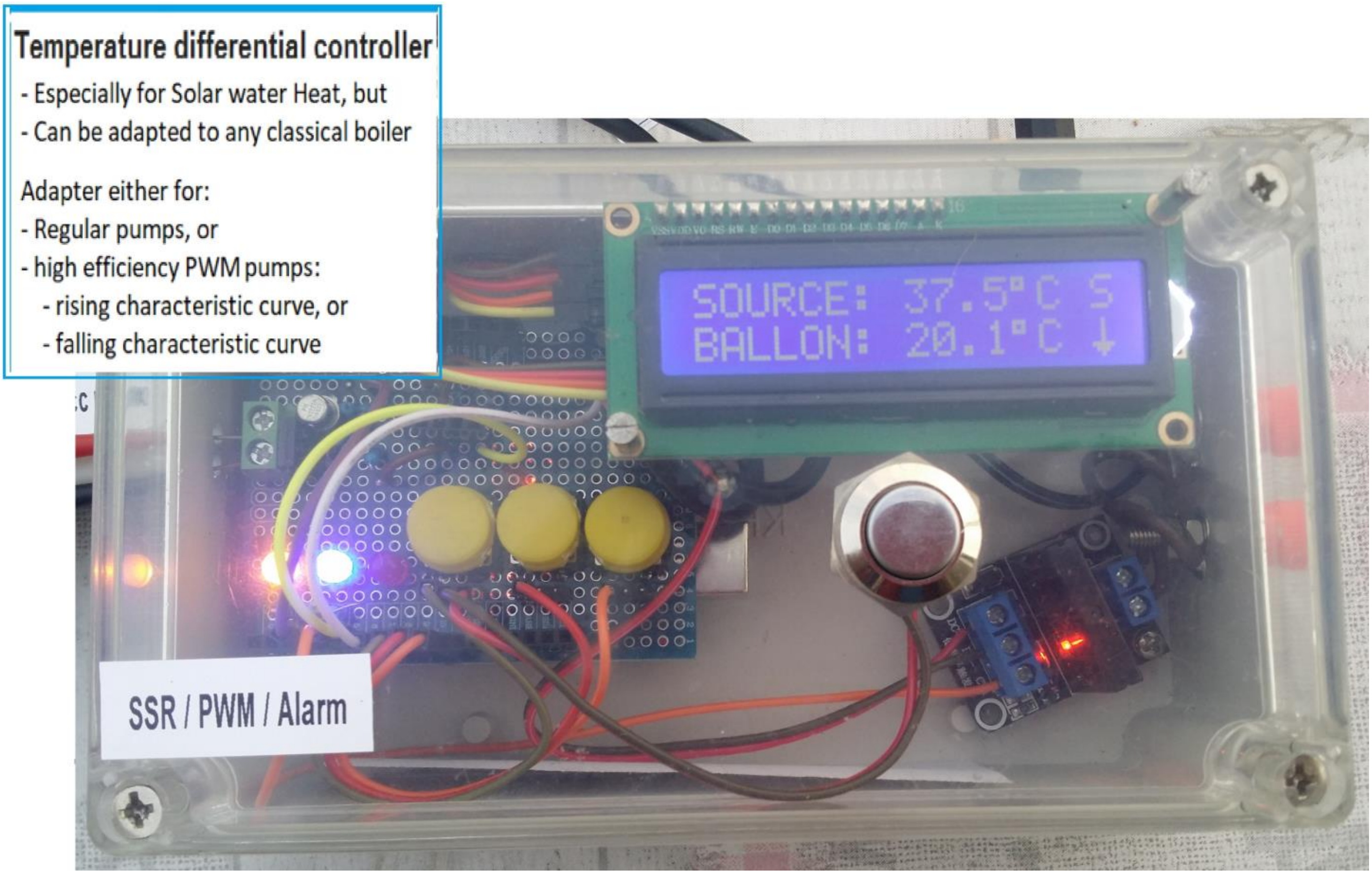
How to make a Water Heater Thermostat for PWM pumps



Author : Philippe de Craene dcphilippe@yahoo.fr

August 2018 – April 2020

Manual version: 2.0
Program version: 4.0

There are hundreds of homemade thermostats with various MCU and sensors. Some are very simple, others are very sophisticated. However I decided to build my own thermostat especially for a solar panel and a new generation high efficiency PWM pumps. Moreover this thermostat offers:

- Cost less than 20€, all included, by the use of very low cost components like the Arduino Uno R3, and two waterproof DS18B20, and a SSR
- Can be use with any kind of water boiler and any kind of pump:
  - Regular pump or Pressure regulated high-efficiency pump
  - High-efficiency pump with a PWM profile for a rising characteristic curve
  - High-efficiency pump with a PWM profile for a falling characteristic curve
- All parameters can be easily set and recovered after a restart,
- Can be built by any beginner in soldering some electronic components

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

Table of content

Introduction

1. First measuring the temperature of a source which may be a solar thermal panel, or a boiler, and the temperature of the cumulus.

Depending on the result of these measurements the circulator – the pump - will be started – or not – :

- if Tsouce < Tmini (safety against freezing) => the pump is started + ALARM
- if Tcumulus > Tmaxi (safety against boiling) => the pump is stopped + ALARM
- if Tsource – Tcumulus > detlaTon => the pump is normally starting
- if Tsource – Tcumulus < deltaToff => the pump stops.

2. Then measuring the tendency to heating or cooling cumulus:
- When the pump is running and the cumulus rises in temperature, from a threshold the PWM reduce the pump speed.
- On the contrary, when the cumulus cools, the PWM increases the pump speed.
- A switchable AC230V for either:
    - the use of a regular pump or a pressure regulated high-efficiency pump,
    - or starts the boiler if Tsource is Tsource is below a threshold. Then the pump will start as soon as Tsource rise another threshold. If Tsource does not increase: ALARM

3. So the device can operate in 3 modes. As there are 2 inputs: the temperature sensor, and 2 outputs: the SSR switch for AC230V and the PWM signal, the role of the output depends of the mode:
- Mode Regular Pump: PWM is not used, AC230V SSR drive the pump.
- Mode Rising PWM: PWM drives the pump, AC230V SSR can drive either the boiler, or a frost protection, or an overheating protection (you can choose only one role in this list)
- Mode Falling PWM: PWM drives the pump, AC230V SSR can drive either the boiler, or a frost protection, or an overheating protection (you can choose only one role in this list)

These modes and roles are defines in the menu and can be adjusted whenever desired.

List of materials

1- 1x Arduino Uno R3

2- 1x LCD 1603 with I2C extension

3- 1x prototype shield or the PCB

4- 2x temperature sensors DS18B20

5- 1x SSR of the right power: it must allow 200W for a pump, or 2000W (or even more) for a boiler

6- 1x power supply for the Arduino from 9V to 12V DC (ideally 10V)

7- Resistors of 4,7k, 3x of 220Ω, 1x red LED rouge, 1x blue or green LED, 1x yellow LED, 3x push-buttons, bipolar transistors: 1x NPN like 2N2222, 1x PNP like BD136

Device around its components

Temperature sensor DS18B20

DS18B20 is 1-Wire interface Temperature. This one digital pin is used for two way communication with a microcontroller. https://en.wikipedia.org/wiki/1-Wire

The sensor comes usually in two form factors**.** One that comes in TO-92 package looks exactly like an ordinary transistor. Other one in a waterproof probe style which can be more useful when you need to measure something far away, underwater or under the ground.

DS18B20 temperature sensor is fairly precise and needs no external components to work. It can measure temperatures from -55°C to +125°C with ±0.5°C Accuracy. The resolution of the temperature sensor is user-configurable to 9, 10, 11, or 12 bits. However, the default resolution at power-up is 12-bit (i.e. 0.0625°C precision).

One of the biggest advantages of DS18B20 is that multiple DS18B20s can coexist on the same 1-Wire bus. As each DS18B20 has a unique 64-bit serial code burned in at the factory, So they can be wired in parallel. But this has a disadvantage however: we do not now *apriori* which DS18B20 will give the Tsource or Tcumulus. **You will have to test the device and identify the role of the sensor before them installation.**

The display LCD 1602

It is the cheapest and the easiest to implement display, composed of 2 lines of 16 characters each. It is usually driven with 4 data ports for a total of 6 ports. In order to reduce the wiring, a I2C converter is used. The protocol I2C has 2 communication port only: SDA and SCL.

Datasheet is there: https://www.openhacks.com/uploadsproductos/eone-1602a1.pdf

The Arduino Uno R3 requires SDA to A4 and SCL to A5 analog inputs.

The use of the EEPROM to keep parameters

The use of the EEPROM allows to store the data at each restart of the device.

Here is the list of parameters:

- deltaTon : threshold to start the pump, default value is: +5°K
- deltaToff : threshold to stop the pump, default value is: +2°K
- deltaTpwm : threshold to drive the PWM cycle ratio, default value is: 2°K
- Tmaxi : maximum temperature allowed before the pomp stops + ALARM, default value is: +85°C
- Tmini : minimum freezing temperature before the pomp starts, default value is: -1°C
- Pump_model: stand-alone pump (without PWM), rising PWM pump, falling PWM pump. Default : stand-alone
- SSR_rule: drives the pump, drives the boiler, drives a overheat alarm, drives risk of freezing alarm. Default: drives pump.

