@gmail.com> pour avoir étudié et mis au point la partie commande du triac il y a quelques années et que j'ai repris dans ce programme :)
source : https://web.archive.org/web/20091212193047/http://www.arduino.cc:80/cgi-bin/yabb2/YaBB.pl?num=1230333861/15

```
_____________________________________________________________
|                                                           |
|       auteur : Philippe de Craene <dcphilippe@yahoo.fr    |
|           pour l' Association P'TITWATT                   |
_____________________________________________________________
```

Toute contribution en vue de l'amélioration de l'appareil est la bienvenue ! Il vous est juste demandé de conserver mon nom et mon email dans l'entête du programme, et bien sûr de partager avec moi cette amélioration. Merci.

Récapitulatif des branchements :
 - broche A0 (analogique 0) => sonde de courant
 - broche A1 (analogique 1) => sonde de tension
 - broche A4 (analogique 4) => broche SDA sur LCD
 - broche A5 (analogique 5) => broche SCL sur LCD
 - broche 2 (numérique 2) => broche ZC (zero-cross) du module triac
 - broche 3 (numérique 3) => LED indiquant le taux de pwm
 - broche 10 (numérique 10) => broche PWM (commande Triac) du module triac
 - brcohe 11 (numérique 11) => circuit de commande du délestage
 - broche 13 (numérique 13) => LED indiquant l'overflow

