Fabriquer un thermostat simple de chauffe-eau

Autheur : Association P'TITWATT

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

Version du document : 1.2
Version du programme : 1.4

Date : Août 2018 – mai 2019

Il existe des centaines de thermostats de réalisation artisanale plus ou moins sophistiqués. Celui-ci devait répondre aux exigences suivantes :

- Être construit autour d'un module Arduino Uno R3,
- Utiliser deux sondes de type DS18B20, qui sont très bon marché et offrent des mesures précises,
- Savoir gérer la nouvelle génération de circulateurs, c'est-à-dire :
    - Soit circulateurs autonomes à régulation interne par mesure de pression,
    - Soit à commande pwm (appelé mli « in french ») externe, sachant qu'il en existe 2 types :
        - Les pompes à mli croissant,
        - Les pompes à mli décroissant.
- Être paramétrable : seuil de mise en marche, seuil d'arrêt, température d'alarme, température hors gel, mise en marche forcée, réinitialisation aux valeurs par défaut,
- Savoir gérer une coupure d'électricité en récupérant les paramètres au redémarrage,
- Être à la portée d'un bricoleur qui sait utiliser un fer à souder.

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

1- Il s'agit d'une part de mesurer la température d'une source qui peut être un panneau solaire thermique, ou une chaudière, et la température du cumulus.

En fonction du résultat de ces mesures le circulateur – la pompe - sera – ou pas – mis en route :

- Si Tsouce < Tmini (température de hors gel) => pompe mise en route
- Si Tcumulus > Tmaxi (température maximum) => pompe arrêtée + alarme (simple LED dans notre cas)
- Si Tsource – Tcumulus > detlaTon => la pompe se met en route
- Si Tsource – Tcumulus < deltaToff => la pompe s'arrête

2- D'autre part il s'agit de mesurer la tendance à la chauffe ou au refroidissement du cumulus :
   - Lorsque la pompe est en marche et que le cumulus monte en température, à partir d'un seuil le signal pwm (ou mli) est généré afin de réduire la vitesse de la pompe.
   - Au contraire, lorsque le cumulus se refroidit, le signal pwm (ou mli) fait en sorte d'augmenter la vitesse de la pompe.

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

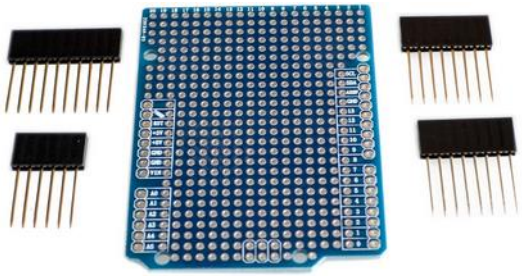
2- Un afficheur LCD 16 caractères sur 2 ligne avec extension I2C : https://www.aliexpress.com/item/1PCS-LCD-module-Blue-screen-IIC-I2C-1602-for-arduino-1602-LCD-UNO-r3-mega2560/32763867041.html?spm=2114.search0104.3.74.52b74888Z22xLA&ws_ab_test=searchweb0_0,searchweb201602_4_10065_10068_10547_319_10892_317_10696_10084_453_454_10083_10618_10304_10307_10820_10821_537_10302_536_10843_10059_10884_10887_321_322_10103,searchweb201603_6,ppcSwitch_0&algo_expid=81c1c3ab-1c44-4f8f-9d52-b0d131b18db6-10&algo_pvid=81c1c3ab-1c44-4f8f-9d52-b0d131b18db6

3- 2 sondes DS18B20 :

https://www.aliexpress.com/item/DS18B20-Stainless-steel-package-1-meters-waterproof-DS18b20-temperature-probe-temperature-sensor-18B20-in-stock-high/753249738.html?spm=2114.search0104.3.22.478a59a1pVbs2o&ws_ab_test=searchweb0_0,searchweb201602_4_10065_10130_10068_10547_319_10892_317_10696_10190_453_10084_454_10083_10618_10307_10820_10821_10303_537_10302_536_10059_10884_10887_321_322_10103,searchweb201603_6,ppcSwitch_0&algo_expid=b8d1e184-960d-4a9f-9356-ce1bf0f064d7-3&algo_pvid=b8d1e184-960d-4a9f-9356-ce1bf0f064d7

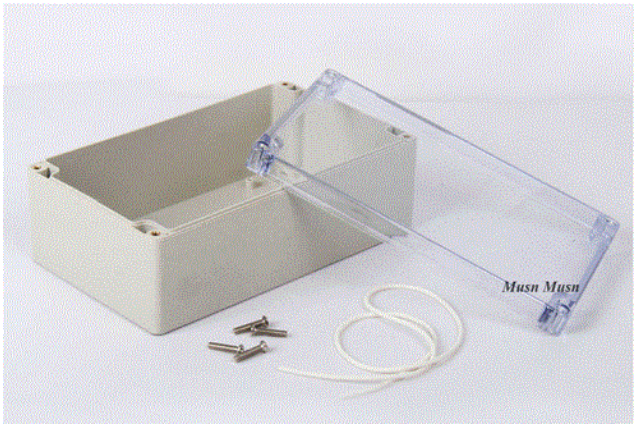
4- Un module de commande SSR ou à triac : https://fr.aliexpress.com/item/Smart-Electronics-1-2-4-Channel-5V-DC-Relay-Module-Solid-State-Low-Level-G3MB-202P/32727486514.html?spm=a2g0s.9042311.0.0.107d6c37yGuDzP
(Ce modèle est est à commande inversée, c'est-à-dire actif avec une commande à 0V, il faudra donc en tenir compte)

5- Une platine à montage de prototypes à souder : https://fr.aliexpress.com/item/Prototype-PCB-Carte-D-extension-Pour-Arduino-ATMEGA328P-UNO-R3-Bouclier-FR-4-Fiber-PCB-Planche/32846515603.html?spm=a2g0s.9042311.0.0.3c126c37BCQnQ0



6- Une alimentation entre 9 et 11V pour l'Arduino ; pour l'arduino 6V aurait été suffisant, mais le signal pwm doit être de 10V environ…

7- Des résistances de 4,7k, 3 résistances de 220Ω, une LED rouge, une LED bleue ou verte, 3 boutons poussoirs, 1 bornier à vis, du câble Dupont (bof bof c'est plein de mauvais contacts, rien de vaut mieux que la soudure), 1 transistor petits signaux NPN genre 2N2222A et 1 transistor PNP (moyennement) puissant genre BD136, je crois que j'ai fait le tour…..



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

Circuit électrique autour des sondes

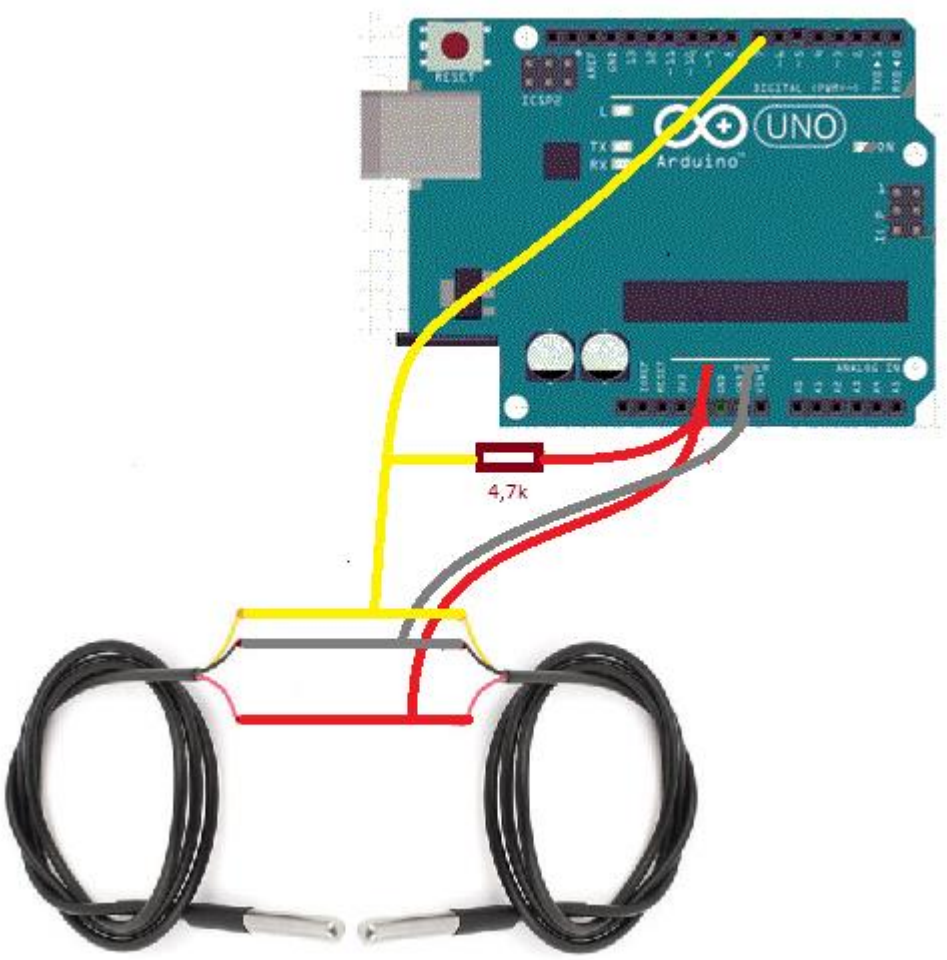
Les sondes numériques DS18B20

Le capteur DS18B20 est un capteur 1-Wire, cela signifie qu'il communique avec une carte maître au moyen d'un bus 1-Wire. Plusieurs capteurs peuvent être reliés sur un même bus 1-Wire. De plus, chaque capteur dispose d'une adresse unique gravée lors de la fabrication, il n'y a donc pas de risque de conflit. Par contre il nous est impossible de savoir à l'avance laquelle correspond à la source, et au ballon d'eau chaude. Ce n'est qu'une fois la 1ere utilisation qu'on aura avantage à étiqueter les sondes.

Un bus 1-Wire est composé classiquement des trois fils : un fil de masse, un fil d'alimentation (5 volts) et un fil de données. Un seul composant externe est nécessaire pour faire fonctionner un bus 1-Wire : une simple résistance de 4.7K ohms en résistance de tirage à l'alimentation sur la broche de données.

