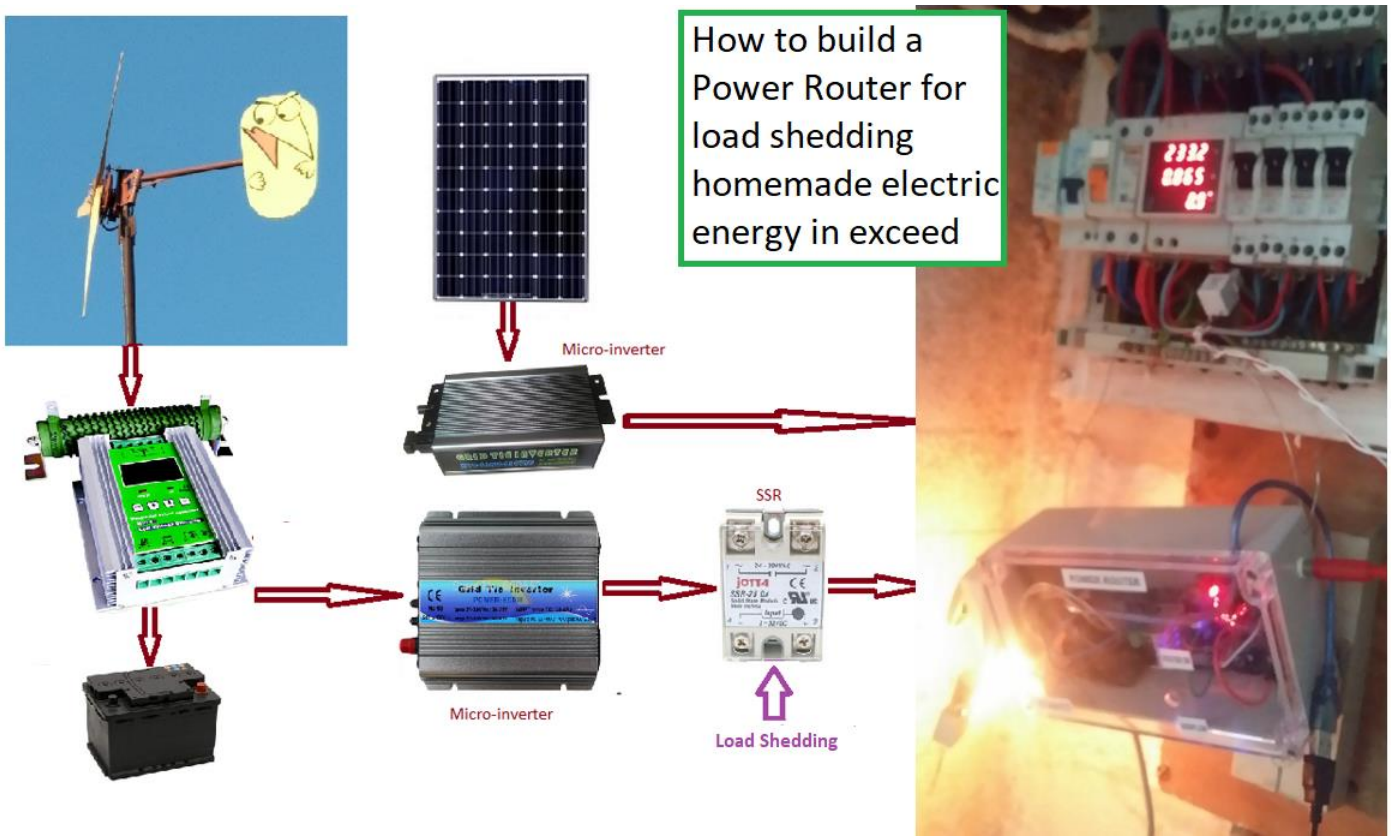
How to make a Power Router to optimize homemade electricity consumption



Author : Philippe de Craene dcphilippe@yahoo.fr

July 2018 – July 2019

Manual version: 1.0

Program version: 3.6

Are your tired to offer to your electricity grid provider any surplus of your own electricity you produce?

Would you prefer either to switch on a machine to consume this surplus, or to switch off a (micro) invertor so that exceeded energy will only charge batteries?

And so this would be done automatically?

A Power Router is a device that will detect any injection in public power grid and help to prevent against it.

This manual explains step by step how to build this device, with an Arduino Uno rev.3.

Table of content

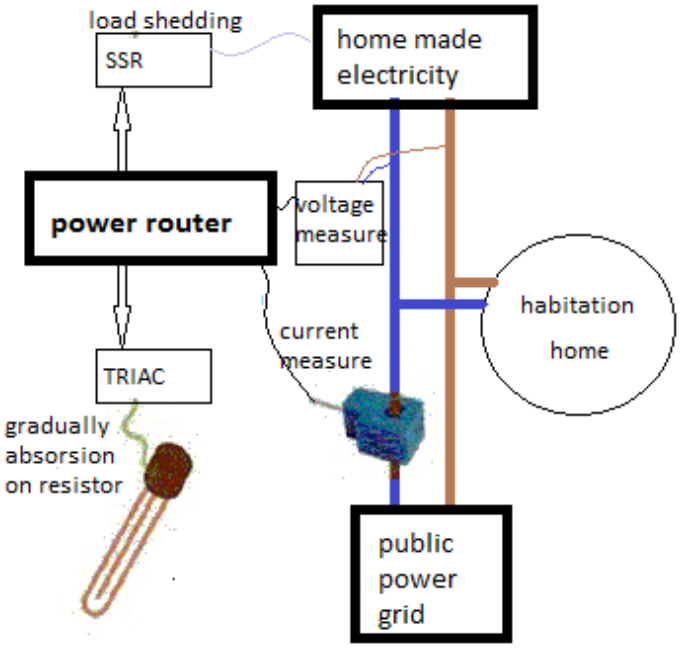
Introduction

The device detects the current at the immediate entrance of the public power grid, and the voltage on the line.

Any injection in the public power grid is detected by the shift between current and voltage:

When “in same way” we are consuming. When “in opposite way” we are injecting.

Then the power consumption is calculated and the Triac is driven by the device in order to prevent any injection. Parameters allow to fix limit values.



In addition a load shedding is possible that will switch on or off any device, depending of the consumption. For instance a SSR can cut off the wind turbine injection (and then it will charge batteries only) if consumption is near 0 (near injection in grid) and will be switched on again once consumption raises the power delivered by the wind turbine invertor.

List of materials

1- One Arduino Uno R3

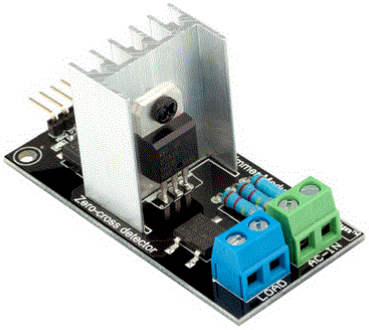
2- One prototype shield

3- One LCD 1602 with I2C extension

4- One current sensor: the operating range must be as sensitive as possible. The current sensor is a kind of mini coil transformer, the primary is the line wire, and the coil is the secondary. The secondary must be loaded by a Burden resistor, usually 100R. Note it is not use to increase this resistor, otherwise the level of saturation will increase. Depending of the current sensor model, either the Burden resistor is included in the component, either not, in this case only you will have to add it (the one in photo does not have it).

5- One voltage sensor: it is a small AC-AC voltage transformer of around 4V peak to peak, so about 2.8Veff. If higher voltage is chosen, the ratio of the voltage divider will be adapted.

6- One Triac driver module with zero cross voltage detector, for instance this one: 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
This module is a kind of all in one so it is very useful. It is a done for 5A maxi, 2A nominal, however the Triac is made to accept 16A so you just have to consolidate the circuit board tracks to allow more current.



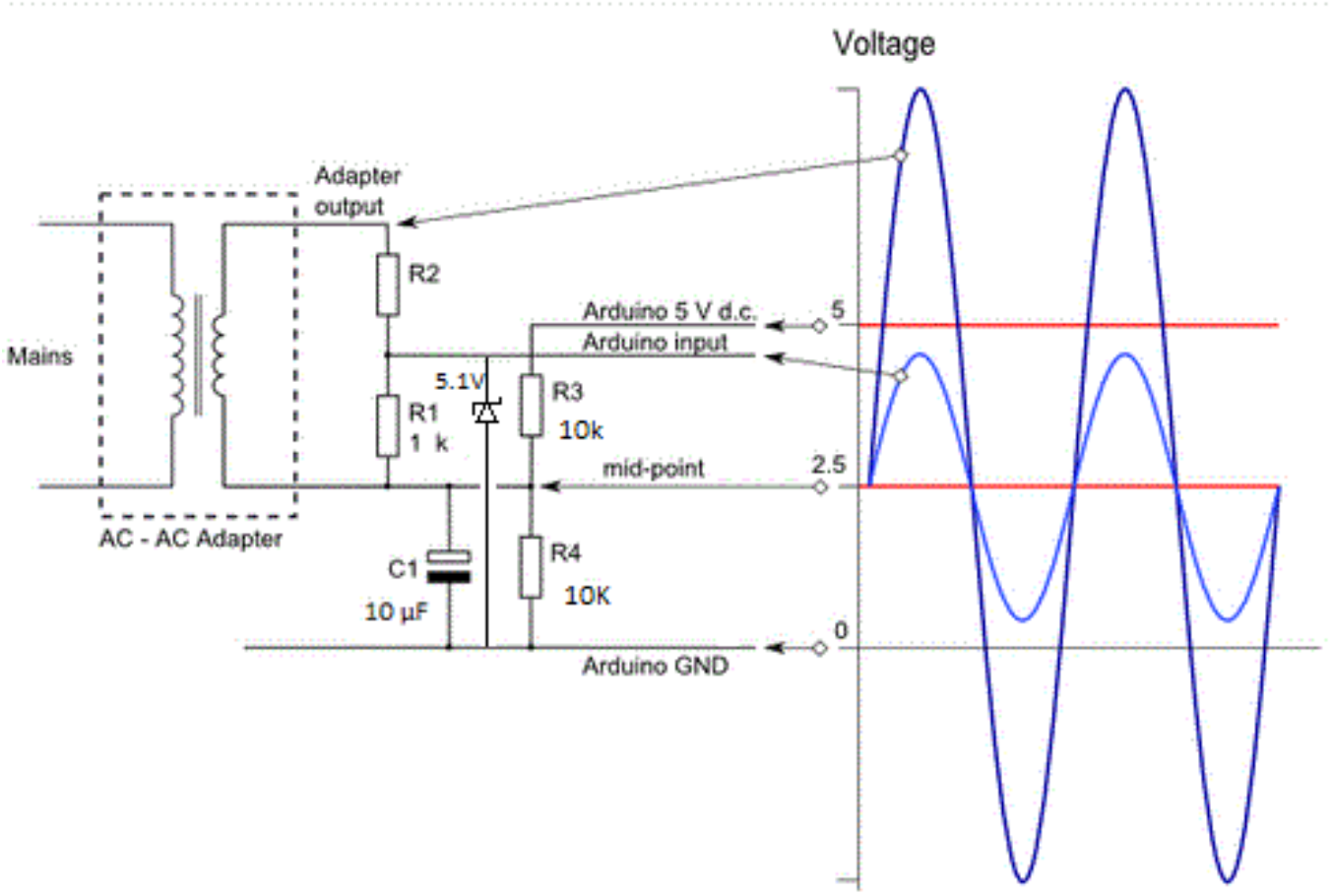
7- One power supply for the Arduino from 7V to 9V.