Chronologie des versions :
version 0.5 - 3 mai 2018   - boucle de décrémentation dim --
version 0.8 - 5 juil. 2018   - 1ère version fonctionnelle, pb du pic de courant du triac
version 1   - 6 juil. 2018   - ajout de la bibliothèque EmonLib.h pour mesure du secteur
version 1.4 - 7 juil. 2018   - simplification des tests sur sPower et dim.
version 1.6 - 8 juil. 2018   - ajout LED d'overflow + optimisation des paramètres + seuilPoff
version 1.8 - 24 sept 2018   - ajout du pas variable sur dim avec dimstep
version 1.9 - 12 oct. 2018   - ajout d'une sortie temporisée de 5min à seuilPoff (25W) du seuil d'injection
version 2.0 - 4 nov. 2018   - ajout d'un watchdog avec comptage de reset en EEPROM
version 2.2 - 7 nov. 2018   - seuilPtempo variable à part entière pour le délestage + correction coquille
version 2.3 - 16 dec 2018   - réaménagemet des messages console pour gagner du temps
version 2.4 - 12 jan 2019   - ajout d'un afficheur LCD 1602 avec extension I2C
version 3.2 - 17 jan 2019   - gain en performances en contournant EmonLib.h
version 3.3 - 22 fev 2019   - abandon de seuilPoff : arrêt en cas de chutte brusque d'injection
version 3.4 - 27 avr 2019   - changement du délestage par les seuils delestON et delestOFF
*/

```
#include <EEPROM.h>
#include <avr/wdt.h>    // documentation : https://tushev.org/articles/arduino/5/arduino-and-watchdog-timer
#include <TimerOne.h>   // librairie à installer depuis http://www.arduino.cc/playground/Code/Timer1
#include <LiquidCrystal_I2C.h> // librairie à installer depuis https://app.box.com/s/czde88f5b9vpulhf8z56
// Attention ! Renomer la librairie d'origine de l'IDE LiquidCrystal.h et LiquidCrystal.cpp

// variables de calibration :

bool CALIBRATION = false; // pour définir les valeurs de Vcalibration et Icalibration
bool VERBOSE = false;     // pour le debugging VERBOSE = true affiche les mesures à la console
                          // cependant au détriment des performances

// Calibration des mesures des sondes pour avoir des valeurs correctes :
// en premier : Vcalibration doit être défini pour obtenir les 230V du secteur
// ensuite : Icalibration en comparant l'affichage de la console avec celui d'un wattmètre
// pour s'approcher de la réalité. En l'absence de wattmètre effectuer le test avec une
// charge résistive connue, par exemple une ampoule halogène neuve.
// REMARQUE : les variables de type float exigent un nombre à virgule

float Vcalibration     = 0.97;   // pour obtenir 230V
float Icalibration     = 40.6;   // le courant est mesuré en mA (milliampères)
float phasecalibration = 1.7;    // valeur empirique compensant le déphasage lié aux capteurs...
```

```
byte totalCount        = 20;     // nombre de demie-périodes étudiées pour la mesure

// Valeurs en mW (milliwatts) qui déterminent les seuils :

int seuilP     = 1000;           // l'hystérésis d'asservissement : 3000 = 3W => hystérésis à 6W
long delestON  = 1000;           // seuil de puissance pour démarrage du délestage
long delestOFF = 350000;         // seuil d'arrêt du délestage
bool etat_delest_repos  = HIGH;  // état de la sortie temporisée au repos : HIGH pour actif

// Valeur du coefficient de réaction de l'asservissement avec le dimensionnement de dimstep :
// Incrémente dimstep par pas issue du rapport entre la puissance à dissiper et coefdeReaction
// compromis entre rapidité de réaction à l'injection et instabilité :
// trop petit = risque l'oscillation, trop grand = plus lent
// détermination de la valeur : coefdeReaction ~ (puissance de la charge)/4 en Watts

unsigned int coefdeReaction  = 90;

// détermination des entrées / sorties :

const byte triac_pin         = 10;    // sortie numérique de commande de charge résistive
const byte delest_pin        = 11;    // sortie pour délestage
const byte triacLED          = 3;     // sortie numérique pour la LED de test du triac
const byte limiteLED         = 13;    // sortie numérique pour la LED d'overflow
const byte voltageSensorPin  = 1;     // détecteur de tension entrée analogique 1
const byte currentSensorPin  = 0;     // détecteur de courant entrée analogique 0
const byte zeroCrossPin      = 2;     // détecteur de phase entrée digitale 2

// variables de gestion des interruptions (zero-crossing) :

byte dimmax = 128;               // valeur max de dim pour inhiber le triac
byte dim = dimmax;               // Dimming level (0-128)  0 = on, 128 = Off
char periodStep = 75;            // détermine la période du timer (65 pour 60Hz, 78 pour 50Hz, en
µs)
                                 // suivant la formule (500000/AC_freq)/NumSteps = periodStep
                                 // 78*128=10ms=1/2 période 50Hz mais aux tests 76 marche mieux
volatile int i = 0;              // Variable to use as a counter
volatile bool zero_cross = false;       // indicateur de zero-cross pour commander le triac
volatile bool zero_cross_flag = false;  // indicateur de zero-cross pour calcul des puissances
électriques

// variables de calcul des grandeurs électriques :

int lectureV, memo_lectureV, lectureI;   // tensions et courants en bits (0 à 1023 bits)
float rPower, V, I, sqV, sumV = 0, sqI, sumI = 0, instP, sumP = 0;
byte zero_crossCount = 0;        // compteur de demie-périodes

// autres variables :

int dimstep;                     // valeur de l'écart de dim
unsigned long decompte;          // décompte de la temporisation du délestage
unsigned int memo_temps = 0;
bool delestage = false;          // indicateur de délestage