Note that the EEPROM works with type ‘byte’ numbers, so positive integers until 255. An offset of 50 is done to prevent against negative values.
For more information: https://www.arduino.cc/en/Reference/EEPROM

The PWM feature

How PWM works

Description
UPM3 Solar is a OEM high efficient circulator offering flexible solutions for thermal solar systems now and in the future. It is designed to work both with and without PWM signal, allowing you to upgrade your systems without having to change the circulators.

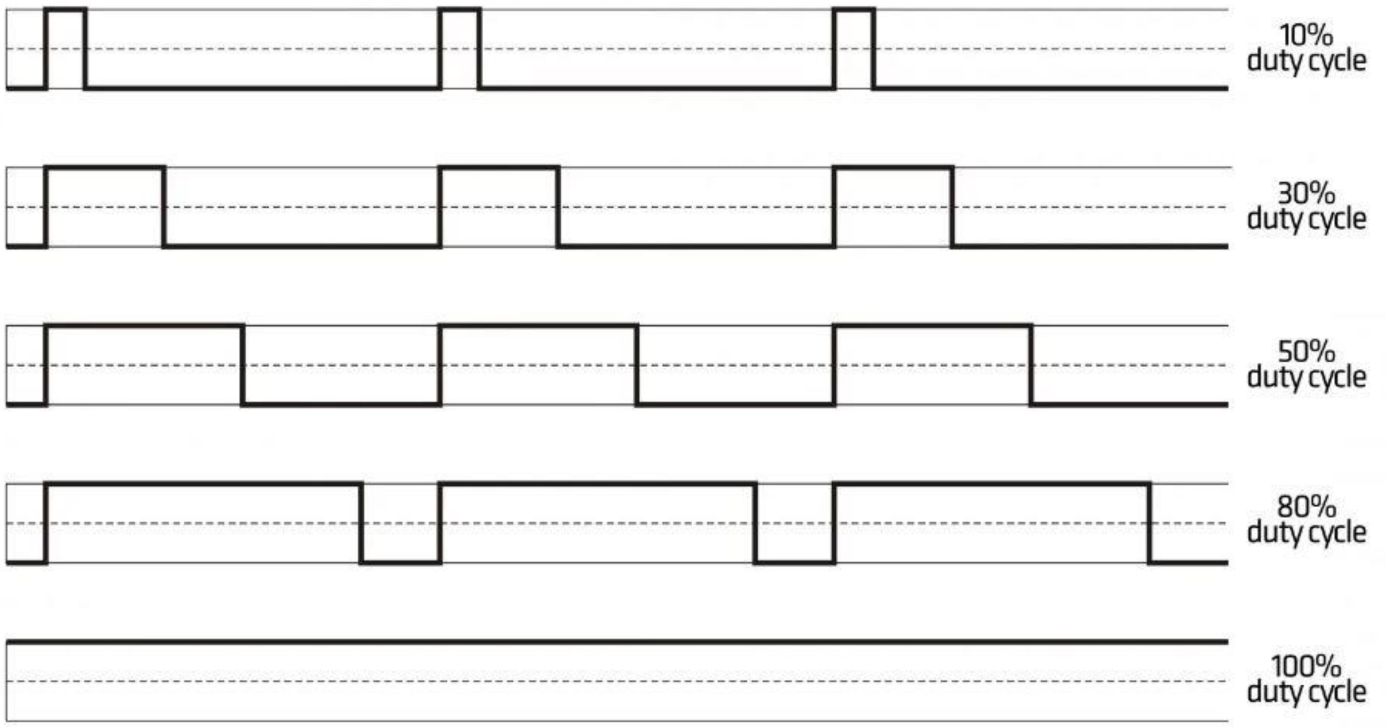
For Thermal Solar Systems with or without externally PWM speed control using Mini Superseal signal cable connection
Via user interface or as factory pre-set it might runs in one of

- 4 Constant Curve (runs without PWM signal)
- 4 PWM solar profile C curves (stops without PWM signal)

Using UPM3 impeller & pump housing
Exceeding benchmark level of the Ecodesign requirements in 2015, EEI ≤ 0.20 EN16297/3

According to measures done with the regulator Steca TR 503, The PWM signal that drives the UPM3 pump has the following characteristics:

- Square signal, voltage 0 / 10V, frequency 250Hz, duty ratio from 0 – pump is stopped - to 100% - full working pump – Rising PWM pump model -



- The pump itself is full time powered by AC230V.
- At startup the duty cycle is 50%. Then this duty cycle increase gradually until 100%.
- When Tsource - Tcumulus reaches deltaTpwm, the PWM duty cycle decreases according to the tendency, the minimum working duty cycle is 25%.

How to generate a 250Hz signal

Several Arduino Uno outputs propose a PWM signal of 490Hz with the code:

```
analogWrite(port, rapport_cyclique)
```

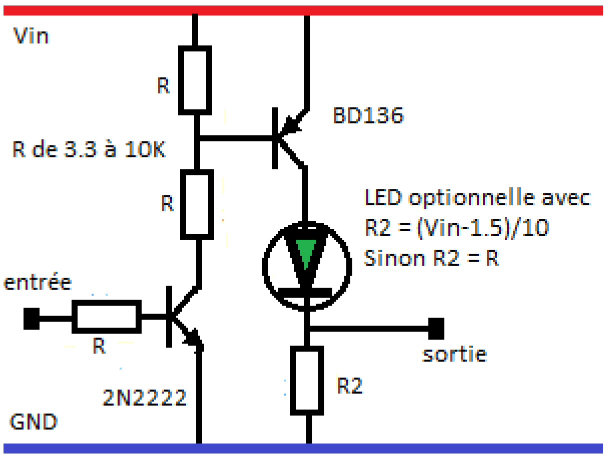
No C code allows to modify this 490Hz frequency to 250Hz. To do this we will play with registries:

- The available frequency is: Fclock / 256 => 16MHz / 256 = 62,5kHz,
- The possible Frequency should be a binary multiple of the available frequency (divisible by 2, 4, 16, etc.)
- Dividing by 256 we get: 62500/256 = 244Hz, almost the require 250Hz.

The code to make a 244Hz PWM signal is:

```
TCCR2A = 0b00100011;  // Fast PWM Mode
TCCR2B = 0b00000110;  // formula => Nbinairy = 62500 / (F * 256)
OCR2B = 128;          // duty cycle of 50% (256/2)
```

How to make ready for use the PWM output

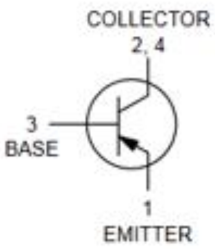


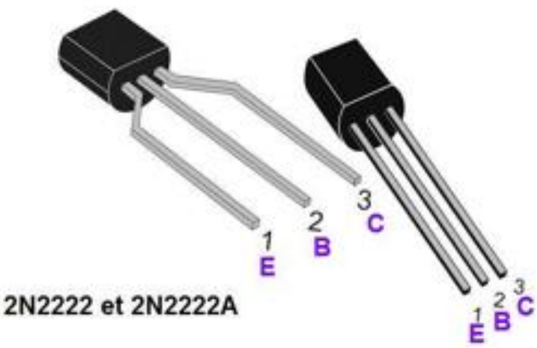
The Pump cannot be driven directly by an output of the Arduino. Current available is low, under 20mA, not protected against any surge or short circuit that could damage the processor ATMega328p. So we use a protection amplifier circuit based on 2 transistors and 4 resistors.

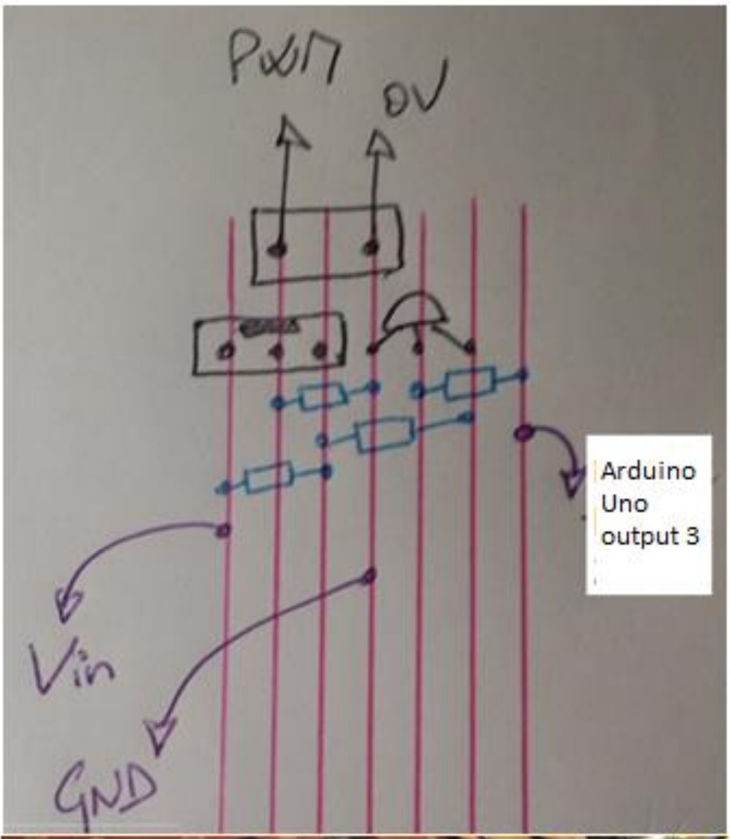
In standby mode both 2 transistors are blocked, no current in the Collector so the voltage at the output (R2) is 0V.