8- Some resistors, capacitors, 5.1V zener diodes, 3 push-buttons, etc. see text below.

Diagram circuit around sensors and accessories

Both sensors provide a sinus signal voltage that must be adapted to Arduino, which means they never exceed 5V. The 0V of sensors is fixed to mid-voltage of the Arduino: 2.5V. The amplitude of the sinus signal delivered by the sensors will oscillate around this mi-voltage, and must not exceed the range of 0-5V.

The measure of the AC voltage

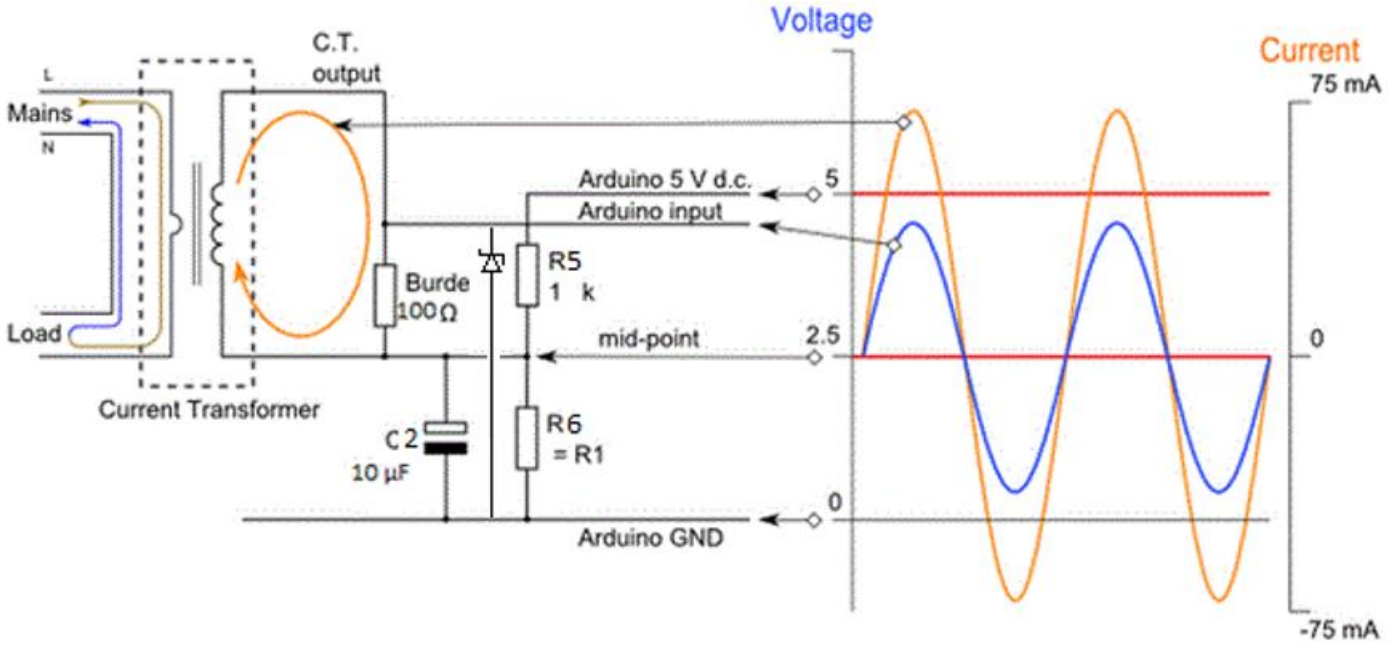


R2 is fixed by the rule R2 = Vtransfo eff *0.7 -1.
Verify twice the real voltage delivered by the AC-AC transformer (the multimeter will display the Veff), also the voltage issued at R1 must always stay between 0V and 5V.

R3 = R4, never mind of the value between 1K and 47K. C1 between 1uF and 47uF.
A zener of 5.1V make sure the protection against any surge.

The output of this diagram is connected to the input A1 of the Arduino.

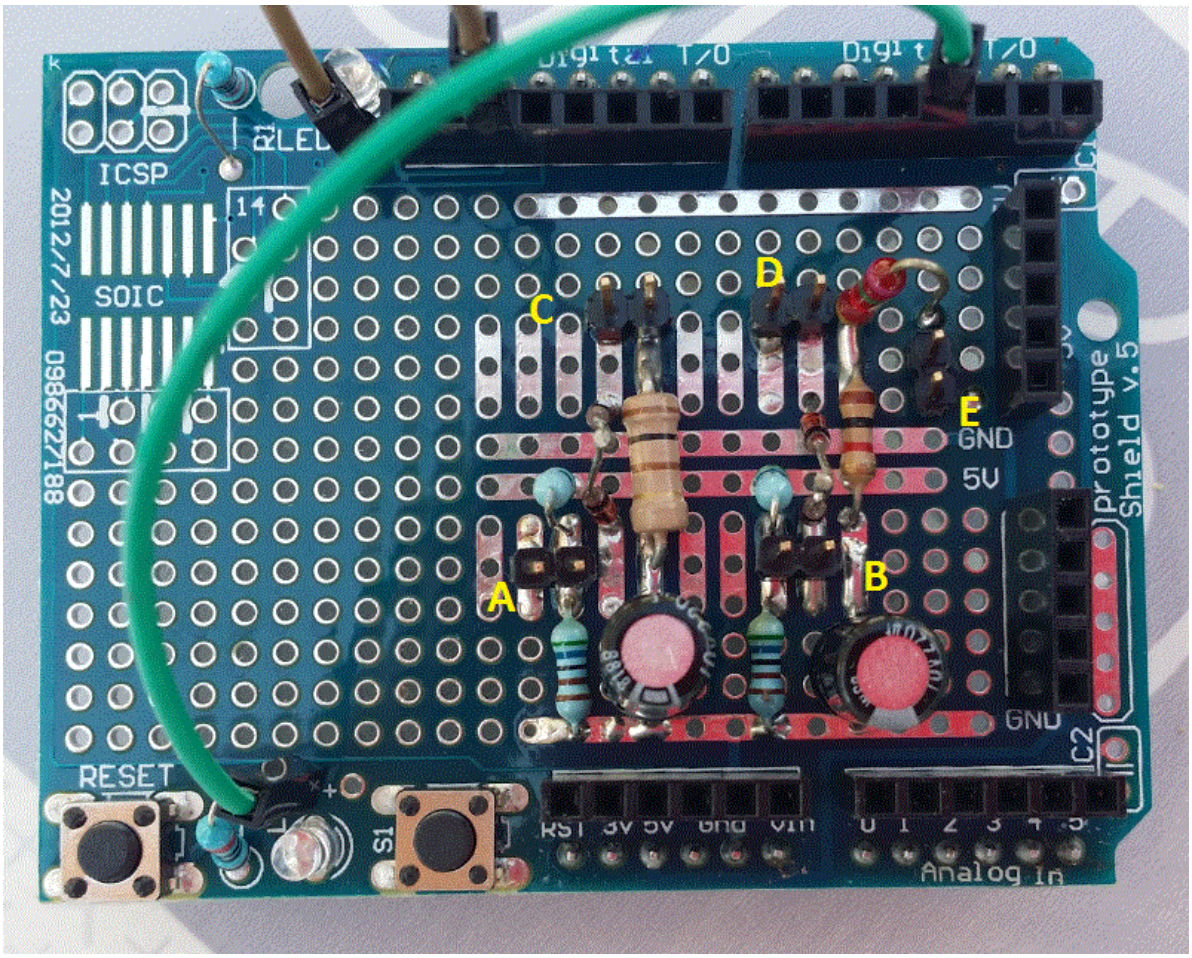
The measure of the AC current

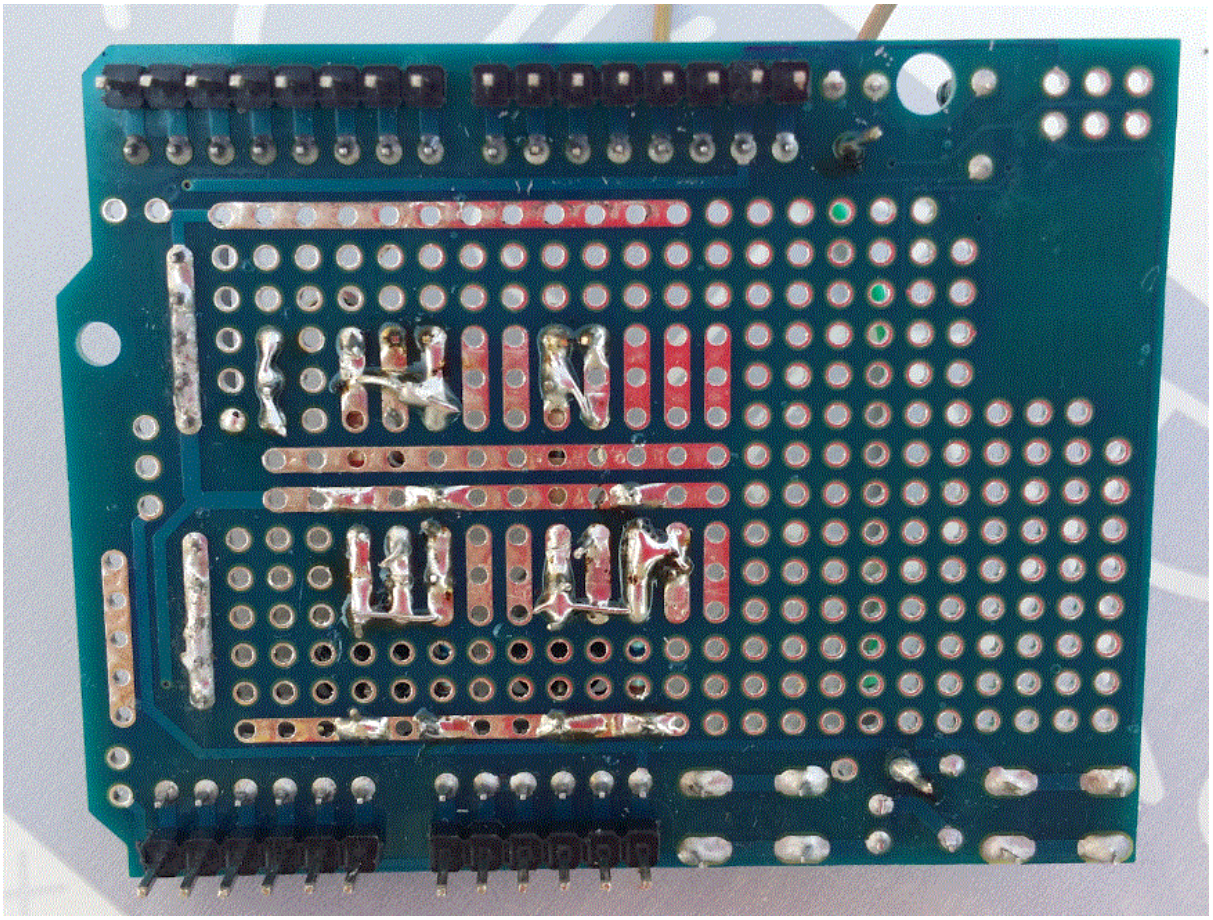


R5 = R6, never mind of the value between 1K and 47K. C2 between 1uF and 47uF.
Once again a zener of 5.1V make sure the protection against any surge.

The output of this diagram is connected to the input A0 of the Arduino.