Ici se trouve une illustration d'utilisation de ces sondes : https://www.carnetdumaker.net/articles/mesurer-une-temperature-avec-un-capteur-1-wire-ds18b20-et-une-carte-arduino-genuino/

Le schéma de câblage des sondes



Programme de test des sondes

1- Connecter les 2 sondes avec la résistance de 4,7K

2- On connecte l'Arduino au PC avec un câble USB.

3- Si ce n'est déjà fait installer l'interface de programmation, le programme est téléchargeable ici : https://www.arduino.cc/en/Main/OldSoftwareReleases

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

5- 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- Télécharger la librairie OneWire.h : https://github.com/PaulStoffregen/OneWire

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

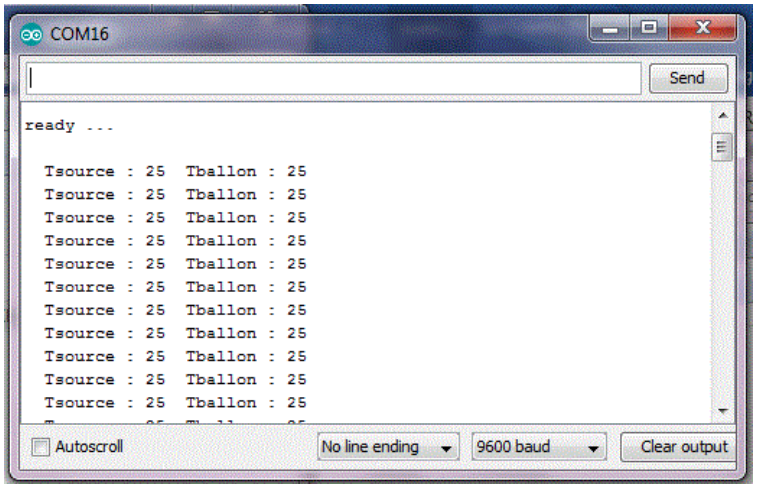
5- On efface ce qu'il y a dans la fenêtre d'édition

6- On y colle le code ci-dessous :

```
/*

test des sondes DS18B20

installer au préalable la librairie additionnelle pour gérer les sondes, OneWire.h disponible ici :
  https://www.pjrc.com/teensy/td_libs_OneWire.html

d'après :
https://www.carnetdumaker.net/articles/mesurer-une-temperature-avec-un-capteur-1-wire-ds18b20-et-une-carte-arduino-genuino/


*/

#include <OneWire.h>        // à importer pour gérer la sonde numérique DS18B20

const byte DS18 = 7;        // broche du bus 1-Wire pour chacune des sondes DS18B20

OneWire sondeDS(DS18);      // Création de l'objet OneWire pour manipuler le bus

enum DS18B20_RCODES {       // énumération de constantes => code de retour de getTemperature()
  READ_OK,                  // Lecture ok
  NO_SENSOR_FOUND,          // Pas de capteur
  INVALID_CRC,              // CRC invalide
  INVALID_SENSOR            // Capteur invalide (pas un DS18B20)
};


//
// SETUP
//_____________________________________________________________________________________

void setup() {

// initialisation du mode console
  Serial.begin(9600);             // préparation du moniteur série
  Serial.println ();
  Serial.println("ready ...");
  Serial.println ();
  delay(1500);
}                                 //Fin de setup


//
// GETTEMPERATURE : initialisation des sondes DS18B20 et mesure des températures
//_____________________________________________________________________________________

byte getTemperature(float *temperature, byte reset_search) {
  byte data[9], addr[8];           // data[] : Données lues depuis le scratchpad
                                   // addr[] : Adresse du module 1-Wire détecté

// reset le carnet d'adresses des sondes, pour le 1er capteur seulement suivant
// la déclaration initiale : char argument = "true" ou "false"
  if (reset_search) { sondeDS.reset_search(); }

// recherche l'adresse des (nouvelles) sondes, le tableau addr stocke les adresses
  if (!sondeDS.search(addr)) { return NO_SENSOR_FOUND; }          // Pas de capteur
```

```
// vérifie que le CRC soit correct :
  if (OneWire::crc8(addr, 7) != addr[7]) { return INVALID_CRC; }  // CRC invalide

// vérifie qu'il s'agit bien d'un DS18B20 :
  if (addr[0] != 0x28) { return INVALID_SENSOR; }        // Mauvais type de capteur

  sondeDS.reset();                  // reset le bus 1-Wire
  sondeDS.select(addr);             // sélection de la sonde
  sondeDS.write(0x44, 1);           // mesure de température => demande 1/2 seconde chacune
  delay(800);                       // delay d'acquisition
  sondeDS.reset();
  sondeDS.select(addr);
  sondeDS.write(0xBE);              // demande d'envoi du scratchpad => données de la sonde

// lecture du contenu du scratchpad qui fait 9 octets :
  for (byte i = 0; i < 9; i++) { data[i] = sondeDS.read(); }

// formule miracle pour avoir la température avec une résolution sur 12 bits soit 0,06°K près :
  *temperature = ((data[1] << 8) | data[0]) * 0.0625;

 return READ_OK;      // cas pas d'erreur
}                     // fin de fonction getTemperature


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

void loop() {

  float temperature[2];

  /* Lit les températures des deux capteurs */
  if (getTemperature(&temperature[0], true) != READ_OK) {
    Serial.println("Erreur de lecture du capteur 1");
    return;
  }
  if (getTemperature(&temperature[1], false) != READ_OK) {
    Serial.println("Erreur de lecture du capteur 2");
    return;
  }

  /* Affiche les températures */
  Serial.print("  Tsource : ");
  Serial.print(temperature[0], 2);
  Serial.print("  Tballon : ");
  Serial.print(temperature[1], 2);
  Serial.println();

}  // fin de loop.
```

Enregistrer ce programme sur le PC sous le nom de test_sondes_DS18B20.ino par exemple.
Puis : Menu croquis -> téléverser.
Ouvrir le moniteur série : Menu outils -> moniteur série
Une autre fenêtre s'ouvre. Il doit s'afficher les 2 valeurs de température. Celles-ci doivent évoluer en fonction des tests de froid et de chaud que nous pouvons leur faire subir.



Ça marche ? on continue.

Circuit électrique autour de l'afficheur LCD

L'Afficheur 1602

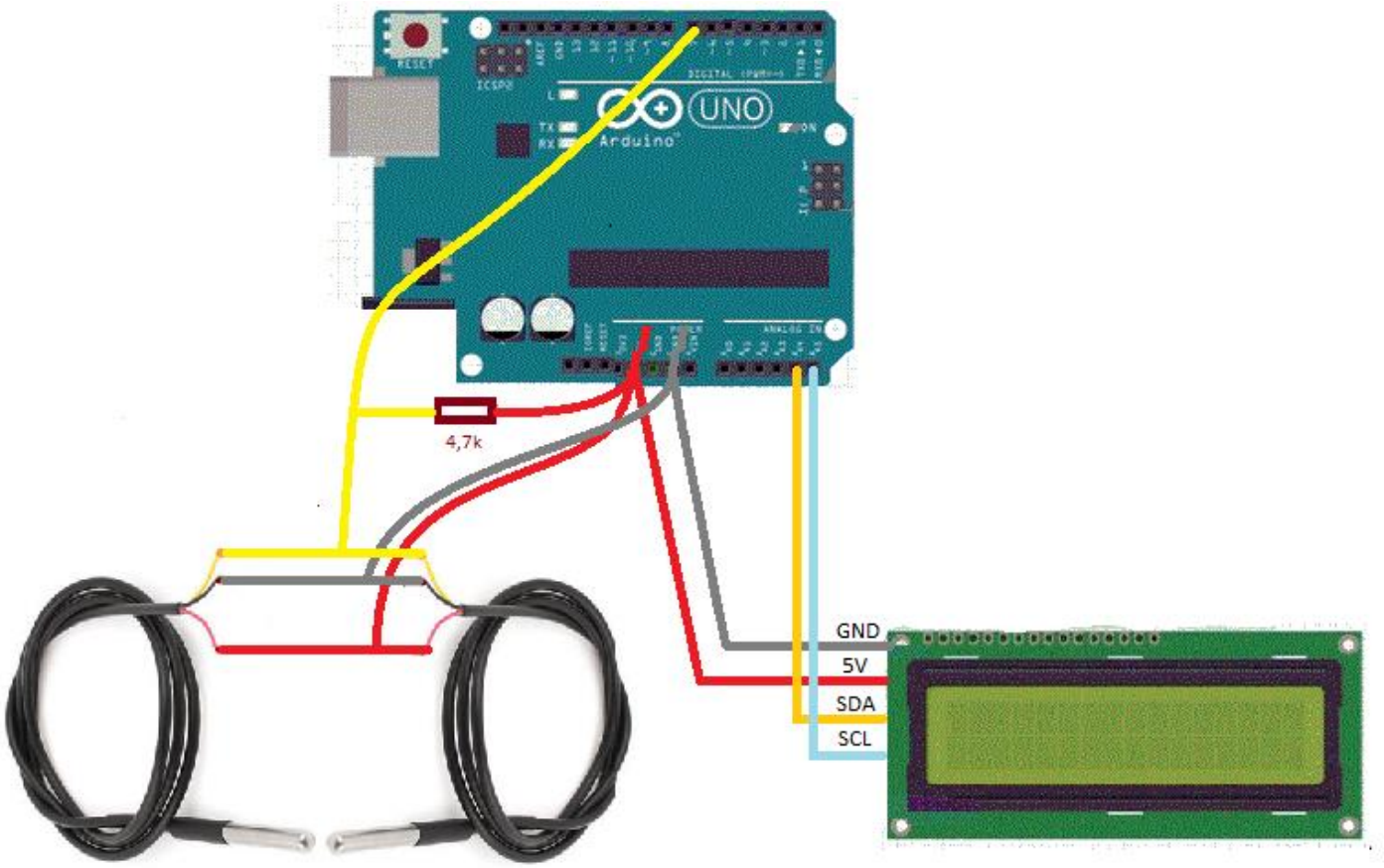
L'afficheur 1602, pour 16 caractères sur 2 lignes est l'afficheur le plus basic qui soit. Il utilise 4 ports pour les données et au total 6 ports entrées/sorties. Afin de réduire le câblage, nous utiliserons un convertisseur I2C – c'est le nom du protocole de communication – qui « transforme » ces 6 entrées/sorties en seulement 2 : SDA et SCL.