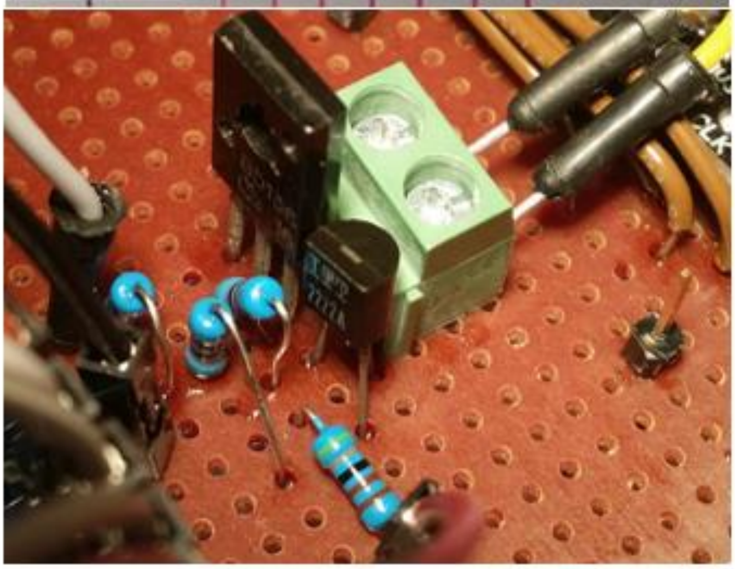
The Arduino Uno logic I signal is the power supply (a little less than 5V) voltage. Both transistors are saturating, the voltage at R2 is Vin. Vin is the 9-12V power supply

BD136









A SSR to drive the power of the pump or the boiler

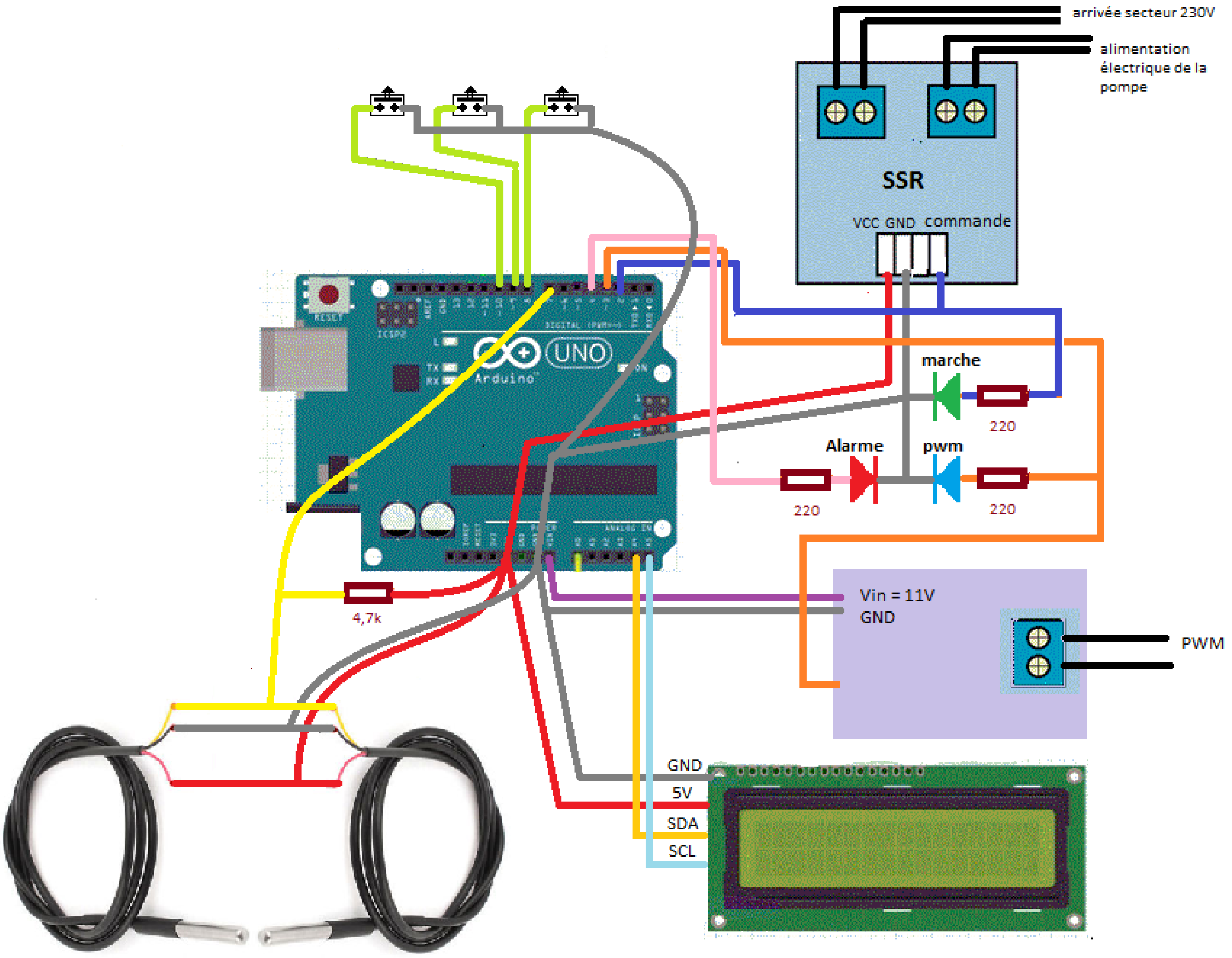
The device must be compatible with non PWM pumps. These pumps are directly driven by their power supply, e.g. AC230V.

Or, in case of use of a PWM pump, the SSR can drive the boiler powering, or to unroll a flap to mask solar panels, or to power any heater system to prevent against frost.

This operation is performed with a non-mechanical relay, a SSR for Solid State Relay. The advantage of the SSR is that a very low power is needed to drive it, also there are no more mechanical constraint.

Of course the SSR must be of the right power dimension.

The complete diagram



Remarks:

- Due to their hardware fixed address, there is no way to change the destination of each temperature sensor. So the temperature sensors must be identified before definitively put in place.
- The small SSR used in this project has an inverted active settle: the SSR is active with a 0 logic input, and is inactive with a I logic input. Therefor most of SSR works in the opposite way. So both logic are proposed: logic I active with PUMP on Arduino Uno's output 3, logic 0 active with PUMP_INV on output 5