Here are two pictures of the shield:





The Triac module

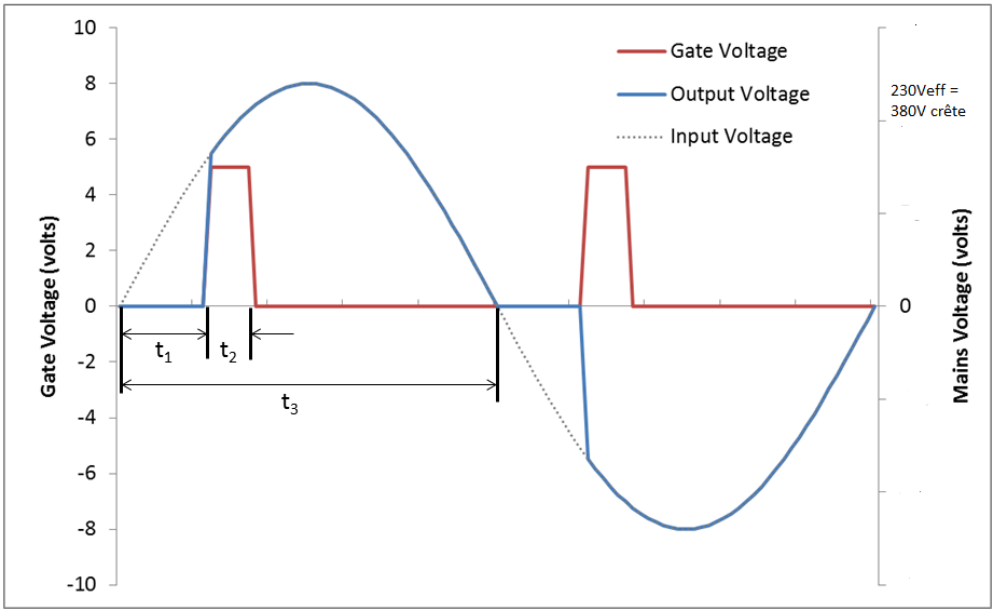
The purpose of the "game" is to open more or less a switch to spill in a resistive load the exceeded energy produced.

The advantage of the Triac is to cut off automatically each time the AC voltage is at 0V.

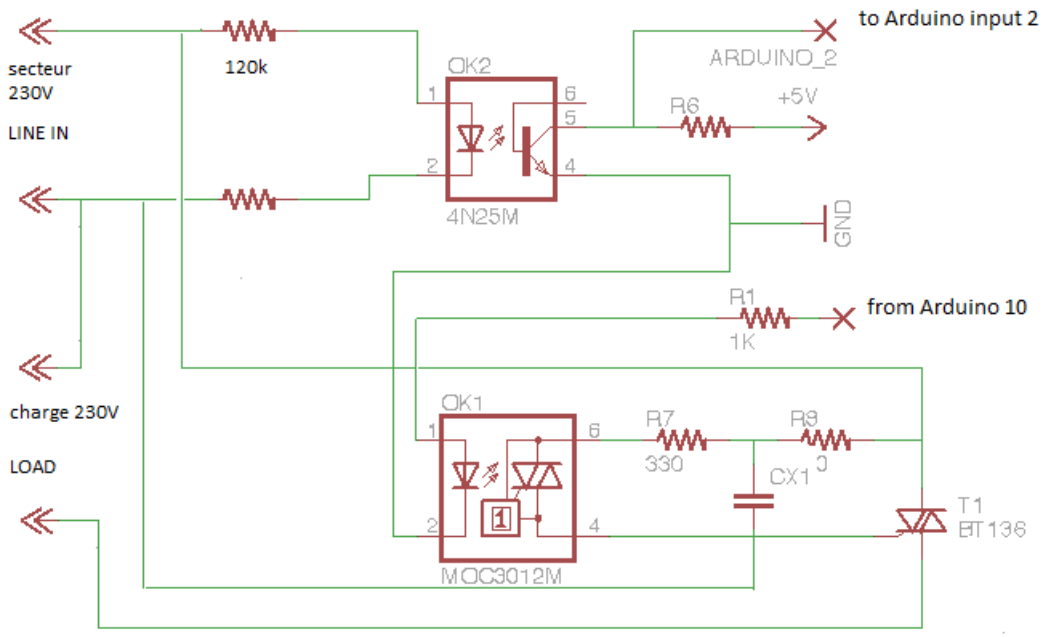
The Triac can be fired between two zero cross: Sooner it is fire, longer it is be switched on.

By this way the power available with the use of the Triac can be modulated depending of the time it is fired (t1 on the diagram).

So a zero cross detection is required. Hopefully the module contains a zero cross detector.

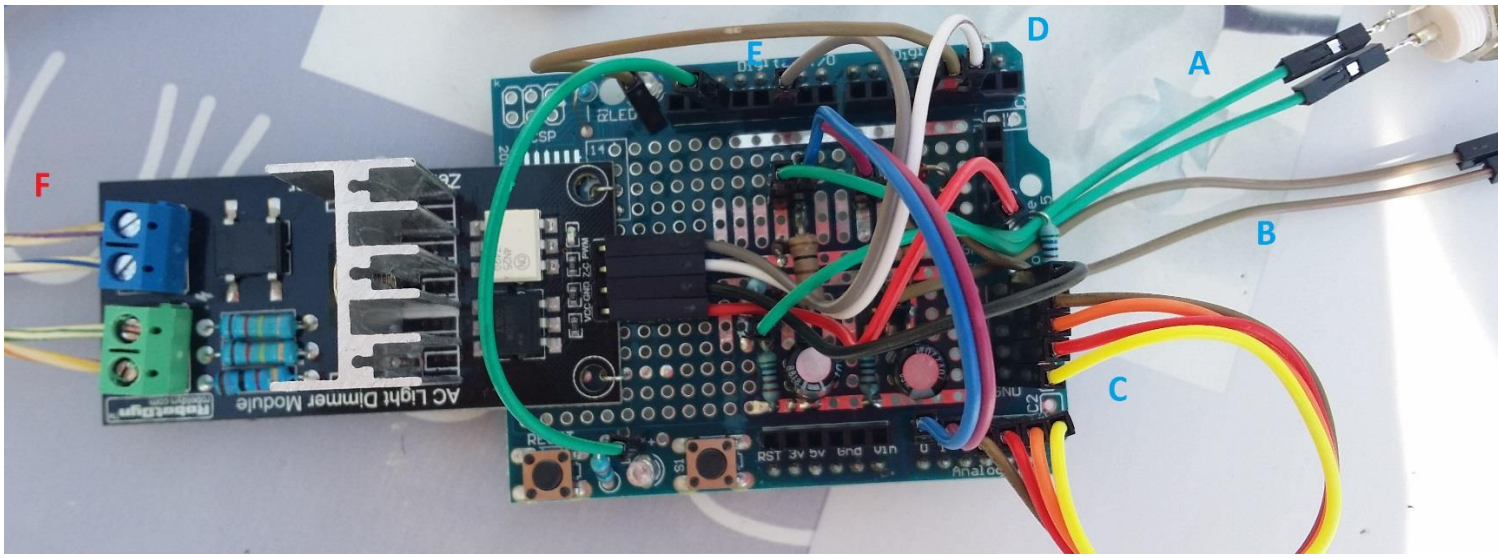


Below is the electric diagram of the module. The grid isolation is performed by opto-couplers:



Please note there are two models of zero cross detectors: those with two LEDs in the opto-coupler that will provide all zero-cross, and those with only one LED that will only detect positive zero-cross. In this case a AC-DC full rectifier (a 4 diodes rectifier) is needed in order to detect all zero-cross.

Here is a picture of the shield:

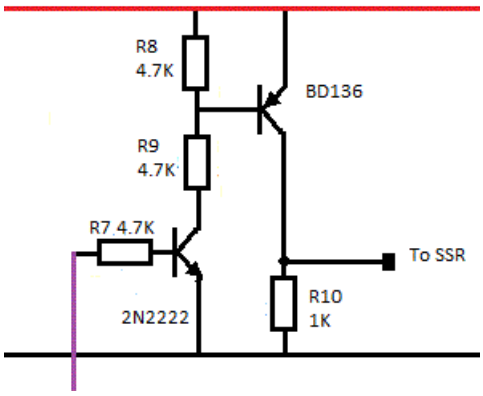


Captions:

- A: to the AC current transformer
- B : to the AC voltage transformer
- C : unused analog inputs are connected to GND.
- D : white wire is the zero-cross detection to Arduino input 2.
- E : brown wire is the pwm Triac gate driver from the Arduino output 10.
- F : Green terminal = 230V grid input, blue terminal = load resistor.

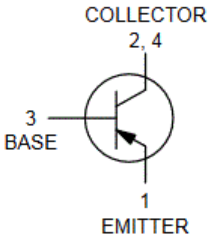
The load shedding – driving the SSR

The job of the Triac that gradually switch the resistive load is only to waste energy in exceed. So this should happen very sporadically only. In case of more constant energy in exceed it is better to take advantage with a load shedding which either: switch on any energy intensive device. Or switch off an energy producer.
The driver gives a logical signal that simply switch on or off a SSR, which can be considered as an electronic relay.

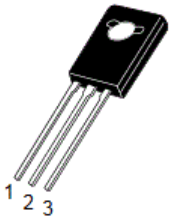


However may be the output of the Arduino will not be able to pilot a SSR directly. May be the SSR will take place far from the Power Router. For those reasons a small amplifier is built with 2 bipolar transistors according to the diagram in the left. The power supply is Vin (9V to 12V). R7 is connected to the Arduino output 11.If no input voltage there, both transistors are blocked, the output is set to 0V. On the contrary if the input gets a positive voltage above 1V both transistors are saturated, the output voltage is set to Vin.

The 2N2222 can be replaced by any NPN bipolar transistor, also the BD136 by any PNP; the advantage of such a PNP transistor is that it will not care of any (reasonable) surge or short-circuit.