La datasheet du modèle v1602 est disponible ici : https://www.openhacks.com/uploadsproductos/eone-1602a1.pdf

Toute l'information au sujet du protocole I2C se trouve ici : http://electro8051.free.fr/I2C/protocole.html

L'arduino Uno traite le protocole I2C avec respectivement SDA en A4, SCL en A5 (entrées analogies 4 et 5)

Le schéma de câblage du 1602 avec l'I2C



Programme de test du LCD

Nous allons reprendre le programme de test précédent, avec le code nécessaire pour l'affichage sur le LCD.

L'utilisation du LCD fait appel à la librairie LiquidCrystal.h qui est installée nativement avec le programme PC. Cependant avec l'I2C nous allons utiliser une nouvelle bibliothèque *LiquidChrystal.h.* Or cette nouvelle bibliothèque fait appel aux même fonctions que la bibliothèque d'origine : 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.

Ensuite télécharger la nouvelle librairie LiquiCrystal_I2C, il y en a plusieurs disponibles, celle qui a été utilisé est celle-ci : https://bitbucket.org/fmalpartida/new-liquidcrystal/downloads/LiquidCrystal_V1.2.1.zip

Comme précédemment avec OneWire, la déclarer dans l'IDE : Sketch > Include Library > Add .ZIP Library > select the downloaded file > Open

Enfin créer un nouveau programme avec le code ci-dessous :

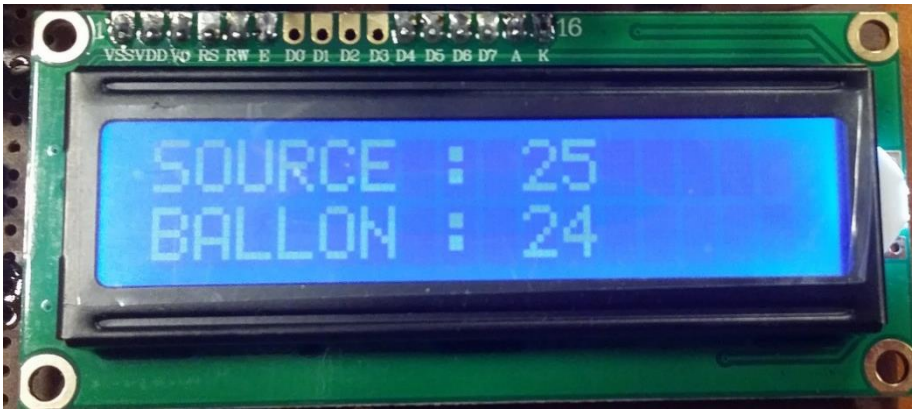
```
/*

test des sondes DS18B20 avec LCD en I2C

installer au préalable les librairies additionnelles :
- pour gérer les sondes, OneWire.h à installer et disponible ici :
  https://www.pjrc.com/teensy/td_libs_OneWire.html
- pour l'utilisation du LCD avec I2C :
  https://andrologiciels.wordpress.com/arduino/lcd/lcd-1602-i2c/

d'après :
https://www.carnetdumaker.net/articles/mesurer-une-temperature-avec-un-capteur-1-wire-ds18b20-et-une-carte-arduino-genuino/

*/

#include <OneWire.h>              // à importer pour gérer la sonde numérique DS18B20
#include <LiquidCrystal_I2C.h>    // à importer pour l'utilisation du LCD avec I2C

// le brochage pour l'exploitation du LCD :

// 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


const byte DS18 = 7;          // broche du bus 1-Wire pour chacune des sondes DS18B20

OneWire sondeDS(DS18);        // Création de l'objet OneWire pour manipuler le bus

enum DS18B20_RCODES {         // énumération de constantes => code de retour de getTemperature()
  READ_OK,                    // Lecture ok
  NO_SENSOR_FOUND,            // Pas de capteur
  INVALID_CRC,                // CRC invalide
  INVALID_SENSOR              // Capteur invalide (pas un DS18B20)
};


//
// SETUP
//_____________________________________________________________________________________

void setup() {

// initialisation de l'affichage et du mode console
  Serial.begin(9600);             // préparation du moniteur série
  Serial.println ();
  Serial.println("ready ...");
  Serial.println ();
  lcd.clear();                    //Effacement de l'afficheur.
  lcd.setCursor(0, 0);            //Positionnement du curseur.
  lcd.print("PTIWATT ***");       //Affichage d'un message.
  lcd.setCursor(0, 1);            //Positionnement du curseur.
  lcd.print("BONJOUR !");         //Affichage d'un message.
  delay(1500);
}                                 //Fin de setup


//
// GETTEMPERATURE : initialisation des sondes DS18B20 et mesure des températures
//_____________________________________________________________________________________

byte getTemperature(float *temperature, byte reset_search) {
  byte data[9], addr[8];           // data[] : Données lues depuis le scratchpad
                                   // addr[] : Adresse du module 1-Wire détecté

// reset le carnet d'adresses des sondes, pour le 1er capteur seulement suivant
// la déclaration initiale : char argument = "true" ou "false"
  if (reset_search) { sondeDS.reset_search(); }

// recherche l'adresse des (nouvelles) sondes, le tableau addr stocke les adresses
```

```
  if (!sondeDS.search(addr)) { return NO_SENSOR_FOUND; }          // Pas de capteur

// vérifie que le CRC soit correct :
  if (OneWire::crc8(addr, 7) != addr[7]) { return INVALID_CRC; }  // CRC invalide

// vérifie qu'il s'agit bien d'un DS18B20 :
  if (addr[0] != 0x28) { return INVALID_SENSOR; }        // Mauvais type de capteur

  sondeDS.reset();                  // reset le bus 1-Wire
  sondeDS.select(addr);             // sélection de la sonde
  sondeDS.write(0x44, 1);           // mesure de température => demande 1/2 seconde chacune
  delay(800);                       // delay d'acquisition
  sondeDS.reset();
  sondeDS.select(addr);
  sondeDS.write(0xBE);              // demande d'envoi du scratchpad => données de la sonde

// lecture du contenu du scratchpad qui fait 9 octets :
  for (byte i = 0; i < 9; i++) { data[i] = sondeDS.read(); }

// formule miracle pour avoir la température avec une résolution sur 12 bits soit 0,06°K près :
  *temperature = ((data[1] << 8) | data[0]) * 0.0625;

 return READ_OK;     // cas pas d'erreur
}                    // fin de fonctiongetTemperature


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

void loop() {

  float temperature[2];

  /* Lit les températures des deux capteurs */
  if (getTemperature(&temperature[0], true) != READ_OK) {
    Serial.println("Erreur de lecture du capteur 1");
    return;
  }
  if (getTemperature(&temperature[1], false) != READ_OK) {
    Serial.println("Erreur de lecture du capteur 2");
    return;
  }

  /* Affiche les températures */
  Serial.print("  Tsource : ");
  Serial.print(temperature[0], 2);
  Serial.print("  Tballon : ");
  Serial.print(temperature[1], 2);
  Serial.println();

  lcd.setCursor(0, 0);            // positionnement du curseur 1ère ligne
  lcd.print("SOURCE :");
  lcd.print(String(temperature[0], 1));  // 1 chiffre après la virgule
  lcd.setCursor(0, 1);            // positionnement du curseur 2ème ligne
  lcd.print("BALLON : ");
  lcd.print(String(temperature[1], 1));

}  // fin de loop.
```

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

Puis : Menu croquis -> téléverser.

Nous devons obtenir l'affichage qui ressemble au suivant :



Ça marche ? on continue.

Sauvegarde de données hors tension : l'EEPROM

Le module Arduino possède une EEPROM qui permet de conserver des données lorsque qu'il n'est plus alimenté. Pour ce faire il suffit de faire appel à la librairie EEPROM.h, installée nativement avec le programme PC.

Les paramètres que nous avons à conserver hors tension sont :

- deltaTon : seuil entre source et cumulus de mise en route de la pompe, par défaut : 5°C
- deltaToff : seuil entre source et cumulus d'arrêt de la pompe, par défaut : 2°C
- Tmaxi : température maximale du cumulus avant arrêt et alarme, par défaut : 85°C
- Tmini : température de mise en hors-gel, par défaut : -1°C
- Pompe_model : pompe autonome, à pwm externe croissant, à pwm externe décroissant, par défaut : autonome
- Inertie : durée du cycle pour calcul de la tendance à la chauffe ou au refroidissement, par défaut : 1s

A noter : l'EEPROM ne conserve que des entiers positifs inférieurs à 255.
Toutes les informations se trouvent ici : https://www.arduino.cc/en/Reference/EEPROM

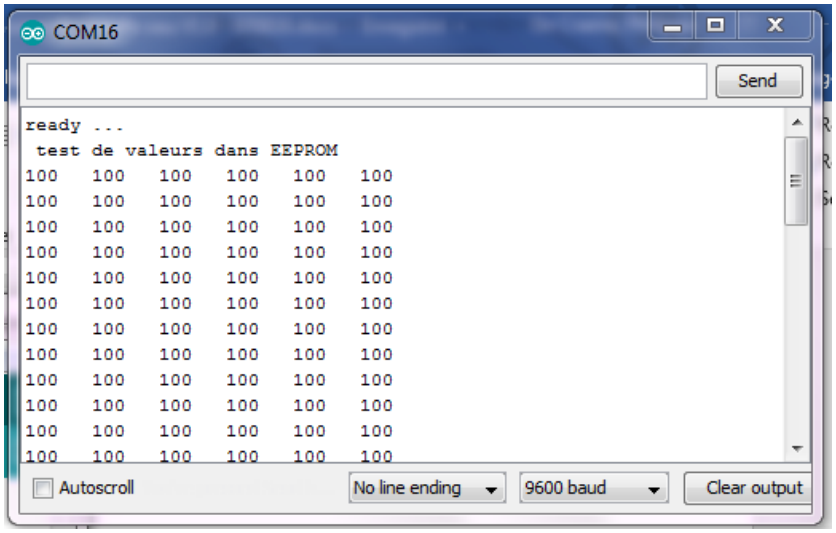
Voici une procédure de test de l'EEPROM :