The final program

```
/*

Water Heater Thermostat is a device that controls any kind of pump in a solar system
or in a classical boiler

The operating principle is as follows:
- a DS18B20 probe at the heat source compares the temperature with a
second DS18B20 probe placed in hot water tank.
- beyond a deltaTon threshold the pump is started, stops below deltaToff
- pump starts when Tmini freezes
- an alarm system reports the Tmaxi temperature exceedance
- the thresholds - deltaTon, deltaToff, Tmaxi and Tmini - as well as the pump model are
configurable and backed up without power
- support of the PWM control for externally controlled pumps.
  Following the measurements made on the Steca TR 503 regulator, the PWM signal has the
```

```
   the following characteristics:
   - 250Hz with variable cyclic ratio from 0 (stop) to 100% (total pump operation) under 10V,
   - the pump is continuously supplied by the 230V,
   - 50% at start-up, the ratio gradually increases to 100%,
   - when Tcumulus reaches deltaTpwm, the cyclic ratio may decrease down to 25%.

The programme provides for:
- 1 Arduino Uno R3
- 2 DS18B20 temperature sensor
- 1 resistor of 4.7k between data bus and +5V of DS18B20 sensors
- 1 SRG
- 1 16x2 LCD display for Arduino with I2C extension for serial mode,
- 3 push buttons

_________________________________________________________________
|                                                               |
|       author : Philippe de Craene <dcphilippe@yahoo.fr>       |
|           any feedback is welcome                             |
_________________________________________________________________

Hardware pinup:
---------------

numeric output  2   => SSR              = logic I active SSR drive
numeric output  3   => PWM              = drives the 250Hz PWM signal, output 3 required for
Timer2
numeric output  4   => ALARM     = red LED
numeric output  5   => SSR_INV  = logic 0 active SSR drive
numeric output  7   => DS18B20   = 1-Wire data bus for DS18B20
numeric output  8   => ENTRY     = push-button
numeric output  9   => PLUS      = push-button
numeric output 10   => MINUS     = bouton poussoir
analog input   A4   => SDA for display
analog input   A5   => SCL for display


Versions history:
-----------------

version 1    - 18 août 2018       - transformation pour sondes numériques et le LCD avec I2C
version 1.11 - 20 août 2018       - correction  + optimisation
version 1.2  - 28 août 2018       - corrections de bugs
version 1.3  - 25 nov. 2018       - adaptation pour SSR actif à LOW
version 1.4  - 20 mai  2019       - mise à jour des liens pour télécharger les bibliothèques
version 1.41 - 22 aout 2019       - mises à jour diverses
version 2.0  - 5 april 2020       - update for boiler driven by SSR + ALARM management + English
translation

*/

#include <EEPROM.h>
#include <Wire.h>
#include <OneWire.h>              // https://github.com/PaulStoffregen/OneWire
#include <LiquidCrystal_I2C.h>   // https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-
library

// Inputs/outputs:
// A4 => SDA for display
// A5 => SCL for display
const byte ENTRYpin   = 8;    // 3 push-buttons
const byte MINUSpin   = 9;
const byte PLUSpin    = 10;
const byte DS18B20    = 7;    // 1-Wire data for DS18B20
const byte SSRpin     = 2;    // SSR drive + yellow LED
const byte SSR_INVpin = 5;    // inverted SSR drive
const byte PWMpin     = 3;    // 244Hz PWM signal + blue LED
const byte ALARMpin   = 4;    // red LED for ALARM

// default parameters:
int Tmaxi_d      = 85;        // first use maximum temperature = 85°C
int Tmini_d      = -1;        // first use frost temperature = -1°C
int deltaTon_d   =  5;        // first use starting threshold = 5°K
int deltaToff_d  =  2;        // first use stoping threshold = 2°K
int deltaTpwm_d  =  1;        // first use PWM ration management threshold = 1°K
byte pump_model_d = 0;        // pump model, first use = 0
                              // 0: non-PWM pump, 1: rising PWM , 2: falling PWM
byte ssr_rule_d   = 0;        // SSR usage drive, first use =0
                              // 0: pump, 1: boiler, 2: Tmini alarm, 3: Tmaxi alarm

// LCD with I2C declaration :
LiquidCrystal_I2C lcd(0x27, 16, 2);

// specific symbols to display:
byte degree[8] =        { 0b11100,  0b10100,  0b11100,  0b00000,  0b00000,  0b00000,  0b00000,
0b00000};
byte fleche_bas[8] =    { 0b00100,  0b00100,  0b00100,  0b00100,  0b00100,  0b11111,  0b01110,
0b00100};
```

```
byte fleche_haut[8] =   { 0b00100,  0b01110,  0b11111,  0b00100,  0b00100,  0b00100,  0b00100,
0b00100};
byte fleche_stable[8] = { 0b00100,  0b01110,  0b11111,  0b00100,  0b00100,  0b11111,  0b01110,
0b00100};

// DS18B20 declaration:
OneWire sondeDS(DS18B20);

// variables for temperature management:
float Tsource, Tcumulus, memo_Tcumulus;  // past value for tendency calculation
int deltaTon, deltaToff, deltaTpwm;
int Tmaxi, Tmini;
byte tendency = 4;             // tendency 2=increase, 3=decrese, 4=stable

// variables for process management:
byte pump_model;               // 0: non PWM, 1: rising PWM, 2: falling PWM
byte ssr_rule;                 // 0: pump, 1: boiler, 2: Tmini alarm, 3: Tmaxi alarm
byte state, memo_state = 0;    // give the overoll state of the device:
                               // 0: off, 1: on, 2: Tmini, 3: Tmaxi, 4: forced, 5: boiler
bool mforce = LOW;             // flag if pump always ON
bool boiler = LOW;             // flag if a boiler is driven
unsigned long memo_tempo = 0;  // flag for time count

// variables for PWM:
byte ratio = 0;                // initial PWM duty cycle
byte PWM;                      // final PWM (take in account if Rising or Falling PWM)
const byte PWMmini = 25;       // minimum ratio when pup is active

// variables for push-buttons & menus
const char label_pump[] = {'A', 'M', 'G', 'T', 'F', 'S' };
       // label displayed : A: off, M: on, G: Tmini, T: Tmaxi, F: forced, S: boiler
byte ret_push_button, memo_ret_push_button = 0;   // function return value when a push-button is
pushed
String label;
byte window = 0;               // index of window to display
byte windowsTotal = 11;        // total number of windows (including window 0)
byte count_before_timeout = 0;
byte timeout = 60;             // display switch off delay
bool eeprom = false;           // flag a backup needed

//
// SETUP
//____________________________________________________________________________________________

void setup() {

  lcd.begin();

  pinMode (ENTRYpin,   INPUT_PULLUP);
  pinMode (PLUSpin,    INPUT_PULLUP);
  pinMode (MINUSpin,   INPUT_PULLUP);
  pinMode (ALARMpin,   OUTPUT);
  pinMode (SSRpin,     OUTPUT);
  pinMode (SSR_INVpin, OUTPUT);
  pinMode (PWMpin,     OUTPUT);

// Timer2 set for PWM at 244Hz
  TCCR2A = 0b00100011;  // Fast PWM Mode
  TCCR2B = 0b00000110;  // formula => binary = 62500 / (F * 256)
  OCR2B = 0;            // duty ration is 0 at startup

// LCD => special homemade characters are assigned
  lcd.createChar(1, degree );        // =1, etc...
  lcd.createChar(2, fleche_haut );
  lcd.createChar(3, fleche_stable );
  lcd.createChar(4, fleche_bas );

// get data from the EEPROM, type byte. So offset of 50 for negative temperature values :
// defaut EEPROM content is 255 at very first usage.
  if(EEPROM.read(0) < 70) { deltaTon = EEPROM.read(0) - 50; }
  else { EEPROM.write(0, deltaTon_d + 50); deltaTon = deltaTon_d; }
  if(EEPROM.read(1) < 60) { deltaToff = EEPROM.read(1) - 50; }
  else { EEPROM.write(1, deltaToff_d + 50); deltaToff = deltaToff_d; }
  if(EEPROM.read(2) < 60) { deltaTpwm = EEPROM.read(2) - 50; }
  else { EEPROM.write(2, deltaTpwm_d + 50); deltaTpwm = deltaTpwm_d; }
  if(EEPROM.read(3) < 150) { Tmaxi = EEPROM.read(3) - 50; }
  else { EEPROM.write(3, Tmaxi_d + 50); Tmaxi = Tmaxi_d; }
  if(EEPROM.read(4) < 60) { Tmini = EEPROM.read(4) - 50; }
  else { EEPROM.write(4, Tmini_d + 50); Tmini = Tmini_d; }
  if(EEPROM.read(5) < 3) { pump_model = EEPROM.read(5); }
  else { EEPROM.write(5, pump_model_d); pump_model = pump_model_d; }
  if(EEPROM.read(6) < 4) { ssr_rule = EEPROM.read(6); }
  else { EEPROM.write(6, ssr_rule_d); ssr_rule = ssr_rule_d; }

// console and LCD initialisation
  Serial.begin(9600);
  Serial.println("ready ...");
```

```
  Serial.println ();
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Thermostat by");
  lcd.setCursor(0, 1);
  lcd.print("PhilippeDC");
  delay(500);
}      // end of setup

//
// LOOP
//______________________________________________________________________________

void loop() {

  unsigned long tempo = millis()/800;       // time count a littke less than one seconds
  if( memo_tempo == tempo ) return;         // everything after is done every second
  memo_tempo = tempo;
  static unsigned int heatTimeOut = 0;      // counter for boiler offline alarm

// STEP 1: get the temperature values
//______________________________________________________________________________

// as it is needed 1.6s to get the temperature for both sensors, this is done
// only when we are out of the parameters setting
  if(window== 0) {
     GetTemperature(&Tsource, true);         // true only for the first reading
     GetTemperature(&Tcumulus, false); }

// STEP 2: compare temperature with the thresholds
//______________________________________________________________________________

// deltaTonn is reached : the pump is starting
  memo_state = state;                        // remaind the past state
  if( Tsource-Tcumulus-deltaTon >= 0 ) {
     state = 1;
     if(memo_state == 0) ratio = 50; }        // duty cycle 50% when pump is starting

// the pump is running, PWM increases until 100%
  if((state==1)&&(Tsource-Tcumulus-deltaToff-deltaTpwm >0)&&(ratio < 100))
    ratio++;

// deltaTpwm is reached so PWM is starting to decrease
  if((state==1)&&(Tsource-Tcumulus-deltaToff-deltaTpwm <=0)&&(tendency==2)&&(ratio > PWMmini))
    ratio--;

// deltaToff is reached so the pump stops
  if(Tsource-Tcumulus-deltaToff <=0) {
    state = 0;
    ratio = 0; }

// Tmini reached : the pump starts + ALARM
  if(Tsource <= Tmini) {
    state = 2;
    ratio = 50;
    Alarm(3); }                               // red LED blinks 3 times/second

// pump is always on
  if(mforce == HIGH) {
    state = 4;
    ratio = 50;
    Alarm(1); }                              // red LED blinks 1 time/second

// drive a boiler
  if(ssr_rule == 1) {
    state = 5;                               // start the boiler
    if(Tsource < 40) {                       // check the heat of the source
      PWM = 0;                               // pump stop if no heat
      heatTimeOut++;                         // count seconds
      if( heatTimeOut > 375 ) Alarm(2); }    // after 375 count = 10 minutes
    else heatTimeOut = 0; }

// Tmaxi reached : the pump is stopped + ALARM
  if(Tcumulus >= Tmaxi || Tsource > Tmaxi ) {
    state = 3;
    ratio = 0;
    Alarm(5); }                               // red LED blinks 5 times/second

//  Step 3: outputs management
//______________________________________________________________________________

// PWM management:
// pump_model == 0 for NON-PWM pump
// pump_model == 1 for Rising PWM pump
// pump_model == 2 for Falling PWM pump

  if(pump_model == 1)      PWM = map(ratio, 0, 100, 0, 255);
```

```
  else if(pump_model == 2) PWM = map(ratio, 0, 100, 255, 0);
  else                     PWM = 0;

  analogWrite(PWMpin, PWM);              // commande du PWM / mli

// SSR management
// ssr_rule == 0 to drive pump (require if pump_model =0)
// ssr_rule == 1 to drive the boiler,
// ssr_rule == 2 to drive Tmini alarm,
// ssr_rule == 3 to drive Tmaxi alarm
  if( ssr_rule == 0 ) {                  // case NON-PWM pump
    if((state == 1)||(state==2)||(state==4)) {
      digitalWrite(SSRpin, HIGH); digitalWrite(SSR_INVpin, LOW); }
    else {
      digitalWrite(SSRpin, LOW); digitalWrite(SSR_INVpin, HIGH); }
  }    // end of ssr_rule == 0

  else if( ssr_rule == 1 ) {             // case boiler
    if( state == 5 ) {
      digitalWrite(SSRpin, HIGH); digitalWrite(SSR_INVpin, LOW); }
    else {
      digitalWrite(SSRpin, LOW); digitalWrite(SSR_INVpin, HIGH); }
  }    // end of ssr_rule == 1

  else if( ssr_rule == 2 ) {             // case alarm Tmini
    if( state == 2 ) {
      digitalWrite(SSRpin, HIGH); digitalWrite(SSR_INVpin, LOW); }
    else {
      digitalWrite(SSRpin, LOW); digitalWrite(SSR_INVpin, HIGH); }
  }    // end of ssr_rule == 2

  else if( ssr_rule == 3 ) {             // case alarm Tmaxi
    if( state == 3 ) {
      digitalWrite(SSRpin, HIGH); digitalWrite(SSR_INVpin, LOW); }
    else {
      digitalWrite(SSRpin, LOW); digitalWrite(SSR_INVpin, HIGH); }
  }    // end of ssr_rule == 2

// Step 4 : tendency calculation
//___________________________________________________________________________________________

  static unsigned int cycleCounter = 0;     // counter for tendencies
  if(++cycleCounter > 5) {
    cycleCounter = 0;
    if( Tcumulus >  memo_Tcumulus ) tendency = 2;    // draw rising arrow on display
    if( Tcumulus == memo_Tcumulus ) tendency = 3;    // draw stable arrow on display
    if( Tcumulus <  memo_Tcumulus ) tendency = 4;    // draw falling arrow on display
    memo_Tcumulus = Tcumulus;                        // keep past measure
  }    // end of test cycleCounter

// Step 5: display & parameters management
//___________________________________________________________________________________________

// check push-buttons & display ON / OFF
//___________________________________________________________________________________________

  memo_ret_push_button = ret_push_button; // memorize past value
  ret_push_button = push_button();        // reading push-button status
                                          // 0: nothing, 1: ENTRY, 2: PLUS, 3: MINUS
  count_before_timeout++;                 // timeout before switch off the display
  if( count_before_timeout > timeout ) {  // timeout reached
      window = 0;                         // return to first display
      lcd.clear();
      lcd.noBacklight();                  // light off display
  }

  if((memo_ret_push_button == 1)&&(ret_push_button == 1)) next_window();

// regular case: no push-buttonactivity
//___________________________________________________________________________________________

  if( window== 0 ) {
    lcd.setCursor(0, 0);
    lcd.print("SOURCE:");
    if( Tsource < 100 ) lcd.print(" ");
    lcd.print(String(Tsource, 1));
    lcd.write(1);                             // display character '°'
    lcd.print("C ");
    lcd.setCursor(15, 0);
    lcd.write(label_pump[state]);
    lcd.setCursor(0, 1);
    lcd.print("BALLON: ");
    lcd.print(String(Tcumulus, 1));
    lcd.write(1);
    lcd.print("C ");
    lcd.setCursor(15, 1);
    lcd.write(tendency);                      // dram the arrow
```

```
  }    // end of windows == 0

// ENTRY button pushed
//____________________________________________________________________________________________

  if( window == 1 ) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("LISTE DES");
    lcd.setCursor(0, 1);
    lcd.print("PARAMETRES :");
    delay(800);
    next_window();
  }    // end of windows == 1

// parameters setting

  if( window == 2 ) {
    if(ret_push_button == 2) {
      deltaTon++;
      deltaTon = constrain(deltaTon, 0, 9); }    // limit value between 0 to 9
    if(ret_push_button == 3) {
      deltaTon--;
      deltaTon = constrain(deltaTon, deltaToff, 9); }
    lcd.setCursor(0, 0);
    lcd.print("DELTA T MARCHE :");
    lcd.setCursor(0, 1);
    lcd.print(deltaTon);
    lcd.write(1);
    lcd.print("K  ");
  }    // end of window == 2

  if( window == 3 ) {
    if(ret_push_button == 2) {
      deltaToff++;
      deltaToff = constrain(deltaToff, 0, deltaTon); }
    if(ret_push_button == 3) {
      deltaToff--;
      deltaToff = constrain(deltaToff, 0, 9); }
    lcd.setCursor(0, 0);
    lcd.print("DELTA T ARRET :");
    lcd.setCursor(0, 1);
    lcd.print(deltaToff);
    lcd.write(1);
    lcd.print("K");
  }    // end of window == 3

  if( window == 4 ) {
    if(ret_push_button == 2) {
      deltaTpwm++;
      deltaTpwm = constrain(deltaTpwm, 0, 9); }
    if(ret_push_button == 3) {
      deltaTpwm--;
      deltaTpwm = constrain(deltaTpwm, 0, 9); }
    lcd.setCursor(0, 0);
    lcd.print("DELTA T PWM :");
    lcd.setCursor(0, 1);
    lcd.print(deltaTpwm);
    lcd.write(1);
    lcd.print("K  ");
  }    // end of window == 4

  if( window == 5 ) {
    if(ret_push_button == 2) {
      Tmaxi++;
      Tmaxi = constrain(Tmaxi, 40, 100); }
    if(ret_push_button == 3) {
      Tmaxi--;
      Tmaxi = constrain(Tmaxi, 40, 100); }
    lcd.setCursor(0, 0);
    lcd.print("T MAXI ALARME :");
    lcd.setCursor(0, 1);
    lcd.print(Tmaxi);
    lcd.write(1);
    lcd.print("C  ");
  }    // end of windows == 5

  if( window == 6 ) {
    if(ret_push_button == 2) {
      Tmini++;
      Tmini = constrain(Tmini, -9, 9); }
    if(ret_push_button == 3) {
      Tmini--;
      Tmini = constrain(Tmini, -9, 9); }
    lcd.setCursor(0, 0);
    lcd.print("T HORS GEL :");
    lcd.setCursor(0, 1);
```