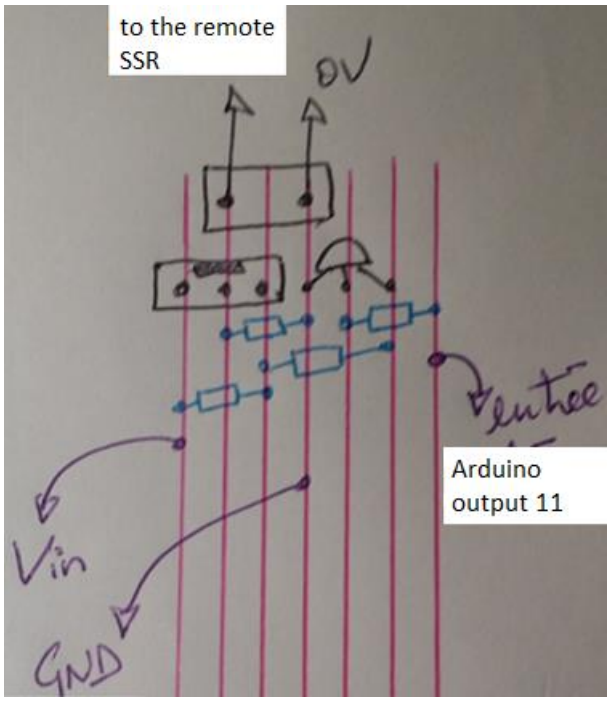
On the left: the BD136 and 2N2222 pinup

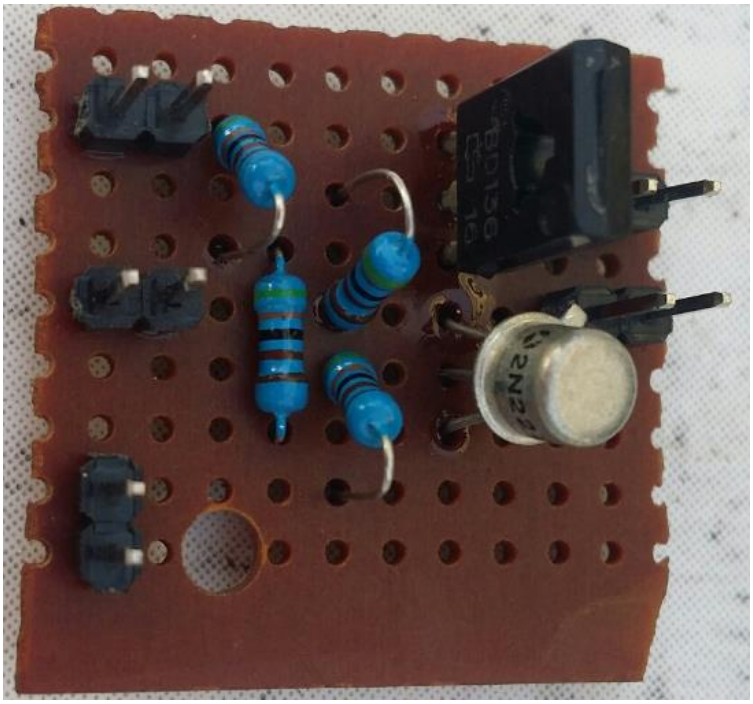


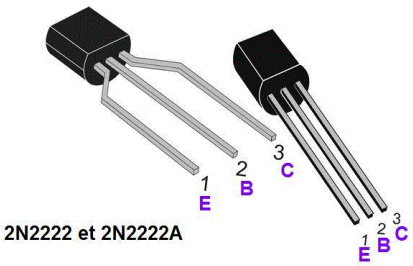
TO-225
CASE 77-09
STYLE 1

The components can take place on the prototype shield. Otherwise a dedicated small shield can receive these components, as shown on the photos on the right and below.

Resistors values are not critical.







The LCD 1602 display with I2C extension

It is not required but it is almost funnier to see how the Power Router is working.

The 1602 LCD is very basic and very little expensive. As it requires many wires, we added an I2C extension which makes it driven with 2 wires only SDA and SCL, respectively on analog output A4 and A5 of the Arduino Uno R3 – the use of these two outputs is required -.

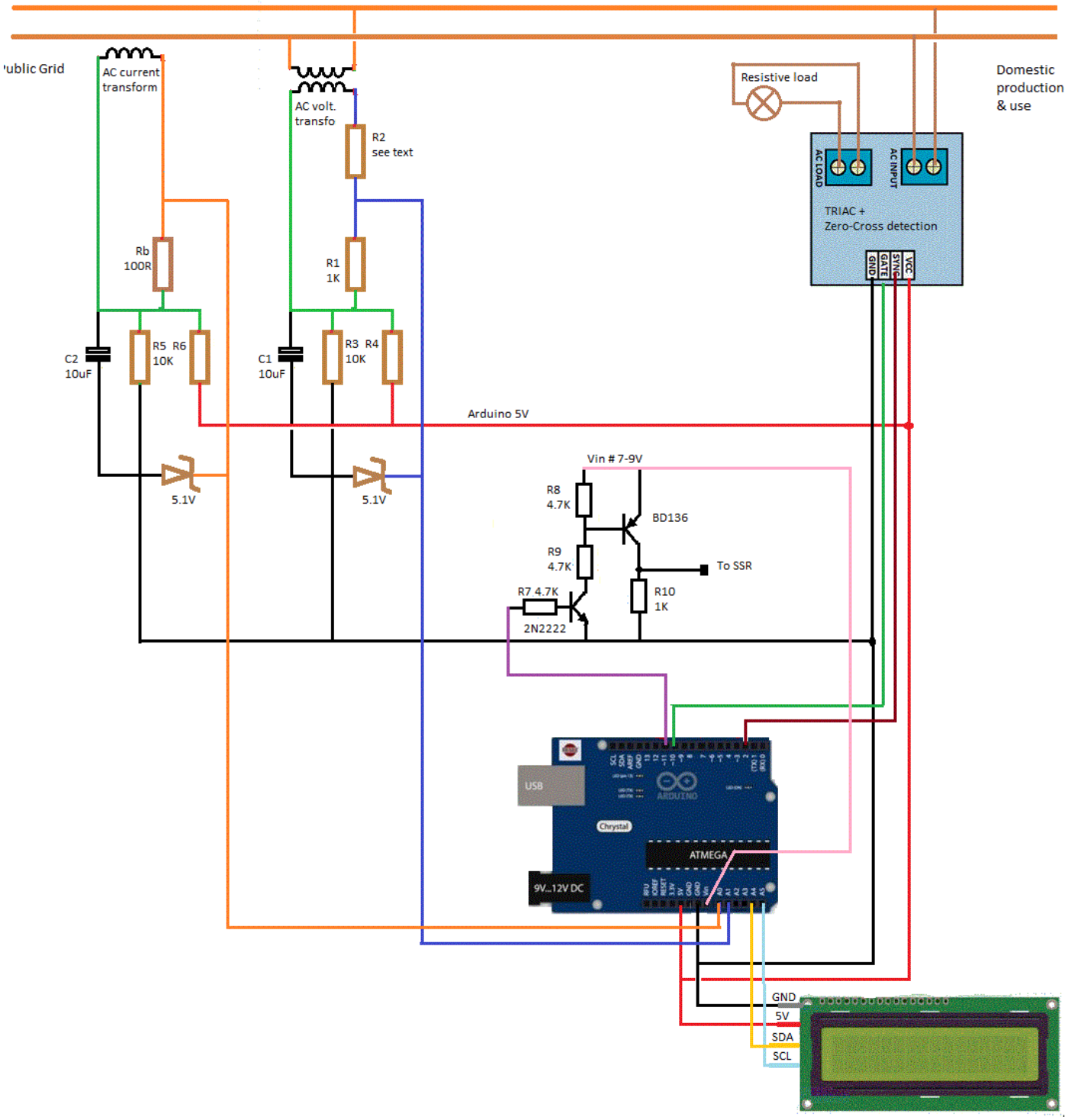
To be able to use the 1602 LCD I2C a specific library is needed, and must be installed on the Arduino IDE.

For information: https://andrologiciels.wordpress.com/arduino/lcd/lcd-1602-i2c/

To get the library: https://app.box.com/s/czde88f5b9vpulhf8z56

All together: the Power Router

The final diagram circuit



You can notice the three push-buttons to setup and review parameters in menus.

The final program

```
/*

A Power Router is a device that detects any homemade electrical energy in exceed. If so
several actions can be taken :
- with the Triac function: gradually switch on a resistive load that will take all the
  exceed energy. This load must be resistive (edison light, water heat, etc...) due to
  the power factor which must stay near 1.
- with the SSR function: prevent against any exceed by performing a load shedding : just
  before injection in the public power grid, either a load can be add to increase
  consumption, or a power generator (solar panel, wind turbine) can be stopped.

The operation is performed by:
- a phase detection between the grid current and the voltage which detect a power consumption
  or a power injection.
```

- in case of injection it initiates the gradual absorption by a resistive load of any exceed power
- the current measure allows the absorption adjustment
- a load shedding becomes active when the system is very near injection and so automatically stops as soon as a level of consumption is raised.

This device is actually in operation for tests in two homes with a power load of 350W and 1000W.

Thanks to Ryan McLaughlin for its description of the triac dimming:
https://web.archive.org/web/20091212193047/http://www.arduino.cc:80/cgi-bin/yabb2/YaBB.pl?num=1230333861/15

```
_____________________________________________________________
|                                                           |
|       author : Philippe de Craene <dcphilippe@yahoo.fr    |
|       Free of use - Any feedback is welcome               |
|                                                           |
_____________________________________________________________
```

Materials :
• 1* Arduino Uno R3 - IDE version 1.8.7
• 1* AC current sensor 20A/25mA
• 1* AC-AC 230V-2.5V transformer
• 1* LCD 1602 with I2C extension
• 1* shield : see manual documentation for wiring implementation
• 1* triac dimming module with zero-cross detection
• 3* push-buttons
• 1* SSR + 1* transistor NPN, 1* transistor PNP, few resistors, capacitors...

Pinup :
 - pin A0 (analog 0)   => AC current sensor
 - pin A1 (analog 1)   => AC voltage sensor
 - pin A4 (analog 4)   => SDA output for LCD
 - pin A5 (analog 5)   => SCL output for LCD
 - pin 2 (numeric 2) => zero-cross detection
 - pin 3 (numeric 3) => output to a LED which indicate pwm
 - pin 4 (numeric 4) => input from push-button "entry"
 - pin 5 (numeric 5) => input from push-button "+"
 - pin 6 (numeric 6) => input from push-button "-"
 - pin 10 (numeric 10) => pwm output that drives the Triac
 - pin 11 (numeric 11) => output to load shedding
 - pin 13 (numeric 13) => overflow LED alarm

Versions chronology:
version 0.5 - 3 may 2018   - first test with a Triac module
version 0.8 - 5 july 2018   - first working version, problem with current measure accuracy
version 1   - 6 july 2018   - tests with EmonLib.h
version 1.8 - 24 sept 2018 - ajustable step for pwm
version 1.9 - 12 oct 2018   - load shedding function added
version 2.0 - 4 nov. 2018   - watchdog and EEPROM added
version 2.4 - 12 jan 2019   - LCD 1602 display added
version 3.2 - 17 jan 2019   - no more EmonLib.h
version 3.4 - 27 avr 2019   - stability improvment, load shedding with delestON and delestOFF
version 3.5 - 9 july 2019 - bug correction if no zero-cress detected which make rebooting infinitely
version 3.6 - 17 july 2019 - menues for parameters setup