bool unefois = false;            // marqueurs pour raz décopte
bool etat_delest_actif = !etat_delest_repos;

// Déclaration du LCD en mode I2C :

// toutes les infos ici : http://arduino-info.wikispaces.com/LCD-Blue-I2C
// Set the pins on the I2C chip used for LCD connections:
//                    addr, en,rw,rs,d4,d5,d6,d7,bl,blpol
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
// => connexion des 2 broches I2C sur l'Arduino Uno R3 : SDA en A4, SCL en A5


//
// SETUP
//_____________________________________________________________________________________________

void setup() {                   // Begin setup
 pinMode(triac_pin, OUTPUT);     // Set the Triac pin as output
 pinMode(delest_pin, OUTPUT);
 pinMode(triacLED,  OUTPUT);     // Set the LED pin as output
 pinMode(limiteLED, OUTPUT);

 attachInterrupt(digitalPinToInterrupt(zeroCrossPin), zero_cross_detect, RISING);
// à chaque changement d'état de zeroCrossPin la fonction 'zero_cross_detect' est exécutée
// quelque soit l'endroit où se trouvait l'exécution du programme : définition d'une
interruption
// documentation : https://www.arduino.cc/reference/en/language/functions/external-
interrupts/attachinterrupt/

 Timer1.initialize(periodStep);                    // initialisation de la librairie TimerOne
 Timer1.attachInterrupt(dim_check, periodStep);
// autre mode d'interruption avec la librairie TimerOne.h : à chaque periodStep écoulé dont la
// valeur est définie en microsecondes la fonction 'dim_check' est lancée : cette interruption
// nous assure d'actionner le triac au bon moment
```

```
// Utilisation de EEPROM pour récupérer et mettre à jour le nombre de redémarrage du programme
// reconstruction du nombre redemarage, pour rappel EEPROM ne gère que le type unsigned char
 unsigned char redemarrage_high = EEPROM.read(0);  // récupère la 1ère moitié du nombre
 unsigned char redemarrage_low  = EEPROM.read(1);  // récupère la 2ème moitié du nombre
 unsigned int redemarrage = (redemarrage_high << 8) + redemarrage_low;
 redemarrage++;
 EEPROM.update(0, highByte(redemarrage));
 EEPROM.update(1, lowByte(redemarrage));

// initialisation du LCD
 lcd.begin(16,2);                // initialize the lcd for 16 chars 2 lines
 lcd.clear();
 lcd.setCursor(0, 0);
 lcd.print("PTIWATT BONJOUR");
 lcd.setCursor(0, 1);
 lcd.print("   **********");

// initialisation de la console
 Serial.begin(250000);
 Serial.println ();
 Serial.print("A NOTER : ");
 Serial.print(redemarrage);
 Serial.println(" ème redémarrage du programme");
 Serial.println();
 Serial.println("Prêt à démarrer ...");
 Serial.println ();
 delay(500);
 if( VERBOSE == true ) Serial.print("  Pu (W) || dimstep |  dim ||  sortie temporisée");
 else Serial.println("C'est parti !");
 Serial.println();

 digitalWrite(delest_pin, etat_delest_repos);    // sortie délestage en mode par défaut

 wdt_enable(WDTO_500MS);         // watchdog = reset après inaction supérieure à 500ms

}                // End setup


//
// ZERO CROSS DETECT : gestion du passage à zéro par interruption
//_____________________________________________________________________________________________

void zero_cross_detect() {   // fonction appelée à chaque passage à zéro de la tension secteur
   zero_cross_flag = true;   // témoin pour démarrer le calcul de la puissance
   zero_cross = true;        // témoin pour la commande du triac
}


//
// DIM CHECK : gestion du triac par interruption
//_____________________________________________________________________________________________

void dim_check() {                        // Function will fire the triac at the proper time
 if(zero_cross == true && dim < dimmax)   // First check to make sure the zero-cross has
 {                                        // happened else do nothing
     if(i>dim) {           // i est un compteur qui détermine le retard au fire. plus dim
                           // est élevé, plus de temps prendra le compteur i et plus tard
       digitalWrite(triac_pin, HIGH);     // se fera le fire du triac
       delayMicroseconds(50);             // Pause briefly to ensure the triac turned on
       digitalWrite(triac_pin, LOW);      // Turn off the Triac gate, le triac reste conducteur
       i = 0;                             // Reset the accumilator
       zero_cross = false;
     }
    else { i++; }          // If the dimming value has not been reached, incriment our counter
 }                         // End zero_cross check
}                          // End dim_check function


//
// LOOP : programme principal (qui tourne en boucle)
//_____________________________________________________________________________________________

void loop() {                  // Main Loop

// 1ère partie : calcul de la puissance réelle relevée par les capteurs avec rPower
//_____________________________________________________________________________________________
//

  unsigned int numberOfSamples = 0;
  sumV = 0;
  sumI = 0;
  sumP = 0;

// à chaque passage à zéro de la tension du secteur réinitialisation périodique du compteur de
// passage à zéro zero_crossCount lorque le nombre de cycles de mesures totalCount est atteint

 if( zero_crossCount >= totalCount ) { zero_crossCount = 0; }
```

```
// processus de relevé de mesures durant le nombre défini de demi-périodes
  while( zero_crossCount < totalCount ) {
    if( zero_cross_flag == true ) {        // incrémentation du compteur de demi-périodes
secteur