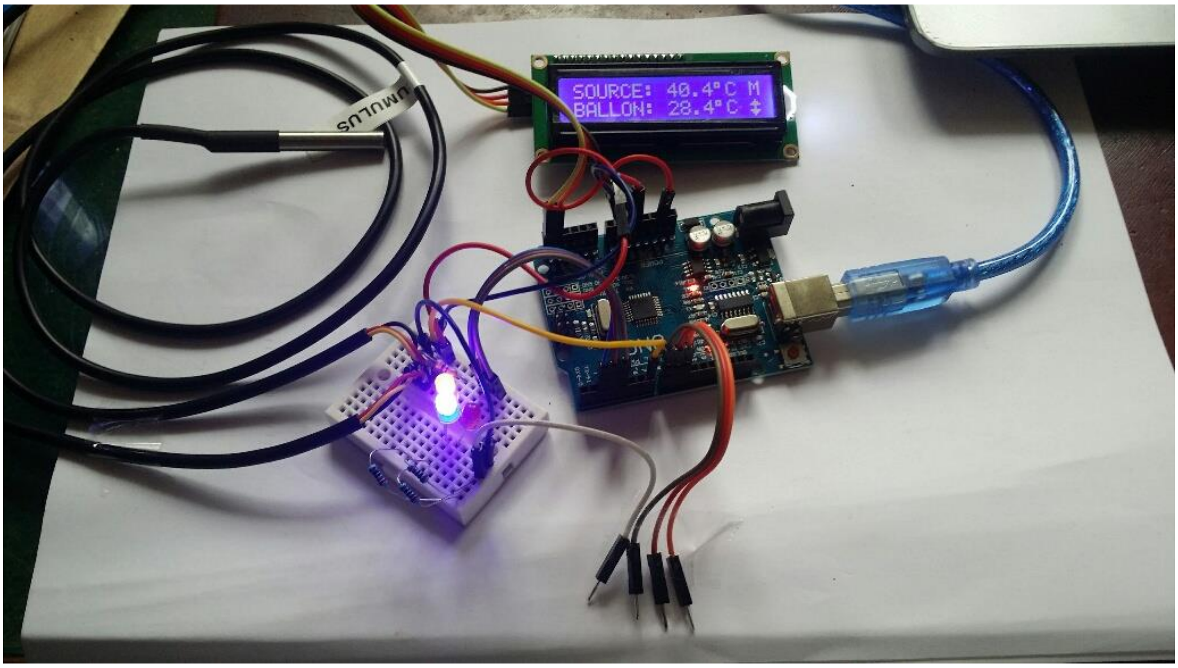
```
    lcd.print(Tmini);
    lcd.write(1);
    lcd.print("C  ");
  }    // end of window == 6

  if( window == 7 ) {
    if(ret_push_button == 2) pump_model = (pump_model +1)%3;
    if(ret_push_button == 3) pump_model = (pump_model -1)%3;
    if(pump_model == 0)      label = "NON-PWM        ";
    else if(pump_model == 1) label = "PWM CROISSANT  ";
    else if(pump_model == 2) label = "PWM DECROISSANT";
    lcd.setCursor(0, 0);
    lcd.print("MODELE DE POMPE:");
    lcd.setCursor(0, 1);
    lcd.print(label);
  }    // end of window == 7

  if( window == 8 ) {
    if( pump_model == 0 ) label = "POMPE NON-PWM   ";
    else {
      if(ret_push_button == 2) ssr_rule = (ssr_rule +1)%4;
      if(ret_push_button == 3) ssr_rule = (ssr_rule -1)%4;
      if(ssr_rule == 0)      label = "POMPE NON-PWM   ";
      else if(ssr_rule == 1) label = "CHAUDIERE       ";
      else if(ssr_rule == 2) label = "PROTECTION GEL  ";
      else if(ssr_rule == 3) label = "PROT. EBULLITION";
    }
    lcd.setCursor(0, 0);
    lcd.print("USAGE COMMANDE :");
    lcd.setCursor(0, 1);
    lcd.print(label);
  }    // end of window == 8

  if( window == 9 ) {
    lcd.setCursor(0, 0);
    lcd.print("REINITIALISATION");
    lcd.setCursor(0, 1);
    lcd.print("POUR VALIDER : +");
    if(ret_push_button == 2) {
       lcd.setCursor(0, 0);
       lcd.print("REINITIALISATION");
       lcd.setCursor(0, 1);
       lcd.print("FAITE           ");
       deltaTon = deltaTon_d;    // default value to all parameters
       deltaToff = deltaToff_d;
       deltaTpwm = deltaTpwm_d;
       Tmaxi = Tmaxi_d;
       Tmini = Tmini_d;
       pump_model = pump_model_d;
       ssr_rule = ssr_rule_d;
       mforce = LOW;             // stop always ON capability
       window= 0; }
  }    // end of windows == 9

  if(  window == 10 ) {
    lcd.setCursor(0, 0);
    lcd.print("MARCHE FORCEE ? ");
    lcd.setCursor(0, 1);
    lcd.print("OUI= + / NON= - ");
    if(ret_push_button == 2) {
      mforce = HIGH;
      lcd.setCursor(0, 0);
      lcd.print("MARCHE FORCEE   ");
      lcd.setCursor(0, 1);
      lcd.print("CONFIRMEE       ");
      window = 0; }
    if(ret_push_button == 3) {
      mforce = LOW;
      lcd.setCursor(0, 0);
      lcd.print("MARCHE FORCEE   ");
      lcd.setCursor(0, 1);
      lcd.print("ARRETEE         ");
      window = 0; }
  }    // end of windows == 10

// EEPROM backup
//________________________________________________________________________________________________

  if( eeprom = true ) {                  // EEPROM backup only if needed
    EEPROM.update(0, deltaTon + 50);
    EEPROM.update(1, deltaToff + 50);
    EEPROM.update(2, deltaTpwm + 50);
    EEPROM.update(3, Tmaxi + 50);
    EEPROM.update(4, Tmini + 50);
    EEPROM.update(5, pump_model);
    EEPROM.update(6, ssr_rule);
  }
```

```
// for debugging
//________________________________________________________________________________________

   Serial.print("ssr_rule: "); Serial.print(ssr_rule);
   Serial.print("  state: "); Serial.print(state);
   Serial.print("  ratio: "); Serial.print(ratio);
   Serial.print("  PWM: "); Serial.print(PWM);
   Serial.print("  Ts: "); Serial.print(Tsource);
   Serial.print("  Tc: "); Serial.print(Tcumulus);
   Serial.print("  Ts-Tc-deltaToff: "); Serial.print(Tsource-Tcumulus-deltaToff);
   Serial.println();

}    // end of loop

//=====================================================================================
// list of functions
//=====================================================================================

//
// next_window() : procedure to change the window display
//________________________________________________________________________________________

void next_window() {
  window = (window+1) % windowsTotal;    // next window
  lcd.clear();
}      // end of next_window()

//
// push_button() : return value depending of the state of the 3 push-buttons
//________________________________________________________________________________________

byte push_button() {
  if ( digitalRead(ENTRYpin) == LOW ) {
    count_before_timeout = 0;         // reset the timeout counter
    lcd.backlight();                  // switch on display
    lcd.clear();
    return 1;
  }
  if ( digitalRead(PLUSpin) == LOW ) {
    count_before_timeout = 0;
    lcd.backlight();
    eeprom = true;
    return 2;
  }
  if ( digitalRead(MINUSpin) == LOW ) {
    count_before_timeout = 0;
    lcd.backlight();
    eeprom = true;
    return 3;
  }
  return 0;
}     // end of push_button()

//
// GetTemperature() : reading measures of DS18B20
//________________________________________________________________________________________

byte GetTemperature(float *temperature, byte reset_search) {
  byte data[9], addr[8];           // data[] : data read from scratchpad
                                   // addr[] : Address of detected 1-Wire device

// reset the list of adresses for the first sensor
// char argument = "true" or "false"
  if(reset_search) sondeDS.reset_search();
  sondeDS.search(addr);
  sondeDS.reset();
  sondeDS.select(addr);
  sondeDS.write(0x44, 1);          // getting the temperature takes as long as 1/2 second for
each sensor
  delay(800);                      // required, see above comment
  sondeDS.reset();
  sondeDS.select(addr);
  sondeDS.write(0xBE);             // send request from scratchpad

// the scratchpad content makes 9 bytes:
  for (byte i = 0; i < 9; i++) { data[i] = sondeDS.read(); }

// miraculous formula to get the temperature in °C with a 12 bits resolution, accurancy of
0,06°K:
// it is possible to modify this formula to get °F
  *temperature = ((data[1] << 8) | data[0]) * 0.0625;
}      // end of getTemperature()

//
// Alarm() red LED lighting management
//________________________________________________________________________________________
```