*/

```
#include <EEPROM.h>
#include <avr/wdt.h>    // documentation: https://tushev.org/articles/arduino/5/arduino-and-watchdog-timer
#include <TimerOne.h>   // library to install: http://www.arduino.cc/playground/Code/Timer1
#include <LiquidCrystal_I2C.h> // library to install: https://app.box.com/s/czde88f5b9vpulhf8z56

// calibration variables:

bool CALIBRATION = false;   // if true = adjustment of Vcalibration and Icalibration
bool VERBOSE =     false;   // if true = console display BUT VERY SLOW

// Calibration of measures which depend of the hardware. Must be done once at the beginning :
// first: ajust Vcalibration for reading 230V on the console
// after: Icalibration by comparing with a multimeter
// optional: phasecalibration can be adjust with the help of a powermeter

float Vcalibration     = 0.97;   // to obtain 230V
float Icalibration     = 40.6;   // to adjust the current reading to reality
float phasecalibration = 1.7;    // to correct the phase shift due to hardware
byte totalCount        = 20;     // number of half-period studied measuring cycle

// power thresholds in Watts

int limitP    = 1;    // hysteresis of tolerance for the Triac action: if 1 => sensibility is +1W/-1W
int delestON  = 1;              // threshold to start the load shedding
int delestOFF = 350;            // value to stop the load shedding
bool etat_delest_repos  = HIGH;  // inactive state of the load shedding: HIGH for switched on
```

```
// Reactance level to calculate dimstep :
// dimstep evoluates by the factor 'power to dissiate'/reactancelevel
// it is a compromise between reaction speed and instability:
// too small = instability risk, too high = slower
// help how to calculate: reactancelevel ~ (dissipation power of the load in Watts)/40

unsigned int reactancelevel  = 9;

// Arduino inputs and outputs

const byte pushEntryPin     =  4;    // push button 'entry'
const byte pushPlusPin      =  5;    // push button '+'
const byte pushMinusPin     =  6;    // push button '-'
const byte triacPin         = 10;    // pwm output to Triac gate
const byte delestPin        = 11;    // output for load shedding
const byte triacLedPin      =  3;    // LED display Triac activity
const byte limitLedPin      = 13;    // LED for power overflow
const byte voltageSensorPin =  1;    // input from voltage sensor
const byte currentSensorPin =  0;    // input from current sensor
const byte zeroCrossPin     =  2;    // input from zero-cross detection

// variables for interruptions (zero-crossing) :

byte dimmax = 128;              // max value of dim that shuttoff the Triac
byte dim = dimmax;              // Dimming level (0-128)  0 = on, 128 = Off
char periodStep = 75;           // value of the timer (65 for 60Hz, 78 for 50Hz, in µs)
                                // according the formula (500000/AC_freq)/NumSteps = periodStep
                                // 78*128=10ms=1/2 period 50Hz but in fact 75 works better
volatile int i = 0;             // Variable to use as a counter
volatile bool zero_cross = false;       // zero-cross detected for driving the Triac
volatile bool zero_cross_flag = false;  // zero-cross detected for power calcultion

// variables for electrical mesasures

int readV, memo_readV, readI;           // voltage and current in bits (0 à 1023 bits)
float rPower, V, I, sqV, sumV = 0, sqI, sumI = 0, instP, sumP = 0;
byte zero_crossCount = 0;               // halp-period counter

// other variables

int dimstep;                            // value of the increment of dim
unsigned long loadsheddingcounter;      // time counter of load shedding duration
unsigned int memo_temps = 0;
bool delestage = false;                 // load shedding state
bool unefois = false;                   // for one time only flag
bool etat_delest_actif = !etat_delest_repos;  // active loads shedding state
byte ret_push_button = 0;
byte windows = 0;
byte count_before_timeout = 0;
byte refresh_tempo = 2;
byte timeout = 20;

// LCD declaration with I2C :

// documentation : 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);
// => pinup for I2C with l'Arduino Uno R3 : SDA = A4, SCL = A5


//
// SETUP
//_____________________________________________________________________________

void setup() {                          // Begin setup
  pinMode(pushEntryPin, INPUT_PULLUP);  // set the push-buttons as entries pullup to +5V
  pinMode(pushPlusPin, INPUT_PULLUP);
  pinMode(pushMinusPin, INPUT_PULLUP);
  pinMode(triacPin, OUTPUT);            // Set the Triac pin as output
  pinMode(delestPin, OUTPUT);
  pinMode(triacLedPin, OUTPUT);         // Set the LED pin as output
  pinMode(limitLedPin, OUTPUT);

  attachInterrupt(digitalPinToInterrupt(zeroCrossPin), zero_cross_detect, RISING);
// each zeroCrossPin rising generates an interruption : the function 'zero_cross_detect()' is
called
// documentation : https://www.arduino.cc/reference/en/language/functions/external-
interrupts/attachinterrupt/

  Timer1.initialize(periodStep);         // TimerOne from library initialisation
  Timer1.attachInterrupt(dim_check, periodStep);
// for every periodStep time spent, dim_check is called

// EEPROM functions are used to calculate how many time the device has rebooted
// EEPROM stored values are of type char.
```

```
// default values in case of first use are set to 255
  unsigned char reboot_high = EEPROM.read(0);  // to get the high value of the number
  unsigned char reboot_low  = EEPROM.read(1);  // to get the low value of the number
  unsigned int reboot = (reboot_high << 8) + reboot_low;
  reboot++;
  EEPROM.update(0, highByte(reboot));
  EEPROM.update(1, lowByte(reboot));

// EEPROM functions are used to store parameters
  if(EEPROM.read(2) < 200)  limitP = EEPROM.read(2); else EEPROM.write(2, limitP);
  if(EEPROM.read(3) < 50) delestON = EEPROM.read(3); else EEPROM.write(3, delestON);
  unsigned char delestOFF_high = EEPROM.read(4);
  unsigned char delestOFF_low  = EEPROM.read(5);
  int delestOFF_full = (delestOFF_high << 8) + delestOFF_low;
  if( delestOFF_full < 10001 ) delestOFF = delestOFF_full;
  else {
    EEPROM.write(4, highByte(delestOFF));
    EEPROM.write(5, lowByte(delestOFF));
  }
  if(EEPROM.read(6) < 2) etat_delest_repos = EEPROM.read(6); else EEPROM.write(6,
etat_delest_repos);
  if(EEPROM.read(7) < 255)  reactancelevel = EEPROM.read(7); else EEPROM.write(7,
reactancelevel);
  if(EEPROM.read(8) < 31) phasecalibration = (EEPROM.read(8))/10.0; else EEPROM.write(8,
(phasecalibration*10));

// LCD initialisation
  lcd.begin(16,2);                // initialize the lcd for 16 chars 2 lines
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("POWER ROUTER");
  lcd.setCursor(0, 1);
  lcd.print("is starting !");

// console initialisation
  Serial.begin(250000);
  Serial.println ();
  Serial.print("PLEASE NOTE : ");
  Serial.print(reboot);
  Serial.println(" total number of reboots");
  Serial.println();
  Serial.println("Ready to start ...");
  Serial.println ();
  delay(500);
  if( VERBOSE == true ) Serial.print("  Pu (W) || dimstep |  dim ||  load shedding");
  else Serial.println("It is working now !");
  Serial.println();

  digitalWrite(delestPin, etat_delest_repos);    // state output to the default load shedding
state
  wdt_enable(WDTO_500MS);                        // watchdog = reset if no activity longer than
500ms

}                // End setup


//
// ZERO CROSS DETECT : zero-cross interrupt
//_____________________________________________________________________________________

void zero_cross_detect() {   // this function is called at each zero-cross interrupt
  zero_cross_flag = true;    // flag for the real power calculation
  zero_cross = true;         // flag to drive the Triac
}

//
// DIM CHECK : drive the Triac
//_____________________________________________________________________________________

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 is a counter that defines the fire delay. higher is dim longer i
will count
      digitalWrite(triacPin, HIGH);        // and later the triac will fire
      delayMicroseconds(50);               // Pause briefly to ensure the triac turned on
      digitalWrite(triacPin, LOW);         // Turn off the Triac gate, but the triac stays
switch on until OV
      i = 0;                               // Reset the counter for the next cycle
      zero_cross = false;
    }
    else i++;    // If the dimming value has not been reached, increase it
  }              // End zero_cross check
}                // End dim_check function


//
```

```
// LOOP
//_____________________________________________________________________________________

void loop() {                   // Main Loop

// 1st part: calculation of the real electric power rPower
//_____________________________________________________________________________________
//

  unsigned int numberOfSamples = 0;
  sumV = 0;
  sumI = 0;
  sumP = 0;
  unsigned int temps_actuel = millis()/1000;      // get the time spent in seconds

// increment zero_crossCount at each zero-cross until totalCount, then rPower value is
calculated

  if( zero_crossCount >= totalCount ) zero_crossCount = 0;

// as most as possible number of measures between the totalCount number of half-periods
// themself defined by zero-cross flag
  while( zero_crossCount < totalCount ) {
    if( zero_cross_flag == true ) {         // increment of half-period count for each zero-
cross
      zero_cross_flag = false;
      zero_crossCount++;
    }
    numberOfSamples++;                      // number of measures
    memo_readV = readV;                     // memorize the past value
    readV = analogRead(voltageSensorPin);   // voltage measure in bits - 0V = bit 512
    delayMicroseconds(50);
    if( readV == memo_readV && readV > 509 && readV < 515 ) {   // test if no grid
      lcd.setCursor(0, 0);
      lcd.print("ABSENCE DE      ");
      lcd.setCursor(0, 1);
      lcd.print("TENSION SECTEUR ");
      delay(200);
      goto nogrid;                          // exit to the end of program
    }
    readI = analogRead(currentSensorPin);   // current measure in bits - 0A = bit 512
    delayMicroseconds(50);

// calculation of the effective values of voltage and current
    if( CALIBRATION == true ) {                   // for calibration only
      sqV= (readV -512.0) * (readV -512.0);       // -512 as offset to get 0V = bit 0
      sumV += sqV;
      sqI = (readI -512.0) * (readI -512.0);
      sumI += sqI;
    }    //   end test upon CALIBRATION

// instant power calculation
    instP = ((memo_readV -512.0) + phasecalibration * ((readV -512.0) - (memo_readV -512))) *
(readI -512.0);
    sumP +=instP;
 }      // End of while upon zero_crossCount

// memorization of the values
  if( numberOfSamples > 0 ) {
    if( CALIBRATION == true ) {
      V = Vcalibration * sqrt(sumV / numberOfSamples);
      I = Icalibration * sqrt(sumI / numberOfSamples);
    }
    rPower = ((Vcalibration * Icalibration * sumP )/ numberOfSamples) / 1000.0;
  }

// 2nd part: dim and dimstep calculation to drive the Triac, and load shedding management
//_____________________________________________________________________________________
//

// dimstep calculation: higher is the power to take in charge, higher will be dimstep
  if( rPower > 0 ) { dimstep = rPower/10/reactancelevel + 1; }
  else { dimstep = 1 - rPower/10/reactancelevel; }

  if( rPower > limitP ) {             // injection increases, the delay to fire the Triac
decreases
    if( dim > dimstep )  dim -= dimstep; else  dim = 0;
  }
  else if( rPower < -limitP ) {       // injection decreases, the delay to fire the Triac
decreases
    if( dim + dimstep < dimmax ) dim += dimstep;  else  dim = dimmax;
  }

  if( dim < 1 ) digitalWrite(limitLedPin, HIGH);     // overload LED
  else { digitalWrite(limitLedPin, LOW); }
  analogWrite(triacLedPin, dimmax-dim);              // Write the value to the LED for testing
```

```
// load shedding management
  if( rPower > -delestON) { delestage = true; }      // threshold activation value for load
shedding

  if( delestage == true ) {
    if( unefois == false ) {
      digitalWrite(delestPin, etat_delest_actif);     // load shedding driver update
      loadsheddingcounter = temps_actuel;             // for load shedding spent time
      unefois = true;
    }
    if( rPower < -delestOFF ) {                       // threshold inactive value for load
shedding
      digitalWrite(delestPin, etat_delest_repos);     // load shedding driver update
      unefois = false;
      delestage = false;
    }
  }           // end of test upon delestON

// LCD and menues management
//_____________________________________________
//
// display update and push-button request every 2 seconds
  if( temps_actuel >= memo_temps + refresh_tempo ) {
    memo_temps = temps_actuel;
    ret_push_button = push_button();       // reading push-button status here only
    lcd.clear();
    lcd.setCursor(0, 0);
    if( ret_push_button == 1 ) next_windows();   // if 'entry' pushed increase of window
    if( windows == 0 ) {
      lcd.print("P= ");
      lcd.print(String(-rPower,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 ");              // load shedding in French
      if( delestage == true ) {
        lcd.print(temps_actuel - loadsheddingcounter);
        lcd.print("s    ");
      }
      else { lcd.print("ARRETE"); }        // stoped in French
    }
    else {                                 //  end of window 0, start of parameters review
      count_before_timeout++;
      if( count_before_timeout > timeout ) {   // timeout to return to usual display if no job
done
        count_before_timeout = 0;
        windows = 0;
      }
      if( windows == 1 ) {
        if(ret_push_button == 2) limitP++;    // if "+" pushed
        if(ret_push_button == 3) limitP--;    // if "-" pushed
        limitP = constrain(limitP, 1, 200);
        lcd.print("SEUIL DETECTION");
        lcd.setCursor(0, 1);
        lcd.print("seuil = ");
        lcd.setCursor(8, 1);
        lcd.print(limitP);
        lcd.print("W");
      }  // end of windows 1
      if( windows == 2 ) {
        if(ret_push_button == 2) delestON++;    // if "+" pushed
        if(ret_push_button == 3) delestON--;    // if "-" pushed
        delestON = constrain(delestON, 1, 50);
        lcd.print("DELESTAGE ACTIF");
        lcd.setCursor(0, 1);
        lcd.print("seuil = ");
        lcd.setCursor(8, 1);
        lcd.print(delestON);
        lcd.print("W");
      }  // end of windows 2
      if( windows == 3 ) {
        if(ret_push_button == 2) delestOFF+= 50;    // if "+" pushed
        if(ret_push_button == 3) delestOFF-= 50;    // if "-" pushed
        delestOFF = constrain(delestOFF, 50, 10000);
        lcd.print("DELESTAGE ARRET");
        lcd.setCursor(0, 1);
        lcd.print("seuil = ");
        lcd.setCursor(8, 1);
        lcd.print(delestOFF);
        lcd.print("W");
      }  // end of windows 3
      if( windows == 4 ) {
        if( ret_push_button > 1 ) etat_delest_repos =! etat_delest_repos;
        etat_delest_actif =! etat_delest_repos;
```

```
        lcd.print("DELESTAGE :");
        lcd.setCursor(0, 1);
        if( etat_delest_repos == HIGH ) lcd.print("DEMARRE au repos");
        else lcd.print("ARRETE au repos");
      }   // end of windows 4
      if( windows == 5 ) {
        if(ret_push_button == 2) reactancelevel++;    // if "+" pushed
        if(ret_push_button == 3) reactancelevel--;    // if "-" pushed
        reactancelevel = constrain(reactancelevel, 1, 254);
        lcd.print("COEF DE REACTION");
        lcd.setCursor(0, 1);
        lcd.print("taux = ");
        lcd.setCursor(7, 1);
        lcd.print(reactancelevel);
      }  // end of windows 5
      if( windows == 6 ) {
        byte phasecalibrationDEC = phasecalibration*10;
        if(ret_push_button == 2) phasecalibrationDEC++;    // if "+" pushed
        if(ret_push_button == 3) phasecalibrationDEC--;    // if "-" pushed
        phasecalibrationDEC = constrain(phasecalibrationDEC, 1, 30);
        phasecalibration = phasecalibrationDEC/10.0;
        lcd.print("CALIBRATION Pu");
        lcd.setCursor(0, 1);
        lcd.print("valeur = ");
        lcd.setCursor(9, 1);
        lcd.print(phasecalibration);
      }  // end of windows 6

// EEPROM updated if needed
      EEPROM.update(2, limitP);
      EEPROM.update(3, delestON);
      EEPROM.update(4, highByte(delestOFF));
      EEPROM.update(5, lowByte(delestOFF));
      EEPROM.update(6, etat_delest_repos);
      EEPROM.update(7, reactancelevel);
          EEPROM.update(8, (phasecalibration*10));

    }   // end of paramerter review
  }     // end of display management

// console display
  if( CALIBRATION == true ) {
         Serial.print(V);
         Serial.print("  |  ");
    Serial.print(I/1000);
    Serial.print("  |  ");
    Serial.print(rPower);
    Serial.println();
  }
  if( VERBOSE == true ) {
    Serial.print(rPower);
    Serial.print("  ||     ");
    Serial.print(dimstep);
    Serial.print("  |  ");
    Serial.print(dim);
    Serial.print(" ||  ");
    Serial.print(" load shedding : ");
    Serial.print(delestage);
    Serial.print(" seconds : ");
    Serial.println(temps_actuel - loadsheddingcounter);
  }
  else delay(1);               // required for stability
  nogrid:
  wdt_reset();                 // watchdog reset

}                              // end of main Loop

//
// NEXT_WINDOWS : next window procedure
//_____________________________________________________________________________

void next_windows() {

  windows = (windows+1) % 7;        // next windows modulo 6
  ret_push_button = 0;              // reset the buttun state
  lcd.clear();
  lcd.setCursor(0, 0);
}     // end of next_windows function

//
// PUSH_BUTTON : return value depending of the state of the 3 push-buttons
//_____________________________________________________________________________

byte push_button() {

  if( digitalRead(pushEntryPin) == 0 ) {
    count_before_timeout = 0;         // reset the timeout counter
```

```
    return 1;
  }
  if( digitalRead(pushPlusPin) == 0 ) {
    count_before_timeout = 0;          // reset the timeout counter
    refresh_tempo = 1;                 // temporary lower display update duration
    return 2;
  }
  if( digitalRead(pushMinusPin) == 0 ) {
    count_before_timeout = 0;          // reset the timeout counter
    refresh_tempo = 1;                 // temporary lower display update duration
    return 3;
  }
  refresh_tempo = 2;                   // go back to initial value
  return 0;
}     // end of push_button function
```