1- Charger le programme de test pour l'écriture de données :

```
/* test écriture dans EEPROM */

#include <EEPROM.h>

int val0 = 100;
int val1 = 100;
int val2 = 100;
int val3 = 100;
int val4 = 100;
int val5 = 100;

void setup(){

  Serial.begin(9600);
  Serial.print("ready ...");
  delay(500);

  Serial.println();
  Serial.println(" test de valeurs dans EEPROM");

// EEPROM ne fonctionne qu'avec des valeurs entières et positives.

  EEPROM.update(0,val0);
  EEPROM.update(1,val1);
  EEPROM.update(2,val2);
  EEPROM.update(3,val3);
```

```
  EEPROM.update(4,val4);
  EEPROM.update(5,val5);
}

void loop() {

val0=EEPROM.read(0);
val1=EEPROM.read(1);
val2=EEPROM.read(2);
val3=EEPROM.read(3);
val4=EEPROM.read(4);
val5=EEPROM.read(5);

 Serial.print(val0);
 Serial.print("   ");
 Serial.print(val1);
 Serial.print("   ");
 Serial.print(val2);
 Serial.print("   ");
 Serial.print(val3);
 Serial.print("   ");
 Serial.print(val4);
 Serial.print("   ");
 Serial.print(val5);
Serial.println("   ");
 delay(500);
}
```

2- A l'exécution, à la console nous devons obtenir une série de valeur 100 :



3- Puis on charge le programme de test de lecture. Remarquez qu'il n'y a dedans aucune assignation de valeur aux variables val0 à val5 :

```
/* test lecture EEPROM */

#include <EEPROM.h>

int val0;
int val1;
int val2;
int val3;
int val4;
int val5;

void setup(void){

  Serial.begin(9600);
  Serial.print("ready ...");
  delay(500);

  Serial.println();
  Serial.println(" test de valeurs dans EEPROM");
}
```

```
void loop(void) {

val0=EEPROM.read(0);
val1=EEPROM.read(1);
val2=EEPROM.read(2);
val3=EEPROM.read(3);
val4=EEPROM.read(4);
val5=EEPROM.read(5);

 Serial.print(val0);
 Serial.print("   ");
 Serial.print(val1);
 Serial.print("   ");
 Serial.print(val2);
 Serial.print("   ");
 Serial.print(val3);
 Serial.print("   ");
 Serial.print(val4);
 Serial.print("   ");
 Serial.print(val5);
 Serial.println("   ");

 delay(500);
}
```

4- On débranche l'Arduino pour le priver de toute alimentation électrique.

5- On ferme la fenêtre de la console pour la ré-ouvrir, car d'avoir débranché l'USB désactive la console.

6- On remarque que les données sont récupérées.

Ça marche ? on continue.

Le circuit de commande pwm / mli

Fonctionnement des pompes à commande PWM / MLI externe

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

Les pompes de nouvelle génération sont construites avec des moteurs synchrones. C'est-à-dire que leur vitesse de rotation est synchronisée avec la fréquence de leur alimentation, à savoir le secteur 50 Hz.

Pour plus d'informations : https://www.cahiers-techniques-batiment.fr/article/une-generation-de-circulateurs-a-la-pointe-de-l-efficacite-energetique.18718

Suivant les mesures faites sur le régulateur Steca TR 503, le signal pwm/mli qui commande l'UPM3 possède les caractéristiques suivantes :

- Signal carré d'amplitude de 10V, de fréquence de 250Hz, à rapport cyclique variable de 0 (arrêt) à 100% (marche totale de la pompe).
- La pompe est alimentée par le 230V en permanence. La pompe s'arrête dès lors que le rapport cyclique du signal pwm/mli est à 0% - soit 0V permanent.
- Au démarrage le rapport cyclique est de 50%. Puis il augmente progressivement jusqu'à 100%, tant que la température du Cumulus se rapproche de quelques degrés de la température de la source.
- Lorsque Tcumulus atteint deltaToff à 2°K près, suivant la vitesse de la montée en température le rapport cyclique du signal reste stable, ou diminuer pour baisser jusqu'à 25%.

Comment fabrique un signal pwm de 250Hz :

Plusieurs sorties numériques de l'Arduino proposent de base un signal pwm de 490Hz avec la commande :

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

Cependant il nous faut un signal de fréquence 2 fois plus basse. Aucune instruction simple permet de réaliser cette action, aussi nous allons « bricoler » dans les registres. Un excellent site explique la démarche :

https://www.locoduino.org/spip.php?article84

Sachant que :

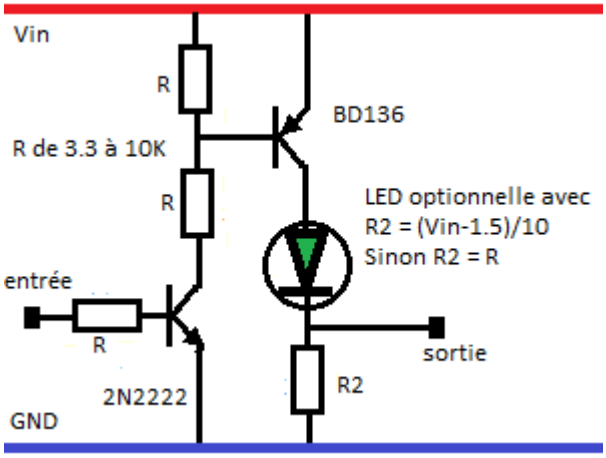
- la Fréquence disponible est : Fclock / 256 = 16MHz / 256 = 62,5kHz,
- la Fréquence possible doit être un multiple binaire de la fréquence disponible (divisible par 2, 4, 16, etc),
- En divisant par 256 on obtient : 62500/256 = 244Hz, soit presque 250Hz, de toute façon on ne pourra pas faire mieux à moins de fabriquer une « usine à gaz ».

Le code pour générer ce signal est le suivant :

```
TCCR2A = 0b00100011;  // paramétrage Fast PWM Mode
TCCR2B = 0b00000110;  // formule => Nbinaire = 62500 / (F * 256)
OCR2B = 128;            // rapport cyclique de 0 à 255
```

Montage électronique du circuit de commande pwm / mli

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



Le signal haut 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 PWM vaut Vin.

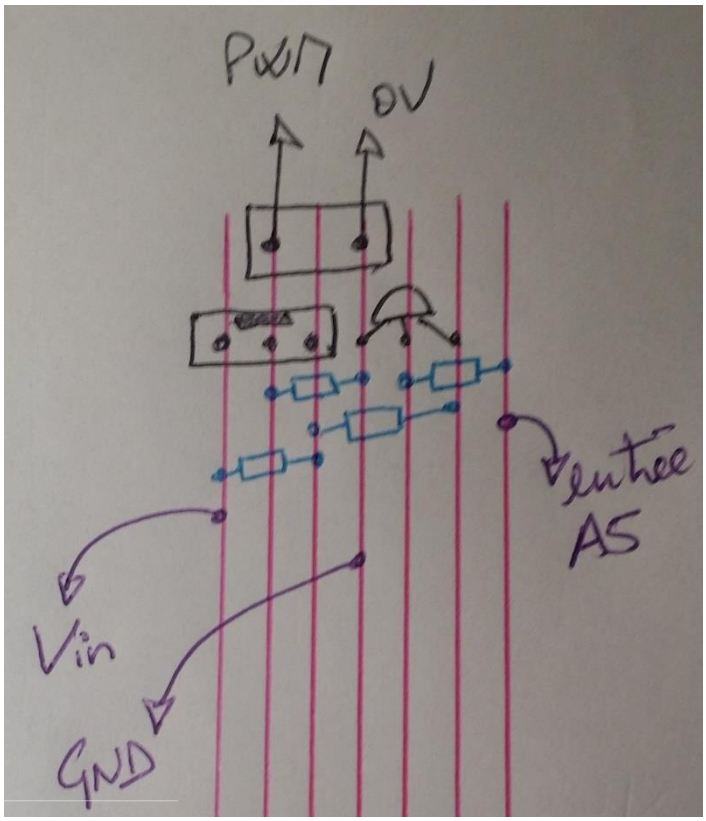
Pour le 2N2222 n'importe quel transistor NPN petits signaux fait l'affaire. Juste faire attention au brochage.

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…

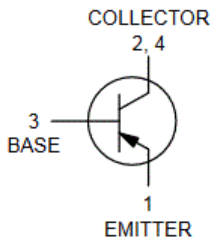
R vaut entre 3,9K et 10K

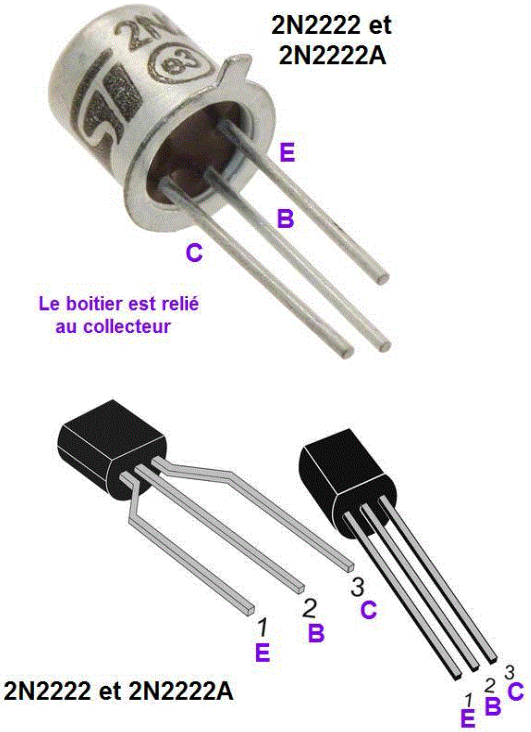
Remarque : la base du 2N2222 est relié au port 3 de l'Arduino, et non pas A5 contrairement aux photos ci-jointes.