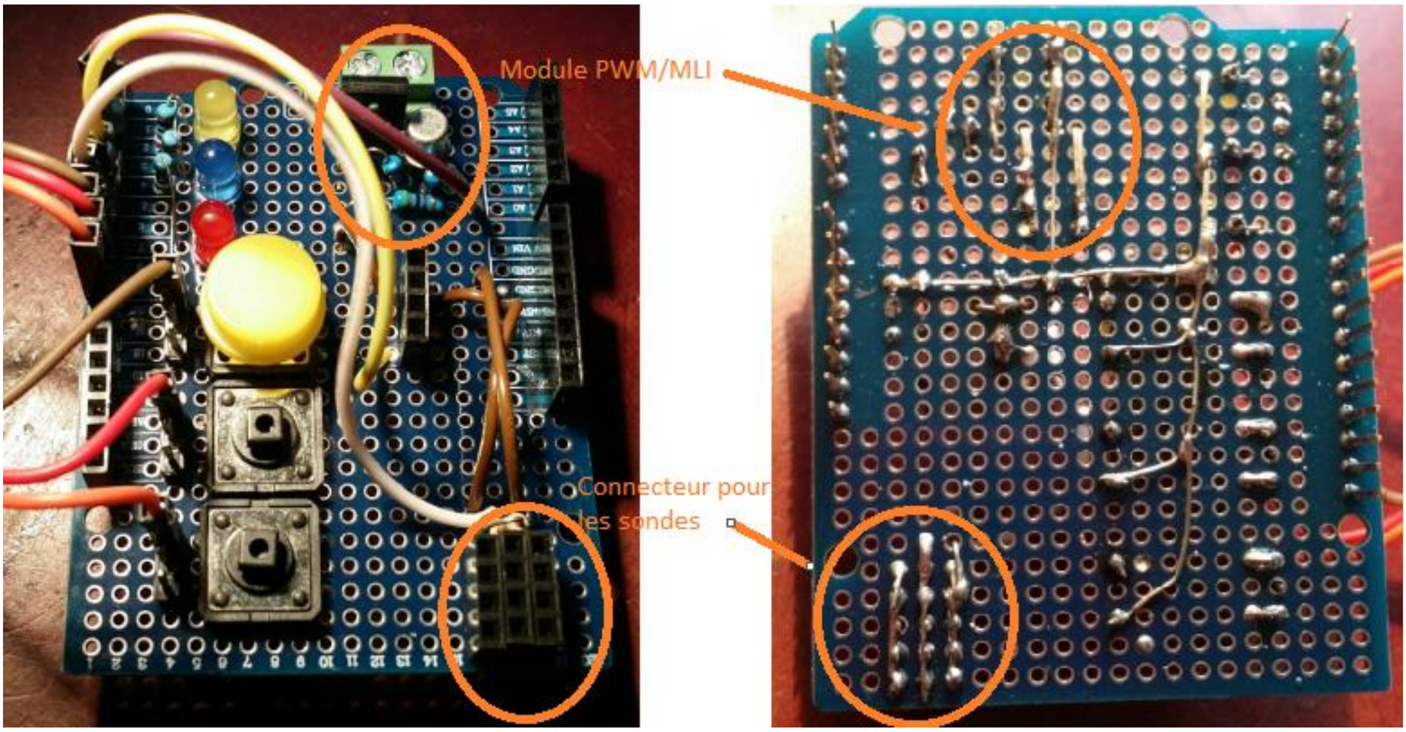
```
void Alarm(byte c) {
  for( byte i=0; i<c; i++ ) {
    digitalWrite(ALARMpin, HIGH); delay(10);
    digitalWrite(ALARMpin, LOW); delay(200);
  }
}      // end of Alarm()
```