Some tips about the program

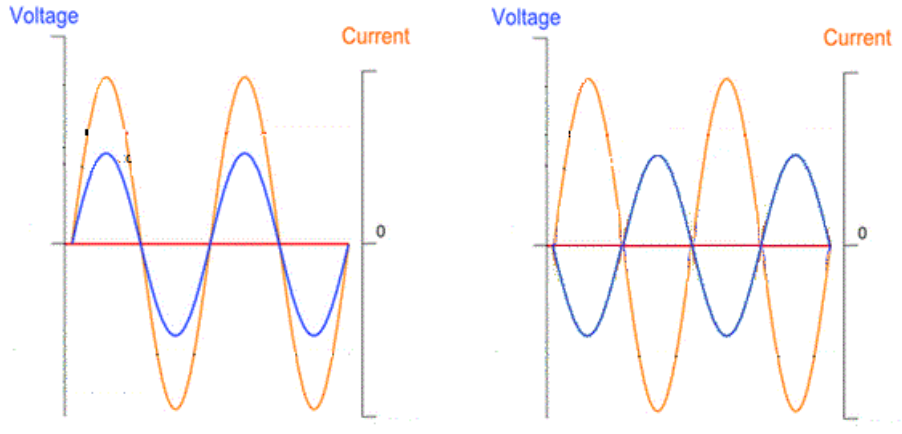
The three main tasks the Power Router must perform are :

1. To detect the shift between grid current and voltage to detect injection,
2. To measure the among of power that are injected, then drives the dimming amount to the Triac to cancel injection,
3. Upon the amount of consumption it drives a SSR for load shedding.

Voltage and current shift detection

Ideally, we should have very nice sinusoidal curves for voltage and current that would look like :

"consumption mode "injection mode""



But in real life if the voltage curve is like a sinus, the current curve is really very distorted and mostly looks like the back of a dromedary. This is because nowadays most of electric equipment own a AC-AC or AC-DC converter that use cutting mode of power supply. As a result of the use of nonlinear supplies, the current curve is no more sinusoidal.

Moreover, the action of the Triac is itself a cutting mode and so a nonlinear device, so how can we measure the current?

The answer is: if the current curve is anything else but not a nice sinusoid, nevertheless the average value of the current still remain appropriate for an average value (not instant). So we will make as many as measures as possible during a fixed period of time and then calculate the average. The time stamp for the multiple measures is defined with 'totalCount = 20': the measures are done for 20 half periods of the grid: 50Hz => 2 half-periods =20ms, so 20 half-periods allow one measure every 200ms, 5 measures per second.

Instructions are in the "1st part: calculation of the real electric power rPower".

If rPower is positive, the homemade energy is injecting to the grid. If negative, we are consuming.

For the best accuracy all measures and calculations are done in bits.

The Triac dimming

We have seen in the chapter “The Triac module” that for a dimming effect the signal which switch on the Triac is applied after a delay once the zero-cross is passed. The Triac will switch off next zero cross.

To be sure to always fire the Triac at the right time, each zero-cross creates an interrupt: whatever the program is doing, an interrupt makes the program stop its actual ‘occupations’ and run a specific sequence, in our case set up the flag ‘zero_cross’. At the same time ‘zero_cross_flag” is also set up for ‘rPower’ calculation purpose.

The way to define the delay to fire the Triac is set in the “2nd part : dim and dimstep calculation to drive the Triac …”. The method used is “observe and act”: if the injection increases then ‘dim” decreases, so the counter ‘i’ in the “dim_check() function will count less which increase conductivity of the Triac. In the opposite, if the injection decreases then ‘dim’ increases and the counter ‘i’ will take longer time to reach the fire time.

If there is no injection ‘dim’ stays stuck to its maximal value 128. TimerOne is set in order to propose 128 states for the Triac, from 0 = always switch on, to 128, always switch off.

By the way this algorithm is reactive thus there is a little injection Yes there is a little bit. But nobody can guess the future, even an Arduino device, and the exact time a device will perturb the global consumption on the grid line.

However the amount of tolerate injection is fixed by the sensibility level with ‘limitP’. Default value is 1 watt. The faster the program can run, the faster it can correct the diming level of the Triac to prevent more injection than ‘limitP’.

In addition dim is not incremented by step of +1/-1, but by ‘dimstep’: this variable is proportional to ‘rPower’. Bigger is ‘rPower’, bigger will be ‘dimstep’, so larger will be the instant variation of ‘dim’. To prevent against any instability the formula for ‘dimstep’ calculation includes the ‘reactancelevel’ which weigh the ‘dimstep’ variations.

Important tip: the load connected to the Triac must be the most resistive possible. If not there will be an additional shift between voltage and current, which would fake the measures.

At last, this functionality of the Power Router has not been thought in order to increase energy efficiency: it is lost energy that may light on a 300W or a 500W bulb. In fact it has been thought in order to never inject energy to the grid. Take in mind that this bulb should rarely bright, anyway very occasionally and very weakly. If not, then the 3rd main task functionality of the device must be better studied.

SRR load shedding

With a SSR there is no dimming ability. It only can be switch on or off. So any electric equipment can be driven by a SSR. So the idea is that a SSR will serve for lead shedding: it starts when we are very near to injection, the level depends of ‘delestON’ and stops once the consumption is greater than stated with ‘delestOFF’.

‘etat_delest_repos’ define is the load shedding is normally active or not. So we can imagine many possible things to do: either to cook our bread, or to disconnect a solar panel, etc…. as soon as when we are in exceed of homemade electric power production.

Instruction for use

How to set parameters

You will need to compilethe program at least three times:

1- The first time with ‘`VERBOSE = false;`’ and ‘`CALIBRATION = true;`’

Then with the help of the console you will probably need to adjust 'Vcalibration' and 'Icalibratiuon'. Each update need a recompile to download the program from the IDE to the Arduino. However, once it is done, it is definitively done, except if your replace the AC current or voltage transformer by another model with different characteristics.
'Vcalibration' is adjusted to get 230V (in Europe).
'Icalibration' is adjusted to get the good current value in comparison with a known load or a multimeter.
'phasecalibration' can be adjusted now or later in the parameters menus in normal operation.

2- The second time '`VERBOSE = true;`' and '`CALIBRATION = false;`'

According to the test circuit described below, you can have a look to the way variables change depending of the simulation of injection you apply.