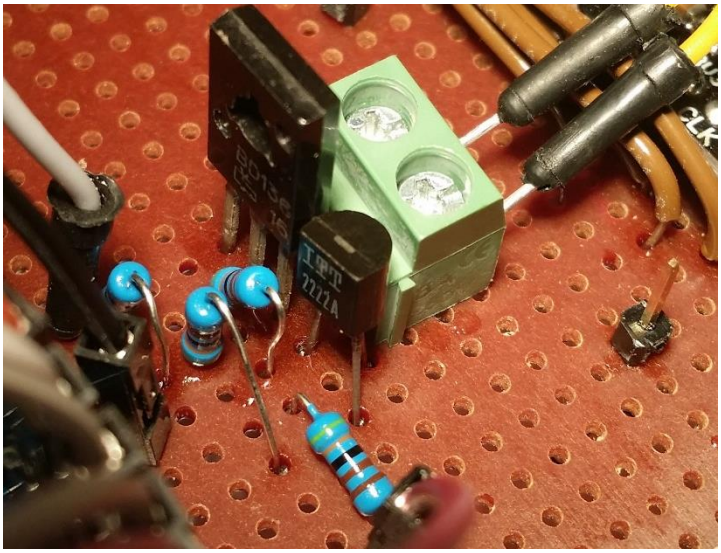
L'implantation des composants se fera sur la plaquette de montage, comme décrit avec le croquis ci-contre.



Brochage du BD136 :







L'utilisation d'un SSR pour alimenter la pompe

Le montage doit être compatible avec l'utilisation des anciens modèles de pompes qui n'utilisent pas de signal pwm/mli. Il faut donc un circuit de commande de puissance qui permettra l'alimentation secteur – ou pas – de la pompe. Pour cela nous allons faire l'usage d'un interrupteur commandable, c'est-à-dire un relai. Ce relai peut être mécanique, ou bien électronique à bas d'un Triac ou Thyristor, le nom usuel dans ce cas est SSR pour Solid State Relay

Fonctionnement du programme

Le séquencement

Au démarrage le programme récupère les paramètres en EEPROM. Il effectue ensuite séquentiellement les tâches suivantes :

1. « Regarde » si la touche « ENTREE » n'est toujours pas appuyé,
2. Relève les températures
3. Comparaison avec les consignes s'il y a lieu de mettre la pompe en marche + calcul du pwm/mli
4. Mesure de la tendance (au réchauffement ou au refroidissement), ce qui impacte la valeur du pwm / mli,
5. Affiche les températures sur l'écran LCD.

Si la touche entrée est appuyée, le programme entre dans une succession d'affichage des fenêtres pour passer en revue les paramètres.

Or si la partie loop() s'exécute plusieurs dizaines de fois par secondes, la majorité des opérations à exécuter n'ont pas à s'ordonnancer à un tel rythme. En effet il est inutile de surcharger les opérations à exécuter, comme le relevé des températures des sondes ou l'actualisation de l'affichage qui provoquera le scintillement.

La mesure du temps avec son intégration dans des tests permet de réaliser des temporisations et permet d'imaginer un séquencement différent :

1- A chaque passage du programme loop(), et donc en temps réel on effectue :
   a. Le relevé du temps,
   b. La vérification si la touche « ENTREE » est appuyée ou non.

2- Si la touche « ENTREE » est appuyée :
   a. Une temporisation d'une dizaine de seconde est lancée : si aucun bouton n'est appuyé durant ce laps de temps le programme revient à l'état initial.
   b. Passage en revue à chaque appui sur la touche « ENTREE » des paramètres dans l'ordre suivant :
      i. Affichage de deltaTon, avec possibilité de modifier la valeur avec les boutons « + » et « - »,
      ii. Affichage de deltaToff, avec la même possibilité de modification précédente,
      iii. Affichage de Tmaxi, avec la même possibilité de modification précédente,
      iv. Affichage de Tmini, avec la même possibilité de modification précédente,
      v. Affichage du modèle de pompe, avec le choix de modifier le modèle,
      vi. Affichage du niveau d'inertie, avec toujours la possibilité de modifier la valeur,
      vii. Affichage de la possibilité de réinitialiser les paramètres à leur valeur par défaut,
      viii. Affichage de la possibilité de forcer la mise en marche de la pompe.

3- A chaque cycle de 1 seconde les opérations suivantes sont exécutées :
   a. Le relevé des températures,
   b. La comparaison des températures aux consignes qui détermine la marche ou l'arrêt de la pompe
   c. La gestion de la mise en marche forcée, du hors-gel et l'alarme de surchauffe,
   d. Le calcul du rapport cyclique du signal pwm/mli.
   e. Mise à jour de l'affichage du LCD

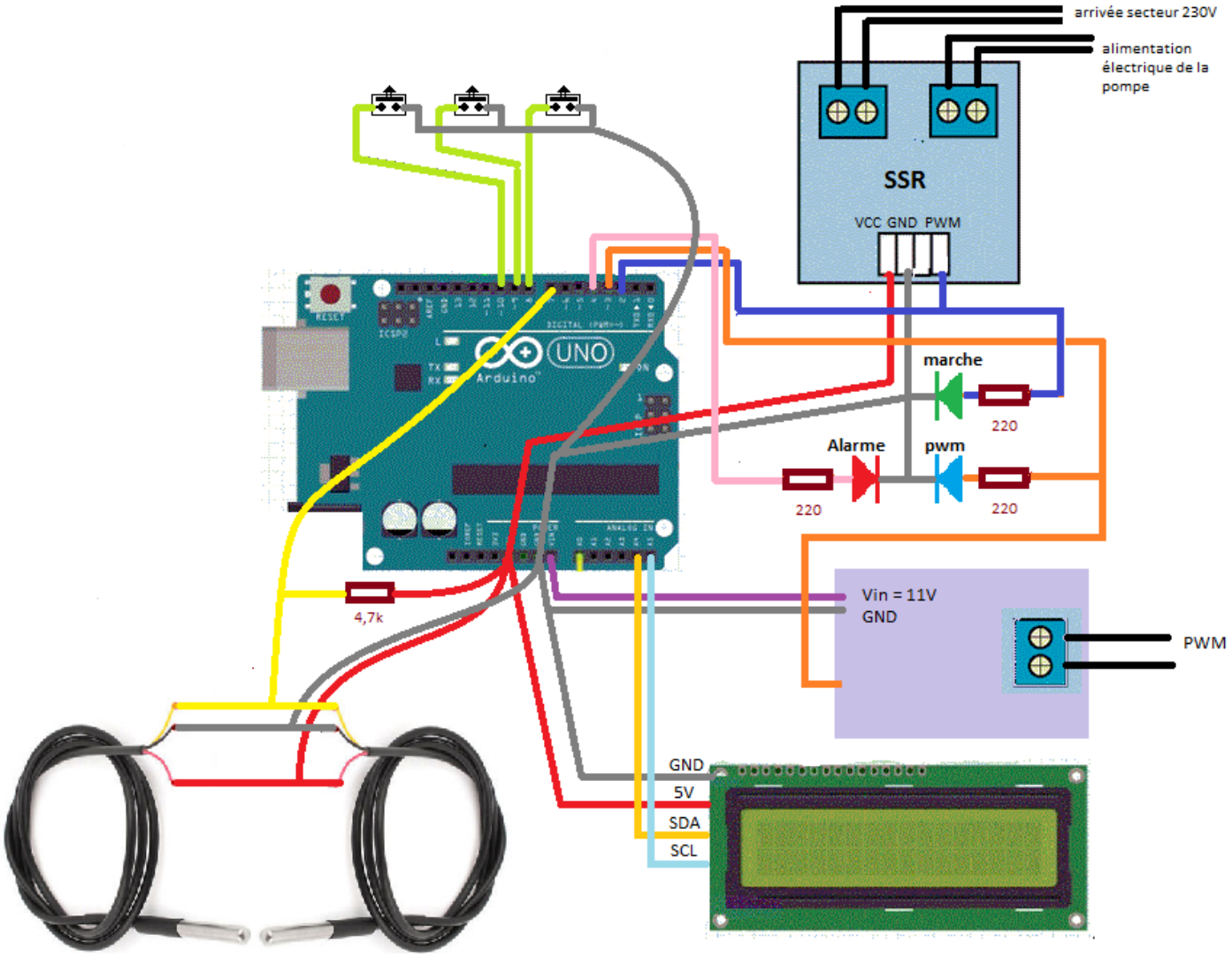
4- A chaque cycle de 5 secondes les opérations suivantes sont exécutées :
   a. La détermination de la tendance au réchauffement ou au refroidissement,
   b. La mise à jour si nécessaire de la sauvegarde des consignes.

Présentation algorithmique du programme :

Cycliquement – chaque seconde - les vérifications suivantes sont effectuées :

Si Tsource > Tcumulus + deltaTon	Mise en chauffe du cumulus : démarrage à 50% / 100% pour une pompe classique, ensuite le rapport augment jusqu'à la limite de 100%
Si Tsource < Tcumulus + deltaToff +deltaTpwm	Au seuil d'arrêt proche de la pompe, si la tendance est toujours à la chauffe le rapport cyclique diminue jusqu'à la limite de 25%
Si Tsource < Tcumulus + deltaToff	Arrêt de la pompe : 0%
Si Tcumulus > Tmaxi ou Tsource > Tmaxi +30	Arrêt de la pompe + alarme
Si Tsource < Tmini	Risque de gel => pompe en marche à 50% / 100% pour une pompe classique)
Si « marche forcée »	Ordre de marche forcée => pompe en marche à 100%

Le schéma de câblage



Remarques

Les sondes DS18B20 se connectent en parallèles. La fonction du programme *getTemperature* procède à une recherche des adresses des sondes, adresses fixées à leur construction. L'inconvénient pratique de la méthode est qu'il nous est impossible de savoir à l'avance laquelle correspond à la source, et au ballon d'eau chaude. Ce n'est qu'une fois la 1ere utilisation qu'on aura avantage à étiqueter les sondes.

En outre, et contre toute attente, le module SSR utilisé est à commande inversé : actif à 0V et au repos à 5V. Il faut donc créer une nouvelle sortie POMPE_INV (broche 5) qui est l'inverse de POMPE (broche 3) pour l'utiliser.