Illustrations

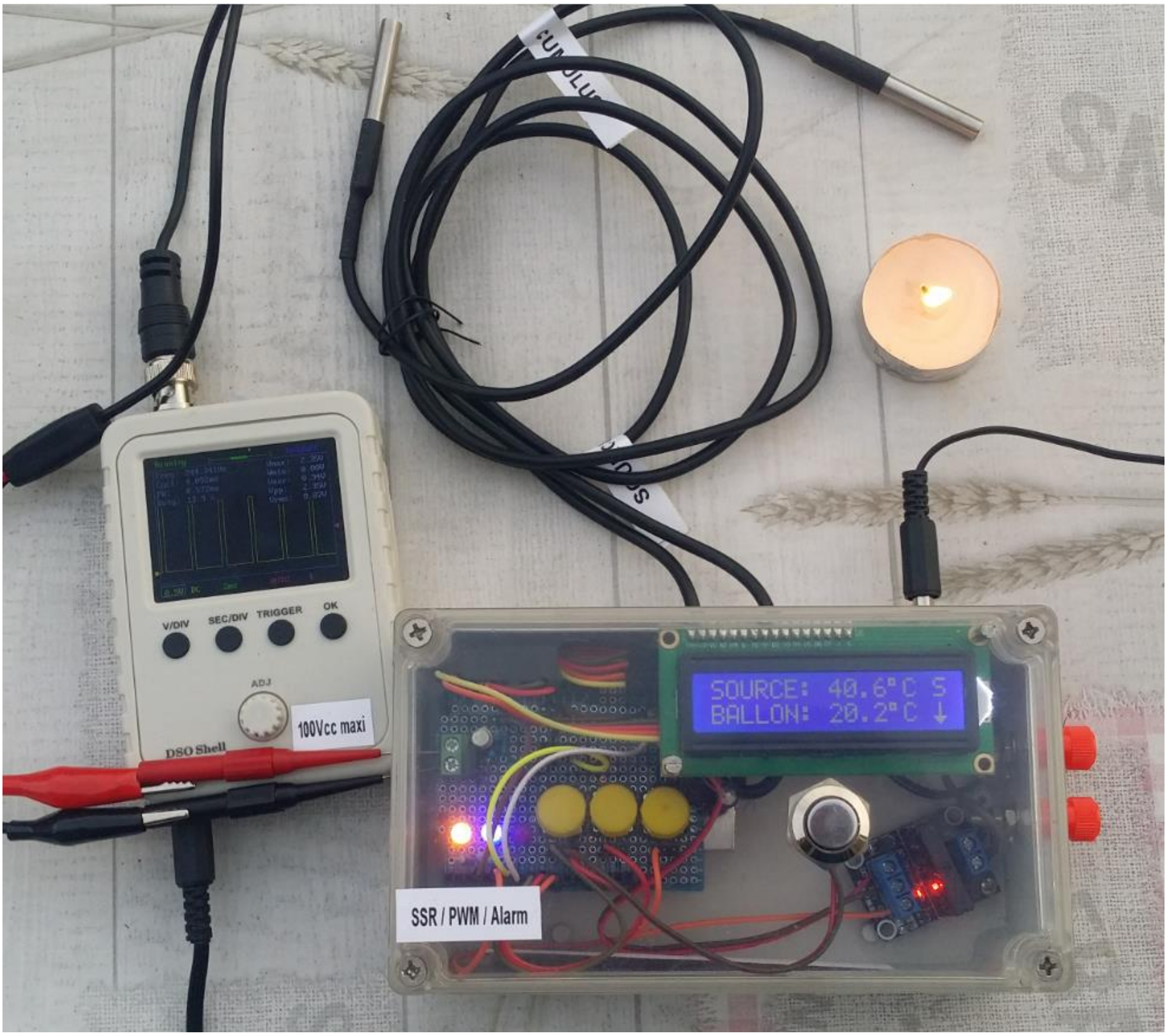
During tests. The 4 free wires are for the 3 push-buttons:



Example of shield implantation:



During some tests, once in box:



Operation manual

Sequence of operations

At startup a look is taken to the EEPROM in order to get the parameter. At the very first startup it is all default values that are stored into the EEPROM.

Then, sequentially:

1. Get the temperature measures : *Tsource, Tcumulus*
2. Compare the measures with the consigns : *deltaTon, deltaToff, deltaTpwm, Tmini, Tmaxi*
3. Drive the SSR start/stop and the PWM ratio depending of *state, ssr_rule*
4. Calculate the tendency of the cumulus heating or cooling (for display only)
5. Read push-buttons state and update the LCD display

Once the “Entry” button is pushed for more or less 2 seconds, the display will pass in review all parameters. ENTRY to enter in parameter and forward to the following screen, PLUS and MINUS to increase/decrease values. Updates are immediately taken in account:

1- Display/update of *deltaTon* : the *Tsource – Tcumulus* starting the pump threshold value
2- Display/update of *deltaToff* : the *Tsource – Tcumulus* stopping the pump threshold value
3- Display/update of *deltaTpwm* : the threshold starting the PWM ratio to decrease
4- Display/update of *Tmini* : the minimum value before a frost security procedure
5- Display/update of *Tmaxi* : the maximum value before a boiling security procedure
6- Display/update of *pump_model* that can be ‘NON PWM’, ‘rising PWM’ or ‘falling PWM’
7- Display/ update of *ssr_rule* that can be ‘DRIVE A PUMP’, ‘DRIVE A BOILER’, ‘DRIVE Tmini ALARM’, ‘DRIVE Tmaxi ALARM’
8- Allow the reset for all above parameters
9- Force the pump to run

Capabilities of the outputs

Two outputs are available:

- The SSR: it is a relay switch ON or OFF.
- The PWM that is the 244Hz variable ratio signal of ~10V especially for high efficiency PWM driven pumps.

Choice of ssr_value:	Events:	Tmini << Tsource < Tcumulus << Tmaxi	Tmini << Tsouce # Tcumulus << Tmaxi	Tsource < Tmini	Tsource or Tcumulus > Tmaxi	Forced	Pump model
DRIVE A PUMP*		SSR is ON	SSR is OFF	SSR is ON	SSR is OFF	SSR is ON	Non PWM driven by SSR
DRIVE A BOILER		SSR is ON**	SSR is OFF	SSR is ON***	SSR is OFF	SSR is ON	Driven by PWM
DRIVE Tmini ALARM		SSR is OFF	SSR is OFF	SSR is ON	SSR is OFF	SSR is ON if needed	Driven by PWM
DRIVE Tmaxi ALARM		SSR is OFF	SSR is OFF	SSR is OFF	SSR is ON***	SSR is ON if needed	Driven by PWM

(*) if ‘DRIVE A PUMP’ is the ssr_rule, then no PWM signal as is not applicable in fact. For any else choice of ssr_rule the pump must be either rising PWM or falling PWM.
(**) if Tsouce does not increase up to 40°C after 10 minutes the alarm display (red LED) blinks
(***) each alarm has its own blinking sequence

Algorithm

The following is done every 1.6 seconds:

Tsource > Tcumulus + deltaTon

The pump is ON
PWM starts at 50% then “slowly” increases to 100% (for rising model pump, otherwise 0% for falling model’s

Tsource = Tcumulus + deltaToff +deltaTpwm

The pump is ON
Following the tendency if still in increasing mode, the ratio “slowly” decreases until the 25% limit (for rising model pump, otherwise 75% for falling model’s

Tsource < Tcumulus + deltaToff	The pump is OFF
Tcumulus > Tmaxi ou Tsource > Tmaxi	The pump is OFF The red LED is blinking 5 times every 1.6 seconds If ssr_rule in DRIVE Tmaxi ALARM mode, the SSR is ON
Tsource < Tmini	The pump is ON The PWM is fixed to 50% The red LED is blinking 3 times every 1.6 seconds if ssr_rule in DRIVE Tmini ALARM mode, the SSR is ON
Forced ON	The pump is ON The PWM is fixed to 50% The red LED is blinking 1 time every 1.6 seconds

Display legend

The letters displayed can be changed line 124 of the code. They are displayed on the top right of the LCD:

- A for pump OFF
- M for pump ON
- G for frost (Tmini)
- T for overheat (Tmaxi)
- F for forced ON
- S for the boiler in use

On the bottom right of the LCD display:

- Falling arrow when the cumulus temperature is decreasing
- Rising arrow when the cumulus temperature is increasing
- Equal arrow when the cumulus temperature reminds the same

The LCD display automatically switch off after less than 2 minutes. It can switch on back by pushing the ENTRY button.