      zero_cross_flag = false;
      zero_crossCount++;
    }
    numberOfSamples++;                          // comptage du nombre d'itérations
    memo_lectureV = lectureV;                   // mémorisation du relevé précédent
    lectureV = analogRead(voltageSensorPin);    // mesure de V en bits - 0V = bit 512
    lectureI = analogRead(currentSensorPin);    // mesure de I en bits - 0A = bit 512

// calcul des valeurs efficaces de tension et courant
    if( CALIBRATION == true ) {                    // ne sert que pour définir la calibration de
V et I
      sqV= (lectureV -512.0) * (lectureV -512.0);       // -512 comme offset pour ramener 0V à
bit 0
      sumV += sqV;
      sqI = (lectureI -512.0) * (lectureI -512.0);
           sumI += sqI;
    }    // fin de test sur VERBOSE
// calcul de la puissance instantanée
    instP = ((memo_lectureV -512.0) + phasecalibration * ((lectureV -512.0) - (memo_lectureV -
512))) * (lectureI -512.0);
    sumP +=instP;
 }      // fin de while sur zero_crossCount

// mémorisation des valeurs électriques
  if( numberOfSamples > 0 ) {
    if( CALIBRATION == true ) {
      V = Vcalibration * sqrt(sumV / numberOfSamples);
      I = Icalibration * sqrt(sumI / numberOfSamples);
    }
    rPower = Vcalibration * Icalibration * sumP / numberOfSamples;
  }

// 2ème partie : calcul du taux de puissance à délester pour ne pas injecter avec dim et dimstep
//               + détection si nécessité de délestage avec etat_delest
//_____________________________________________________________________________________________
//

// calcul de dimstep : le pas d'incrémentation de dim qui dépend de la puissance en jeu
  if( rPower > 0 ) { dimstep = (rPower/1000)/coefdeReaction + 1; }
  else { dimstep = 1 - (rPower/1000)/coefdeReaction; }

  if( rPower > seuilP ) {      // l'injection augmente, on diminue le délai d'allumage du triac
    if( dim > dimstep )  dim -= dimstep; else  dim = 0;
  }
  else if( rPower < -seuilP ) {                    // moins de prod : on baisse la charge
    if( dim + dimstep < dimmax ) dim += dimstep;  else  dim = dimmax;
  }

  if(dim < 1) { digitalWrite(limiteLED, HIGH); }  // led témoin de surcharge
  else { digitalWrite(limiteLED, LOW); }
  analogWrite(triacLED, dimmax-dim);              // Write the value to the LED for testing

// Sortie de délestage quand le seuil delestON est atteint

  unsigned int temps_actuel = millis()/1000;      // mesure du temps en secondes

  if( rPower > -delestON) { delestage = true; }   // détection seuil atteint

  if( delestage == true ) {
    if( unefois == false ) {
      digitalWrite(delest_pin, etat_delest_actif);     // maj sortie délestage
      decompte = temps_actuel;                    // initialisation du compteur
      unefois = true;
    }
    if( rPower < -delestOFF ) {                   // si le conso dépasse delestOFF
      digitalWrite(delest_pin, etat_delest_repos);    // maj sortie délestage
      unefois = false;
      delestage = false;
    }
  }                                               // fin test de délestage

// affichage de ce qui se passe toutes les 2 secondes car ça bouffe du temps....

  if( temps_actuel >= memo_temps +2 ) {
    memo_temps = temps_actuel;
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("P= ");
    lcd.print(String(-rPower/1000,0));
    lcd.print("w");
    lcd.setCursor(10, 0);
    lcd.print("T= ");
```

```
    lcd.print( map(dim, 0, dimmax, 99, 0) );
    lcd.print("%");
    lcd.setCursor(0, 1);
    lcd.print("DELESTAGE ");
    if( delestage == true ) {
      lcd.print(temps_actuel - decompte);
      lcd.print("s    ");
    }
    else { lcd.print("ARRETE"); }
  }  // fin tempo de 2 secondes

  if( CALIBRATION == true ) {
        Serial.print(V);
        Serial.print("  |  ");
    Serial.print(I/1000);
    Serial.print("  |  ");
    Serial.print(rPower/1000);
    Serial.println();
  }
  if( VERBOSE == true ) {
    Serial.print(rPower/1000);
    Serial.print("  ||     ");
    Serial.print(dimstep);
    Serial.print("  |  ");
    Serial.print(dim);
    Serial.print(" ||  ");
    Serial.print(" état délestage : ");
    Serial.print(delestage);
    Serial.print(" décompte : ");
    Serial.println(temps_actuel - decompte);
  }
  else { delay(1); }           // obligatoire pour la stabilité

 wdt_reset();                  // raz de la tempo du watchdog

}                               // fin de Main Loop
```

Mise en route

A la mise en place du montage, raccordé au PC via le câble USB, et avec l'aide d'un simple wattmètre du commerce, on effectuera les réglages suivants :

VERBOSE = true ;

1- Vcalibration à déterminer pour obtenir la tension secteur (environ 230V)
2- Icalibration à déterminer pour obtenir la même valeur de courant que le wattmètre

Revenir à VERBOSE = false ; les réglages sont terminés.

Remarque : aux premiers essais le gain en vitesse d'exécution est spectaculaire, on passe d'environ 3 cycles de loop par secondes à plus d'une quinzaine. Cependant la vitesse a un prix : pour ceux qui utilisent comme moi une ampoule halogène comme charge, il faut savoir que la résistance ohmique d'une ampoule n'est pas constante, elle diminue avec la tension, et il lui faut du temps pour varier. Et pour le coup le montage surcompense et entre en oscillation. La solution adoptée pour éviter ce phénomène est le `delay(1);` - 1 milliseconde - lorsqu'il n'y a pas la console pour ralentir l'asservissement.


P= 782w T= 0%
DELESTAGE ARRETE
ROUTER ON
PTIWATT
OVERFLOW