Le programme final

```
/*

thermostat_chauffe_eau est un système qui permet d'actionner une pompe de circulation d'eau
entre une source de chaleur telle qu'un panneau solaire ou une chaudière et un cumulus.

Le principe de fonctionnement est le suivant :
- une sonde DS18B20 placée au niveau de la source de chaleur compare la température avec une
seconde sonde DS18B20 placée dans le ballon d'eau chaude.
- au delà d'un seuil deltaTon la pompe est mise en marche, s'arrête en dessous de deltaToff
- la pompe se met en route lorsqu'il gèle Tmini
- un dispositif d'alarme signale le dépassement de température Tmaxi
- les seuils - deltaTon, deltaToff, Tmaxi et Tmini - ainsi que le modèle de pompe sont
paramétrables et sauvegardés hors tension
- prise en charges de la commande pwm / mli pour les pompes à contrôle externe.
  Suivant les mesures faites sur le régulateur Steca TR 503, le signal pwm possède les
  caractéristiques suivantes :
  - 250Hz à rapport cyclique variable de 0 (arrêt) à 100% (marche totale de la pompe) sous 10V,
  - la pompe est alimentée par le 230V en permanence,
  - 50% au démarrage, le rapport augmente progressivement jusqu'à 100%,
  - lorsque Tcumulus atteint deltaTpwm, le rapport cyclique peut diminuer jusqu'à 25%.

Le programme prévoit :
- 2 sondes DS18B20,
- 1 résistance de 4,7k entre le bus de données et le +5V des sondes DS18B20
- 1 module de gestion d'un triac ou d'un SSR
- 1 afficheur LCD 16x2 pour Arduino avec l'extension I2C pour le mode série,
- 3 boutons poussoirs,
- la librairie additionnelle pour gérer les sondes, OneWire.h à installer et disponible ici :
  https://www.pjrc.com/teensy/td_libs_OneWire.html
  https://github.com/PaulStoffregen/OneWire
- librairie additionnelle pour l'utilisation du LCD avec I2C :
  https://andrologiciels.wordpress.com/arduino/lcd/lcd-1602-i2c/
  https://bitbucket.org/fmalpartida/new-liquidcrystal/downloads/LiquidCrystal_V1.2.1.zip

ATTENTION ! il faut retrouver les fichiers d'origine LiquidCrystal.h et LiquidCrystal.cpp
présents
            dans le répertoire d'installation de l'IDE et les renommer en .orig sinon la
nouvelle
                    bibliothèque LiquidCrystal.h peut ne pas être reconnue

Merci à http://plaisirarduino.fr/ qui m'a aidé dans les astuces avec son programme :
thermostat-temperature-arduino
Merci à Christian son aide sur les timers https://www.locoduino.org/spip.php?article84

_____________________________________________________________
|                                                           |
|       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 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 module triac actif à LOW
version 1.4  - 20 mai  2019       - mise à jour des liens pour télécharger les bibliothèques

*/
```

```
#include <EEPROM.h>              // l'EEPROM pour sauvegarder les consignes hors tension
#include <Wire.h>                // provient aussi de l'Arduino IDE
#include <OneWire.h>             // à importer pour gérer la sonde numérique DS18B20
#include <LiquidCrystal_I2C.h>   // à importer pour l'utilisation du LCD avec I2C

// les variables paramétrables :

int Tmaxi_d = 85;           // par défaut la température d'alarme = 85°C
int Tmini_d = -1;           // par défaut la température hors gel = -1°C
int deltaTon_d = 5;         // par défaut le seuil de mise en route de la pompe = 5°K
int deltaToff_d = 2;        // par défaut le seuil d'arrêt de la pompe = 2°K
int deltaTpwm_d = 1;        // par défaut le seuil de diminution de pwm = 1°K
byte inertie_d = 1;         // valeur par défaut du temps de réaction pour la tendance = 1s
byte pompe_model_d = 0;     // modèle de pompe :
                            // 0 pour ordinaire, 1 pour pwm croissant, 2 pour pwm décroissant
byte taux_maxi = 100;       // en % le taux maximal de MLI
byte taux_mini = 25;        // en % le taux minimal de MLI

// le brochage pour l'exploitation du LCD :

// 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

// symboles spécifiques pour le LCD :

byte celsius[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};

// les autres entrées sorties :

// A4 => SDA du I2C du LCD
// A5 => SCL du I2C du LCD
const byte ENTREE    = 8;      // les 3 boutons poussoirs
const byte HAUT      = 9;
const byte BAS       = 10;
const byte DS18      = 7;      // broche du bus 1-Wire pour chacune des sondes DS18B20
const byte POMPE     = 2;      // la commande de Triac de la pompe
const byte POMPE_INV = 5;      // la commande inverse de Triac de la pompe
const byte MLI       = 3;      // la sortie MLI à 250Hz, broche 3 requise pour Timer2
const byte ALARM     = 4;      // LED ROUGE alarme dépassement de température limite

// Déclaration des sondes :

OneWire sondeDS(DS18);      // Création de l'objet OneWire pour manipuler le bus

enum DS18B20_RCODES {       // énumération de constantes => code de retour de getTemperature()
  READ_OK,                      // Lecture ok
  NO_SENSOR_FOUND,              // Pas de capteur
  INVALID_CRC,                  // CRC invalide
  INVALID_SENSOR                // Capteur invalide (pas un DS18B20)
};

// les variables pour la gestion des températures :

float Tsource = 0;          // température de la source de chaleur
float Tcumulus = 0;         // température du cumulus
int deltaTon;
int deltaToff;
int deltaTpwm;
int Tmaxi;
int Tmini;
float memo_Tcumulus;       // mémorisation de la valeur précédente pour calcul de la tendance
byte tendance = 3;         // tendance à la chauffe ou au refroidissement (3 = stable)

// les variables de la gestion de la pompe :

bool pompe = LOW;           // état d'activation de la pompe
bool memo_pompe = LOW;      // mémorisation de l'état de la pompe
char label_pompe = 'A';     // label affiché : A =arrêt, M =marche, G =gel, F = forcée
byte pompe_model;           // modèle de pompe
String pompe_model_label;   // étiquette indiquant le modèle de la pompe
bool mforce = LOW;          // paramètre marche forcée (pas forcée par défaut)

// les variables pour les boutons
```

```
byte etat_bouton_entree = 0;
byte precd_entree = 0;
byte etat_bouton_plus = 0;
byte precd_plus = 0;
byte etat_bouton_moins = 0;
byte precd_moins = 1;
byte ret_bouton_entree = 0;         // valeur retournée par la fonction bouton_entree()
byte ret_boutons_pm = 0;            // valeur retournée par la fonction boutons_pm()
byte fenetre = 0;                   // indice de changements de fenêtres d'affichage
byte fenetre_max = 9;               // nombre de fenetres à passer en revue

// les temporisations :

byte memo_temps_1;         // temporisation "cycle inertie"
byte memo_temps_2;         // temporisation d'accès aux paramètres
byte memo_temps_3;         // temporisation de 5 secondes pour la tendance

// les variables de gestion du PWM / MLI à 250Hz:

byte taux = 0;             // rapport cyclique du MLI
byte varia;                // dim ou inverse de dim suivant modèle de pompe HE

//
// SETUP
//_____________________________________________________________________________________________

void setup() {

  lcd.begin(16, 2);                // déclaration du LCD

  pinMode (ENTREE, INPUT_PULLUP);
  pinMode (HAUT, INPUT_PULLUP);
  pinMode (BAS, INPUT_PULLUP);
  pinMode (ALARM, OUTPUT);
  pinMode (POMPE, OUTPUT);
  pinMode (POMPE_INV, OUTPUT);
  pinMode (MLI, OUTPUT);

// paramétrage des registres pour un compteur Timer2 à 250Hz pour le pwm/mli :
// tout sur les compteurs ici : https://www.locoduino.org/spip.php?article119
  TCCR2A = 0b00100011;  // paramétrage Fast PWM Mode
  TCCR2B = 0b00000110;  // formule => Nbinaire = 62500 / (F * 256)
  OCR2B = 0;            // rapport cyclique de 0 à 255 - au départ 0% pour pompe éteinte

// LCD => Affectation des caractères spéciaux :
  lcd.createChar(1, celsius );        // Assigne 1 à ce caractère, etc....
  lcd.createChar(2, fleche_haut );
  lcd.createChar(3, fleche_stable );
  lcd.createChar(4, fleche_bas );

// Chargement des consignes en EEPROM :
// à savoir que l'EEPROM ne travaille qu'avec des nombres entiers de 0 à 255.
// d'où l'offset de 50 pour gérer les valeurs négatives,
// à priori à la 1ere utilisation la valeur est 255
  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) { pompe_model = EEPROM.read(5); }
  else { EEPROM.write(5, pompe_model); pompe_model = pompe_model_d; }

// initialisation de l'affichage et du mode console
  Serial.begin(9600);              // préparation du moniteur série
  Serial.println ();
  Serial.println("ready ...");
  Serial.println ();
  lcd.clear();                     //Effacement de l'afficheur.
  lcd.setCursor(0, 0);             //Positionnement du curseur.
  lcd.print("PTIWATT ***");        //Affichage d'un message.
  lcd.setCursor(0, 1);             //Positionnement du curseur.
  lcd.print("BONJOUR !");          //Affichage d'un message.
  delay(1500);
}                                  //Fin de setup


//
// GETTEMPERATURE : initialisation des sondes DS18B20 et mesure des températures
//_____________________________________________________________________________________________

byte getTemperature(float *temperature, byte reset_search) {
  byte data[9], addr[8];           // data[] : Données lues depuis le scratchpad
```

```
                                    // addr[] : Adresse du module 1-Wire détecté

// reset le carnet d'adresses des sondes, pour le 1er capteur seulement suivant
// la déclaration initiale : char argument = "true" ou "false"
  if (reset_search) { sondeDS.reset_search(); }

// recherche l'adresse des (nouvelles) sondes, le tableau addr stocke les adresses
  if (!sondeDS.search(addr)) { return NO_SENSOR_FOUND; }          // Pas de capteur

// vérifie que le CRC soit correct :
  if (OneWire::crc8(addr, 7) != addr[7]) { return INVALID_CRC; }  // CRC invalide

// vérifie qu'il s'agit bien d'un DS18B20 :
  if (addr[0] != 0x28) { return INVALID_SENSOR; }        // Mauvais type de capteur

  sondeDS.reset();                  // reset le bus 1-Wire
  sondeDS.select(addr);             // sélection de la sonde
  sondeDS.write(0x44, 1);           // mesure de température => demande 1/2 seconde chacune