3 LEDs show the following events:

Yellow LED ON = SSR ON
Yellow LED OFF = SSR OFF

Blue/green LED ON = rising PWM pump ON or falling PWM pump OFF
Blue/green LED OFF = rising PWM pump OFF or falling PWM pump ON

Red LED blinking = according to the number of blinking the alarm is:

Number of blinks per 1.6 seconds:	1	2	3	5
Event:	Forced: pump always ON	Tsouce < 40°C after 10 minutes	Tmini alarm	Tmaxi alarm

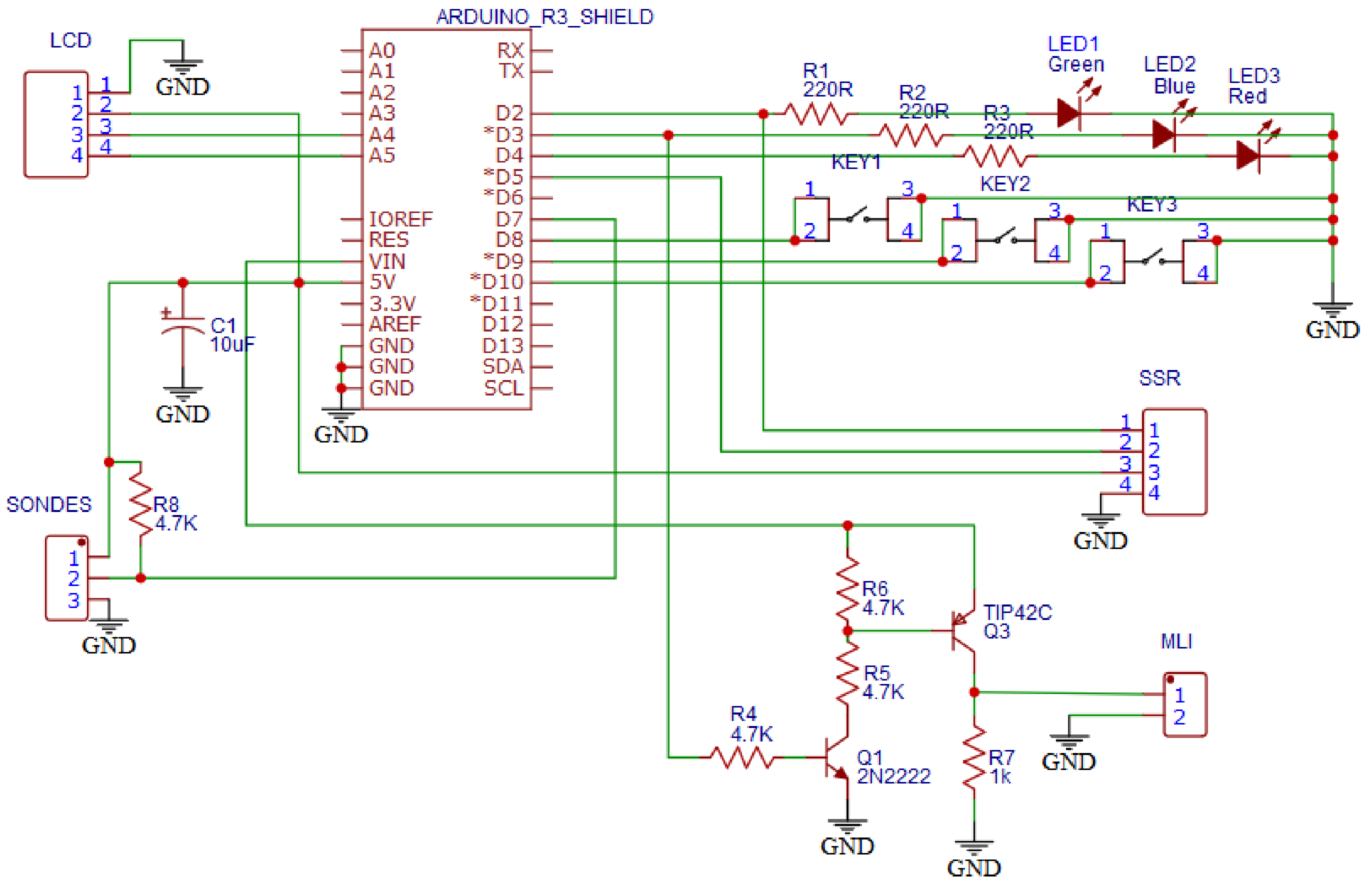
Putting all in PCB

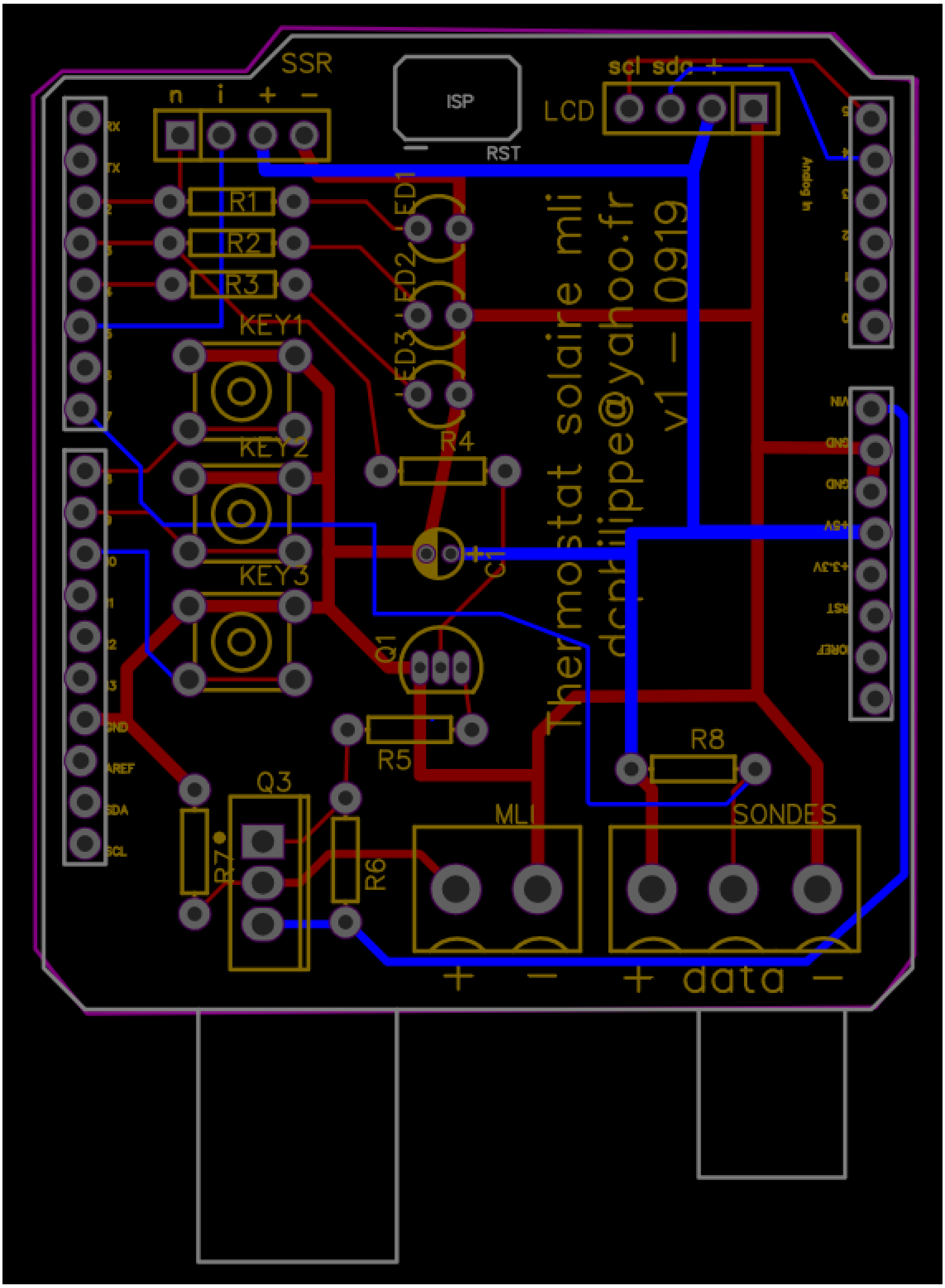
With the success of this project, a PCB has been created in order to facilitate the construction. It has been made to directly settle on an Arduino Uno R3.

For the PCB I use the the EasyEDA's services: https://easyeda.com

First a circuit diagram must be drawn. Then a PCB is proposed, we are free to place components wherever we want… The PCB is like a shield to connect to the Arduino Uno. The SSR stays out of the circuit, so that no dangerous high voltage can be found on the PCB.

Below is the diagram made on EasyEDA:




SSR
n i + −
ISP
RST
LCD
scl sda + −
R1
R2
R3
KEY1
KEY2
KEY3
LED1
LED2
LED3
R4
C1
Q1
R5
Q3
R7
R6
R8
MLI
SONDES
+ −
+ data −
Thermostat solaire mli
dcphilippe@yahoo.fr
v1 – 0919