3- Finally you can set '`VERBOSE = false;`' and '`CALIBRATION = false;`'

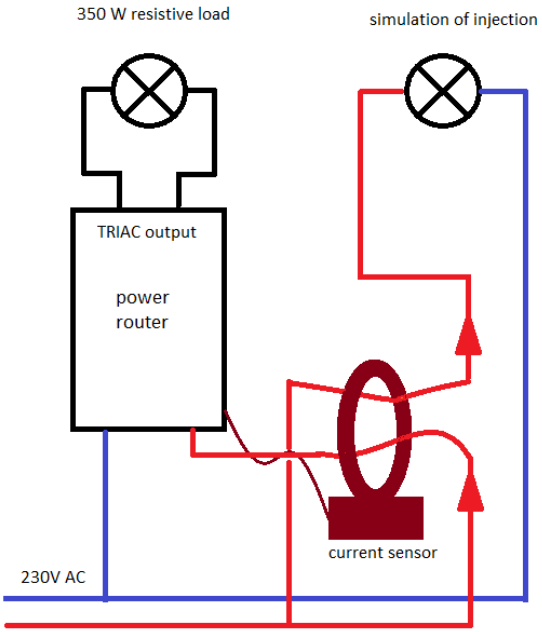
It should be the setting for normal operation mode. Therefor several parameters can be adjusted once the Power Router is in operation. To enter in parameters just press 'Entry' push-button for more than 2 seconds. Each press will change the parameter to review, then "+" or "-" push-buttons press (more than 2 seconds) increase or decrease the value:

- 'limiP' define the threshold of Triac operation. Default value is '1', that means that 'dim' will increase or decrease within an hysteresis of +/-1W : 1W of injection to start the lighting or decrease the dimming, 1W of consumption to increase the dimming until switch off of the load. So the sensibility is 2W. limitP can be defined from 1W to 200W.
- 'delestON' define the power consumption value before the load shedding becomes active. Default value is '1', for 1W. This value can be increase from 1W to 50W.
- 'delestOFF' define the power consumption value before the deactivation of the load shedding. Default value is 350W. It can be defined from 50W to 1000W. Of course it would be silly to adjust 'delestON = 50' and 'destOFF = 50'….
- 'etat_delest_repos' define if an inactive load shedding is a switch on or a switch off of the SSR. By default it is set to active.
- 'reactancelevel' define the variable step evolution of dim. The step is proportional to the power measured. The value of 'reactancelevel' balances the variable step: too high the step will stick to 1 whatever the variation of 'rPower' and the Power Router will need much more time to reach the final value. In opposite, a too small value will make too big step variations and the Power Router may oscillate. The value can be defined by this rule: reactancelevel = (power absorption of the resistive load) / 40
  With a load of 350W, the 'reactancelevel' is 9. It is the default value.
- 'phasecalibration' correct the phase shift due to hardware. Range is from 0.1 to 3.0. Default is 1.7. You may need to adjust it to get the true value of power, if you have a powermeter to make the comparison.

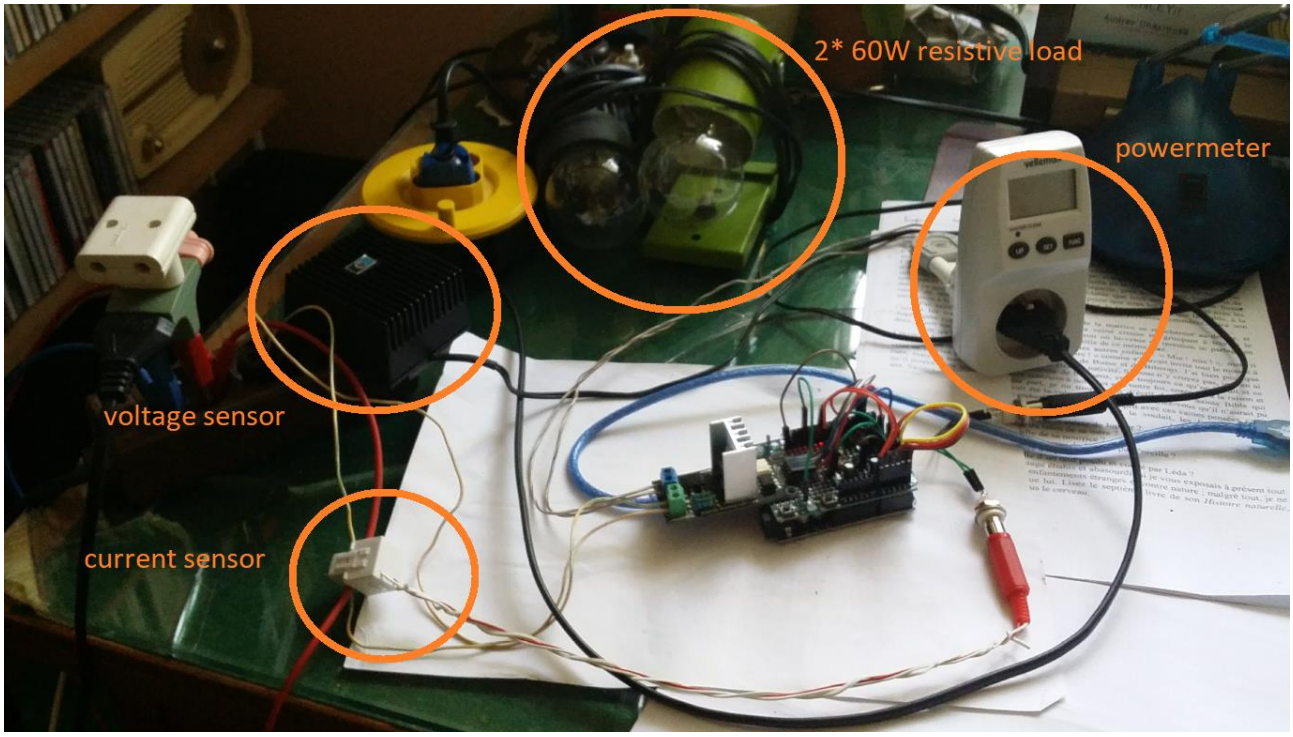
Method for testing

If you want to test the Triac diming function before to install the Power Router in operation, you can built a circuit that will simulate injection. Instead of having 1 wire (from the grid) through the current sensor, you have two : 1 wire for the grid, 1 wire for a load that will simulate the homemade energy. In fact it is the way the current will pass that simulate: in opposite way the current measured by the sensor = “simulation of injection” minus “350W resistive load”.

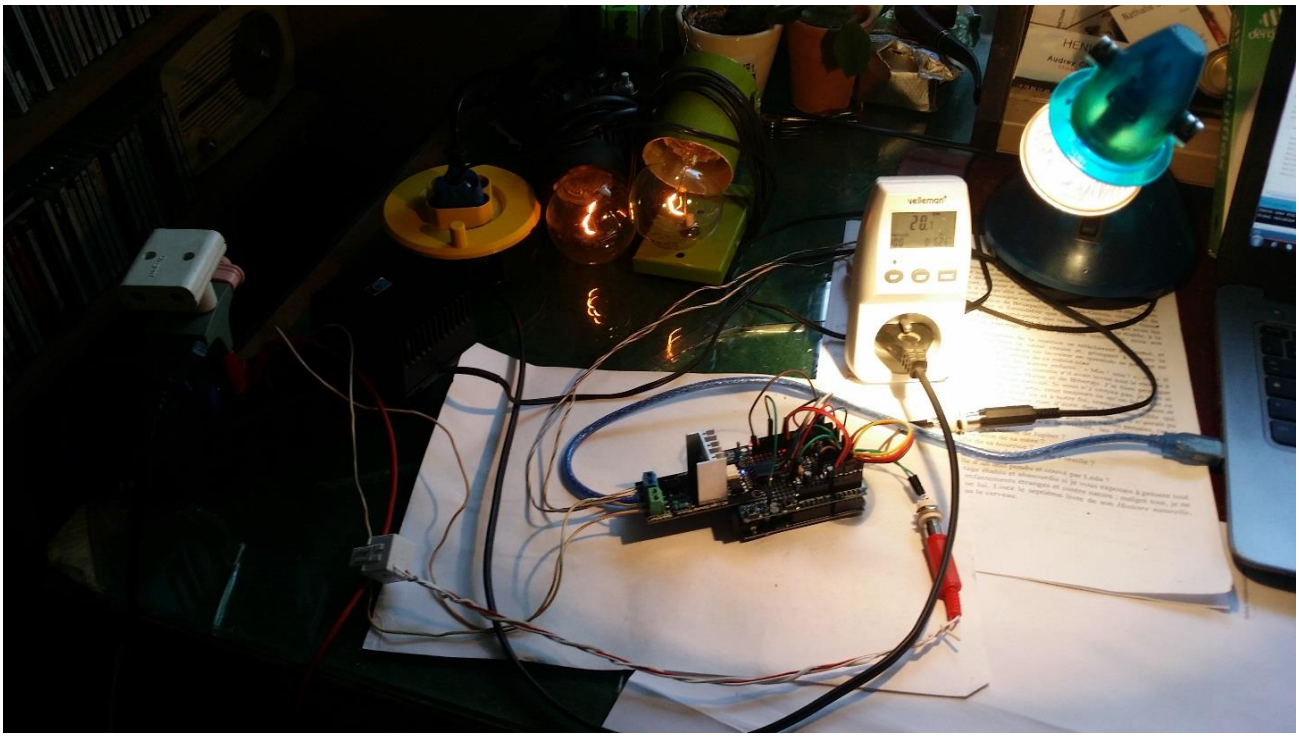
Note: if tests show that the Triac resistive load is always full bright after a while, it may be due to a wrong way the wires pass through the sensor.



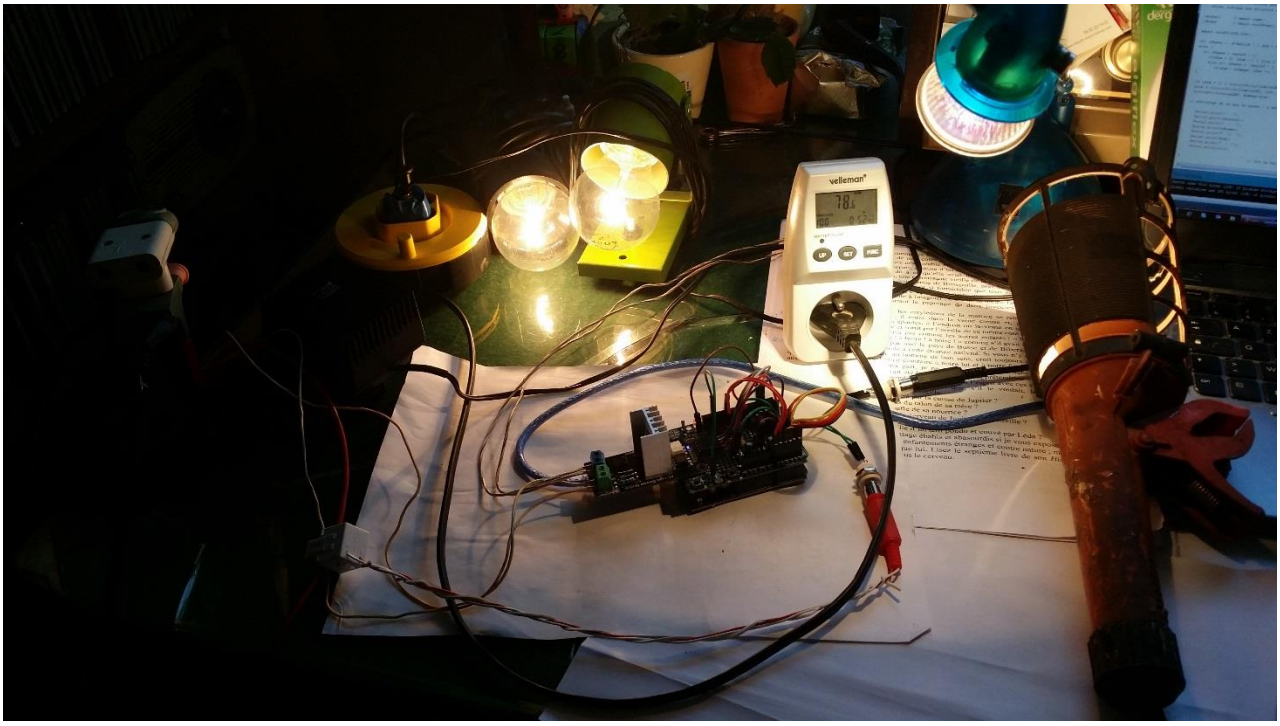
For illustration here are some pictures of the project at its very early state for tests for various cases.