  delay(800);                       // delay d'acquisition
  sondeDS.reset();
  sondeDS.select(addr);
  sondeDS.write(0xBE);              // demande d'envoi du scratchpad => données de la sonde

// lecture du contenu du scratchpad qui fait 9 octets :
  for (byte i = 0; i < 9; i++) { data[i] = sondeDS.read(); }

// formule miracle pour avoir la température avec une résolution sur 12 bits soit 0,06°K près :
  *temperature = ((data[1] << 8) | data[0]) * 0.0625;

 return READ_OK;     // cas pas d'erreur
}                    // fin de fonctiongetTemperature


//
// ECRAN_SUIVANT : procédure pour passer à la séquence d'affichage suivante
//_____________________________________________________________________________

void ecran_suivant() {
   if(ret_bouton_entree != 0 ) {             // nouvel appui sur le bouton ENTREE
     fenetre = (fenetre+1) % fenetre_max;    // incrémentation de l'indice de fenêtre modulo
     lcd.clear();                                        // fenetre_max
     ret_bouton_entree = 0;                  // réinitialisation de la touche ENTREE
     }
}          // fin de ecran_suivant function

//
// BOUTON_ENTREE : gestion du bouton entrée
//_____________________________________________________________________________

byte bouton_entree() {
  byte etat_bouton_entree = digitalRead(ENTREE);  // lecture de l'état des boutons

// Gestion de l'état du bouton ENTREE :

  if(etat_bouton_entree != precd_entree) {    // contrôle de l'état précédent un appui.
    if(etat_bouton_entree == LOW) {           // conditionnement d'exécution.
      precd_entree = etat_bouton_entree;      // chargement de l'état bouton.
      delay(250);                             // temporisation d'appuis.
      return 1;                               // retour d'état => ENTREE.
    }
  }
  precd_entree = etat_bouton_entree;          // chargement de l'état bouton
  return 0;                                   // état par défaut
}         // fin de bouton_entree function

//
// BOUTONS_PM : gestion des boutons plus et moins
//_____________________________________________________________________________

byte boutons_pm() {
  byte etat_bouton_plus = digitalRead(HAUT);
  byte etat_bouton_moins = digitalRead(BAS);

// Gestion de l'état du bouton PLUS :

  if(etat_bouton_plus != precd_plus) {          // contrôle de l'état précédent un appui
    if(etat_bouton_plus == LOW) {               // conditionnement d'exécution
      precd_plus = etat_bouton_plus;            // chargement de l'état bouton
      delay(250);                               // temporisation d'appuis
      return 2;                                 // retour d'état => PLUS
    }
  }
  precd_plus = etat_bouton_plus;                // mémorisation de l'état bouton

// Gestion de l'état du bouton MOINS :

  if(etat_bouton_moins != precd_moins) {        // contrôle de l'état précédent un appui
```

```
    if(etat_bouton_moins == LOW) {              // conditionnement d'exécution
      precd_moins = etat_bouton_moins;          // chargement de l'état bouton
      delay(250);                               // temporisation d'appuis
      return 3;                                 // retour d'état => MOINS
    }
  }
  precd_moins = etat_bouton_moins;              // mémorisation de l'état bouton
  return 0;
}                  // fin de boutons_pm function


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

void loop() {

//  ce qui est exécuté à chaque cycle : mesure du temps et contrôle des boutons
//_____________________________________________________________________________________

// mesure du temps qui va servir pour les temporisations :
  unsigned long temps_actuel = millis()/1000;     // mesure du temps en secondes

// contrôle de l'activité des 3 boutons poussoirs :
  ret_bouton_entree = bouton_entree();    // appel à la fonction pour voir si appui sur ENTREE
  ret_boutons_pm = boutons_pm();          // idem pour les boutons PLUS et MOINS

  if(fenetre != 0) {                      // si nous sommes dans les paramètres
    if(ret_bouton_entree != 0 || ret_boutons_pm != 0) { // en fait on lance la tempo dès qu'on
      memo_temps_2 = temps_actuel; }                    // n'appuie plus sur les boutons
    if (temps_actuel - memo_temps_2 >= 10) {            // contrôle de temporisation à 10s
      memo_temps_2 = temps_actuel;        // charge le temps actuel au passage de boucle
      ret_bouton_entree = 0;              // réinitialisation après la temporisation
      fenetre = 0;                        // retour à l'affichage des températures des sondes
      lcd.clear();
    }
  }

// ce qui est exécuté toutes les 5 secondes : détermination de la tendance + sauvegarde
//_____________________________________________________________________________________

// tendance à la chauffe ou au refroidissement :
  if(temps_actuel - memo_temps_3 >= 5) {              // temporisation de 5s
     memo_temps_3 = temps_actuel;

     if( Tcumulus > memo_Tcumulus ) { tendance = 2; }    // flèche haut sur lcd
     if( Tcumulus == memo_Tcumulus ) { tendance = 3; }   // flèche stable sur lcd
     if( Tcumulus < memo_Tcumulus ) { tendance = 4; }    // flèche bas sur lcd

     memo_Tcumulus = Tcumulus;            // mémorisation pour le cycle suivant

// sauvegarde éventuelle des paramètres s'il y a eu modification lors d'un cycle précédent :
     EEPROM.update(0, deltaTon + 50);    // mise à jour des consignes dans EEPROM
     EEPROM.update(1, deltaToff + 50);
     EEPROM.update(2, deltaTpwm + 50);
     EEPROM.update(3, Tmaxi + 50);
     EEPROM.update(4, Tmini + 50);
     EEPROM.update(5, pompe_model);
  }  // fin de tempo 3

//  ce qui est exécuté à chaque seconde :
//_____________________________________________________________________________________

  if(temps_actuel - memo_temps_1 >= 1) {   // test si temporisation dépasse 1 seconde
     memo_temps_1 = temps_actuel;

// ETAPE 1 : aquisition des valeurs de température :
//_____________________________________________________________________________________

// étant donné qu'il faut 1,6s pour lire les températures avec 2 fois un delay(800), il faut
// interrompte la procédure d'acquisition lorsque l'on entre dans les paramètres

  if(fenetre == 0) {
     getTemperature(&Tsource, true);    // true uniquement à la 1ere appel à la fonction
     getTemperature(&Tcumulus, false);
  }

// ETAPE 2 : gestion des températures => commande marche arrêt de la pompe + taux de pwm/mli :
//_____________________________________________________________________________________

// le seuil deltaTonn est atteint ou dépassé : la pompe se met en marche
  if( Tsource-Tcumulus-deltaTon >= 0 ) {
     pompe = HIGH;
     label_pompe = 'M';
     if(pompe != memo_pompe) { taux = 50; }   // pompe à 50% quand elle démarre
   }
```

```
// la pompe est en marche : le taux de pwm croît jusqu'au maximum
  if((pompe==HIGH)&&(Tsource-Tcumulus-deltaToff-deltaTpwm >0)&&(taux < 100)) { taux++; }

// le seuil deltaTpwm est atteint, soit proche du seuil d'arrêt :  le pwm peut diminuer
  if((pompe==HIGH)&&(Tsource-Tcumulus-deltaToff-deltaTpwm <=0)&&(tendance==2)&&(taux >
taux_mini)) { taux--; }

// le seuil deltaToff est atteint : arrêt de la pompe
  if(Tsource-Tcumulus-deltaToff <=0) {
     pompe = LOW;
     label_pompe = 'A';
     taux = 0;
  }

// cas dépassement de Tmaxi : arrêt de la pompe + alarme
  if(Tcumulus >= Tmaxi || Tsource > Tmaxi+30) {
     pompe = LOW;
     label_pompe = 'A';
     taux = 0;
     digitalWrite(ALARM, HIGH);               // LED rouge s'allume
  }
  else { digitalWrite(ALARM, LOW); }

// cas d'atteinte de Tmini : mise en route de la pompe pour éviter le gel
  if(Tsource <= Tmini) {                      // la pompe en marche en cas de gel Tmini
     pompe = HIGH;
     label_pompe = 'G';
     taux = 50;                               // fonctionnement à 50% en hors-gel
  }

// cas où la mise en marche est forcée (à travers le menu)
  if(mforce == HIGH) {                        // cas de marche forcée de la pompe
     pompe = HIGH;
     label_pompe = 'F';
     taux = 100;
   }

//  ETAPE 3 : détermination du mode de pwm ou mli suivant le modèle de pompe. Pour rappel :
//_____________________________________________________________________________________
// pompe_model == 0 pour pompe ORDINAIRE
// pompe_model == 1 pour pompe à MLI CROISSANT
// pompe_model == 2 pour pompe à MLI DECROISSANT

   if(pompe_model == 2) { varia = map(taux, 0, 100, 255, 0); }
   else { varia = map(taux, 0, 100, 0, 255); }

// ETAPE 4 : envoi des ordres de commande de la pompe :
//_____________________________________________________________________________________

   if( pompe_model == 0 ) {
     digitalWrite(POMPE, pompe);           // commande du triac actif à HIGH -> broche 2
     digitalWrite(POMPE_INV, !pompe);      // commande du triac actif à LOW  -> broche 5
   }
   else {                                  // pompe toujours alimentée en commande pwm / mli
     digitalWrite(POMPE, HIGH);
     digitalWrite(POMPE_INV, LOW);
   }

   analogWrite(MLI, varia);                // commande du pwm / mli

   memo_pompe = pompe;                     // mémorisation pour le cycle suivant

// pour le debugging sur la console :
   Serial.print("  taux : ");
   Serial.print(taux);
   Serial.print("  varia : ");
   Serial.print(varia);
   Serial.print("  pompe : ");
   Serial.print(pompe);
   Serial.print(" Ts : ");
   Serial.print(Tsource);
   Serial.print(" Tc : ");
   Serial.print(Tcumulus);
   Serial.print(" Ts-Tc-deltaToff : ");
   Serial.print(Tsource-Tcumulus-deltaToff);
   Serial.println();

// ETAPE 5 : gestion de l'affichage
//_____________________________________________________________________________________

// cas général : aucun bouton d'appuyé

 if( fenetre == 0 && ret_bouton_entree == 0) {
    lcd.setCursor(0, 0);             // positionnement du curseur 1ère ligne
    lcd.print("SOURCE:");
    if( Tsource < 100 ) {lcd.print(" "); } //   ajoute un " " si valeur sur 2 chiffre
    lcd.print(String(Tsource, 1));  // 1 chiffre après la virgule
```