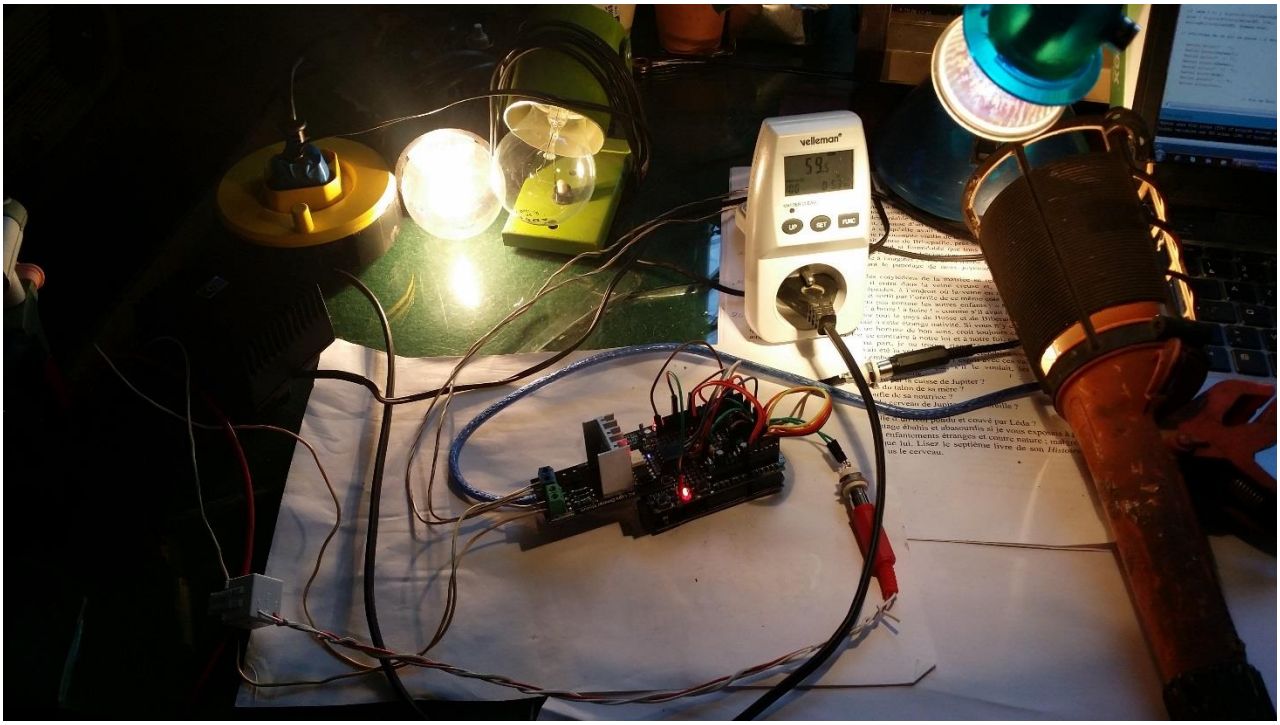
Case 1: the “injection” of 20W (the bleu lamp on the right). Note the resistive load just lights up a little.



Case 2 : another 60W light bulb has been added to the “injection” for a theoretical total of 80W. The resistive load is lighting much more.



Case 3 : the resistive load has been reduce to 60W only (1 light bulb). Note the red LED for overflow is on.

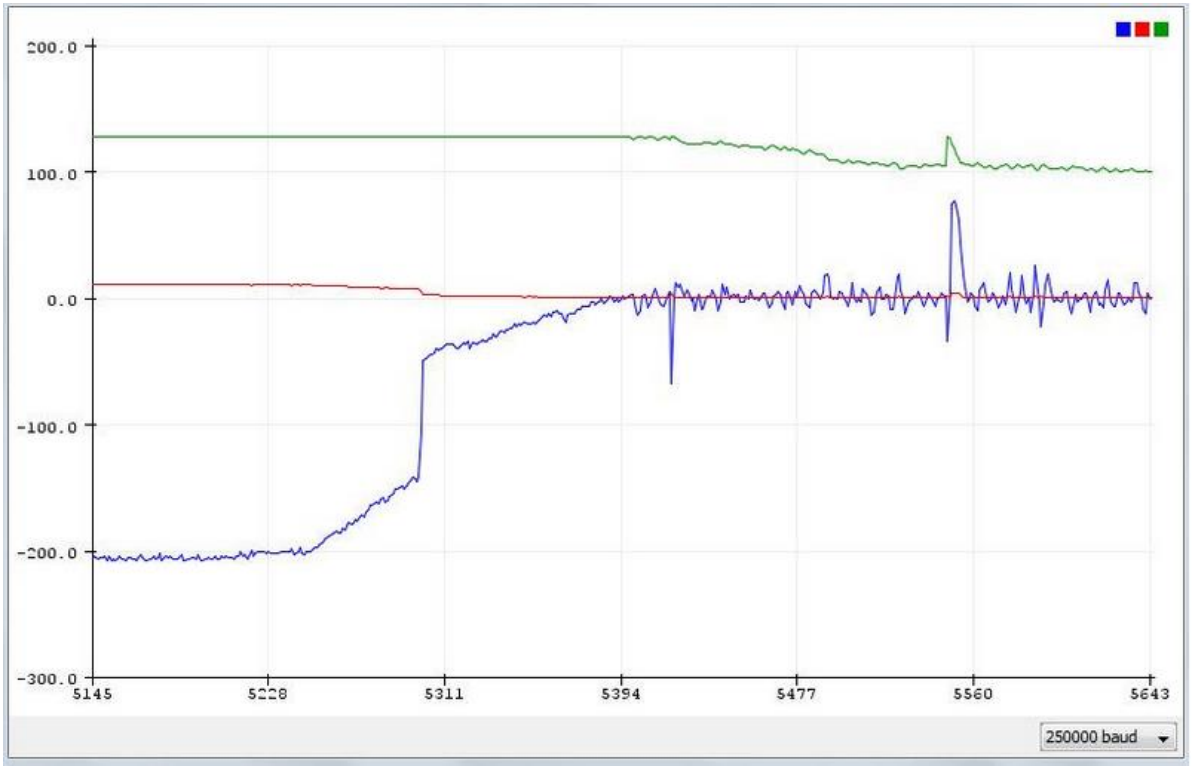


Illustrations in use

blue curve = rPower, red curve = dimstep, green curve = dim

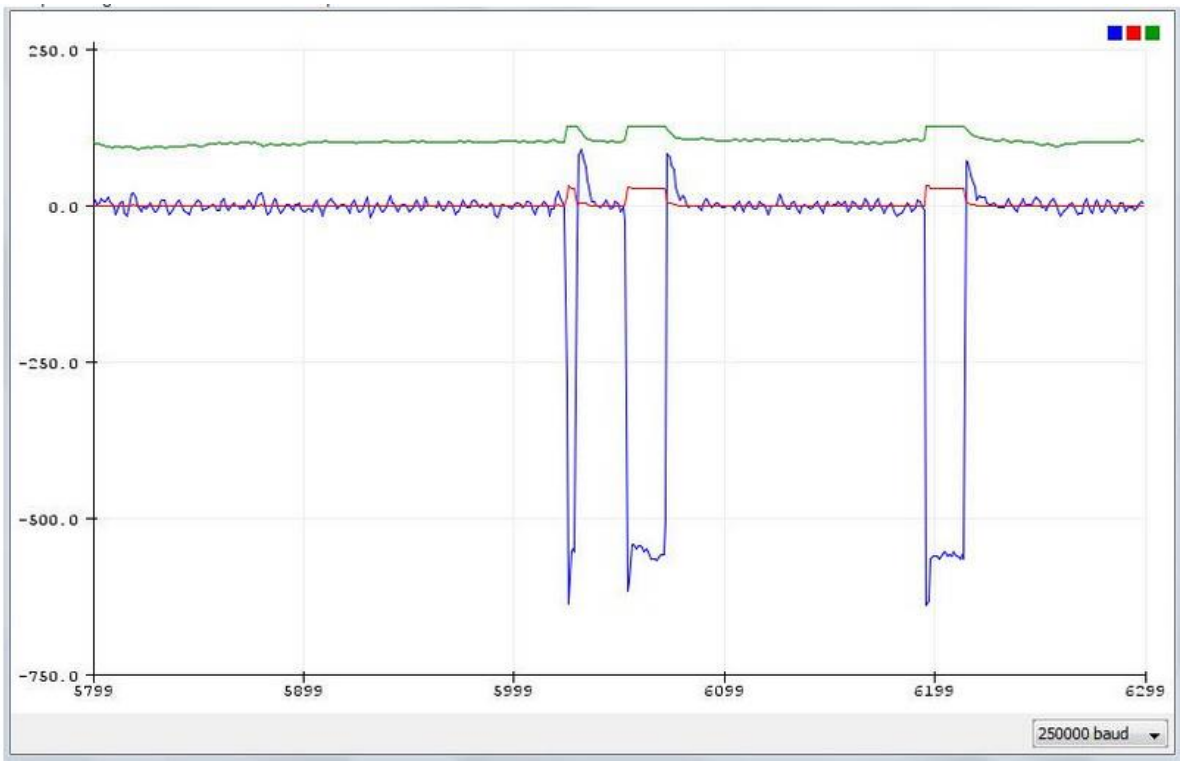
Case 1: at 5220 the wind turbine inverter is switch on. The grid consumption decreases, to reach 0 at 5394, dim decreases and the resistive load starts to light up.

At 5560 a device in the house has stopped (the fridge?). There is a quick positive pic of injection then dim increases to cancel any injection as fast as possible.



Case 2 : the grid power consumption is at 0, ‘dim’ adapts himself and send all exceeded power to the resistive load.

3 times there are a sudden pic of consumption. During these moment ‘dimstep’ and ‘dim’ changes as suddenly as possible. We can notice a little injection just after the consumption stop. The parameter ‘reactancefactor’ can attenuate this default. Note that with console message the speed of the program is very slow.



The photo below illustrates a way to build the Power Router. The Red RCA cinch on the right is the arrival of the current sensor. The voltage sensor is inside the box device, as it uses the same incoming 230V than the Arduino power supply which is also inside the box. On the left there is the 350W light bulb. Below the box another RCA may be notice: it is the SSR driver command.

As this device is still in progress – since September 2018 – the USB cable is still in place in order to be able to download any new program version. Note that the three push-buttons are not yet in place when this photo has been taken.

According to default values set in the program, the SSR is wire to manage the configuration diagram below: whenever the consumption is as near as 1W the load shedding becomes active, the SSR switches off (the default inactive state is switch on), the wind turbine invertor cannot operate anymore and the wind turbine can only charge the batteries.

As soon as the consumption exceeds 350W, the load shedding becomes inactive, the SSR switches on back again, and so on.