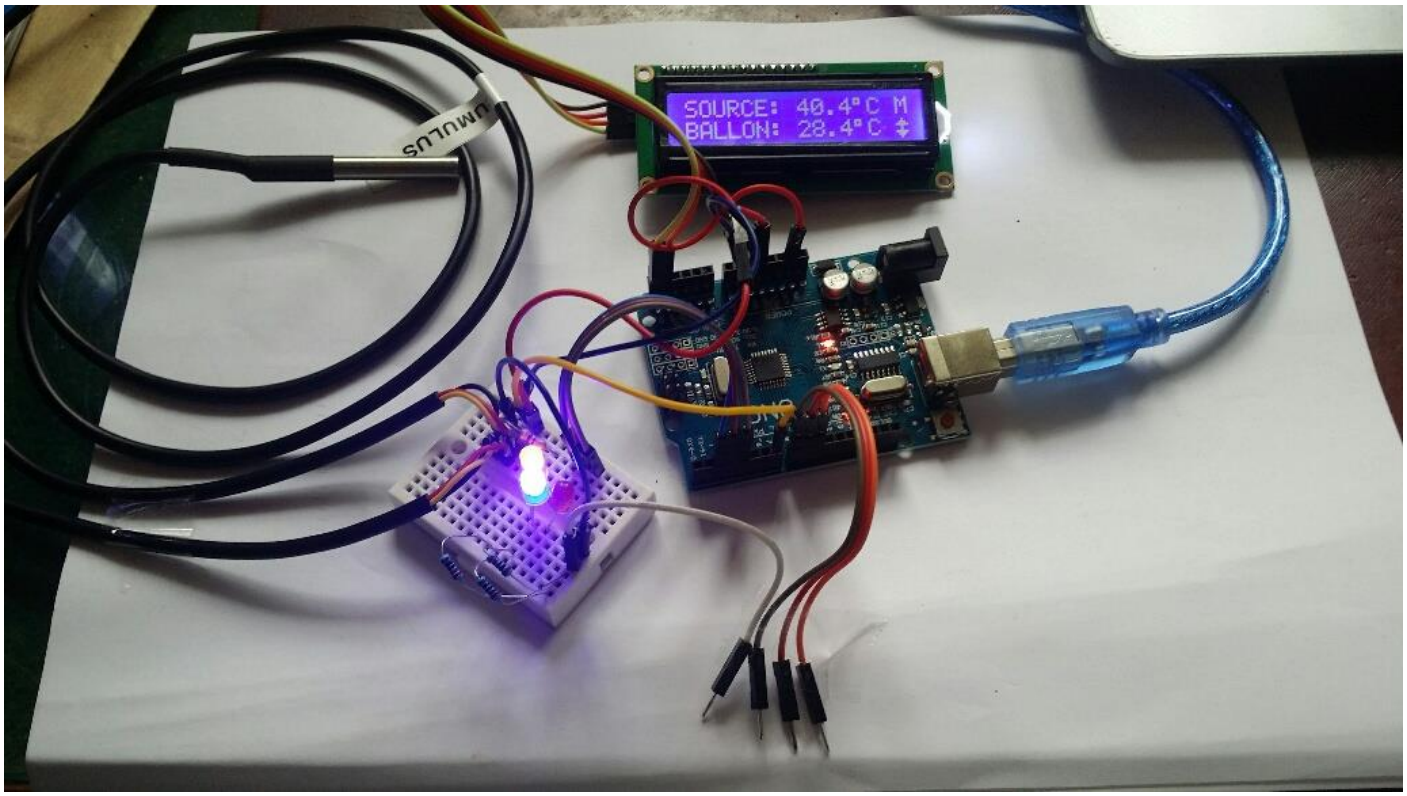
```
    lcd.write(1);                   // affichage du caractère 1 " ° " .
    lcd.print("C ");
    lcd.setCursor(15, 0);           // positionnement au dernier caractère de la 1ère ligne
    lcd.write(label_pompe);         // affichera A ou M suivant l'état de la pompe
    lcd.setCursor(0, 1);            // positionnement du curseur 2ème ligne
    lcd.print("BALLON: ");
    lcd.print(String(Tcumulus, 1));
    lcd.write(1);                   // affichage du caractère 1 " ° " .
    lcd.print("C ");
    lcd.setCursor(15, 1);           // positionnement au dernier caractère de la 2ème ligne
    lcd.write(tendance);            // affichage de la flèche de la tendance
 }
}   // fin de tempo 1 = "cycle inertie"

// ce qui est exécuté si la bouton ENTREE est appuyé : passage en revue des paramètres
//_____________________________________________________________________________________

 if(fenetre == 0 && ret_bouton_entree == 1) {
   lcd.clear();
   lcd.setCursor(0, 0);
   lcd.print("LISTE DES");
   lcd.setCursor(0, 1);
   lcd.print("PARAMETRES :");
   delay(800);              // un délai d'affichage avant de passer à l'écran suivant
   ecran_suivant();         // fonction pour passer à la suite de l'affichage des paramètres
 }

// passage en revue des paramètres suivant l'appui sur le bouton ENTREE

 if(fenetre ==1) {
   if(ret_boutons_pm == 2) {        // si appui sur PLUS la consigne s'incrémente
      deltaTon++;
      deltaTon = constrain(deltaTon, 0, 9); }    // valeur limitée entre 0 et 10
   if(ret_boutons_pm == 3) {        // si appui sur MOINS la consigne se décrémente
      deltaTon--;
      deltaTon = constrain(deltaTon, deltaToff, 9); }
   lcd.setCursor(0, 0);             // le seuil de mise en marche > seuil d'arrêt
   lcd.print("DELTA T MARCHE :");
   lcd.setCursor(0, 1);
   lcd.print(deltaTon);
   lcd.write(1);
   lcd.print("K");
   ecran_suivant();
 }
 if(fenetre == 2) {
   if(ret_boutons_pm == 2) {         // si appui sur PLUS la consigne s'incrémente
     deltaToff++;                    // le seuil d'arrêt < seuil mise en marche
     deltaToff = constrain(deltaToff, 0, deltaTon); }
   if(ret_boutons_pm == 3) {         // si appui sur MOINS la consigne se décrémente
     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");
   ecran_suivant();
 }
 if(fenetre == 3) {
   if(ret_boutons_pm == 2) {         // si appui sur PLUS la consigne s'incrémente
     deltaTpwm++;                    // le seuil d'arrêt < seuil mise en marche
     deltaTpwm = constrain(deltaTpwm, 0, 9); }
   if(ret_boutons_pm == 3) {         // si appui sur MOINS la consigne se décrémente
     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");
   ecran_suivant();
 }
 if(fenetre == 4) {
   if(ret_boutons_pm == 2) {         // si appui sur PLUS la consigne s'incrémente
     Tmaxi++;
     Tmaxi = constrain(Tmaxi, 60, 100); }   // valeur comprise entre 60 et 100
   if(ret_boutons_pm == 3) {         // si appui sur MOINS la consigne se décrémente
     Tmaxi--;
     Tmaxi = constrain(Tmaxi, 60, 100); }
   lcd.setCursor(0, 0);
   lcd.print("T MAXI ALARME :");
   lcd.setCursor(0, 1);
   lcd.print(Tmaxi);
   lcd.write(1);
   lcd.print("C");
```

```
  ecran_suivant();
 }
 if(fenetre == 5) {
   if(ret_boutons_pm == 2) {           // si appui sur PLUS la consigne s'incrémente
     Tmini++;
     Tmini = constrain(Tmini, -9, 9); }     // valeur comprise entre -10 et 5
   if(ret_boutons_pm == 3) {           // si appui sur MOINS la consigne se décrémente
     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");
   ecran_suivant();
 }
 if(fenetre == 6) {
   if(ret_boutons_pm == 2) {           // si appui sur PLUS la consigne s'incrémente
     pompe_model = (pompe_model+1)%3;  // le reste de la division par 3 donne entre 0 à 2
     pompe_model = constrain(pompe_model, 0, 2); }     // valeur comprise entre 0 et 2
   if(ret_boutons_pm == 3) {           // si appui sur MOINS la consigne se décrémente
     pompe_model = (pompe_model-1)%3;  // le reste de la division par 3 donne entre 0 à 2
     pompe_model = constrain(pompe_model, 0, 2); }
   if(pompe_model == 0) { pompe_model_label = "ORDINAIRE      "; }
   else if(pompe_model == 1) { pompe_model_label = "MLI CROISSANT  "; }
        else if(pompe_model == 2) { pompe_model_label = "MLI DECROISSANT"; }
   lcd.setCursor(0, 0);
   lcd.print("MODELE POMPE :");
   lcd.setCursor(0, 1);
   lcd.print(pompe_model_label);
   ecran_suivant();
 }
 if(fenetre == 7) {
   lcd.setCursor(0, 0);
   lcd.print("REINITIALISATION");
   lcd.setCursor(0, 1);
   lcd.print("POUR VALIDER : +");
   if(ret_boutons_pm == 2) {
      lcd.setCursor(0, 0);
      lcd.print("REINITIALISATION");
      lcd.setCursor(0, 1);
      lcd.print("FAITE           ");
      deltaTon = deltaTon_d;    // réaffectation des valeurs par défaut
      deltaToff = deltaToff_d;
      deltaTpwm = deltaTpwm_d;
      Tmaxi = Tmaxi_d;
      Tmini = Tmini_d;
      pompe_model = pompe_model_d;
      mforce = LOW;             // arrêt marche forcée
      fenetre = 0;
   }
   ecran_suivant();
 }
 if(fenetre == 8) {
   lcd.setCursor(0, 0);
   lcd.print("MARCHE FORCEE ?");
   lcd.setCursor(0, 1);
   lcd.print("OUI= + / NON= -");
   if(ret_boutons_pm == 2) {
      mforce = HIGH;            // si appui sur PLUS => marche forcée
      lcd.setCursor(0, 0);
      lcd.print("MARCHE FORCEE");
      lcd.setCursor(0, 1);
      lcd.print("CONFIRMEE       ");
      fenetre = 0;
   }
   if(ret_boutons_pm == 3) {
     mforce = LOW;              // si appui sur MOINS => arrêt marche forcée
     lcd.setCursor(0, 0);
     lcd.print("ARRET MARCHE");
     lcd.setCursor(0, 1);
     lcd.print("FORCEE");
     fenetre = 0;
   }
   ecran_suivant();
 }
}  // fin de loop.
```

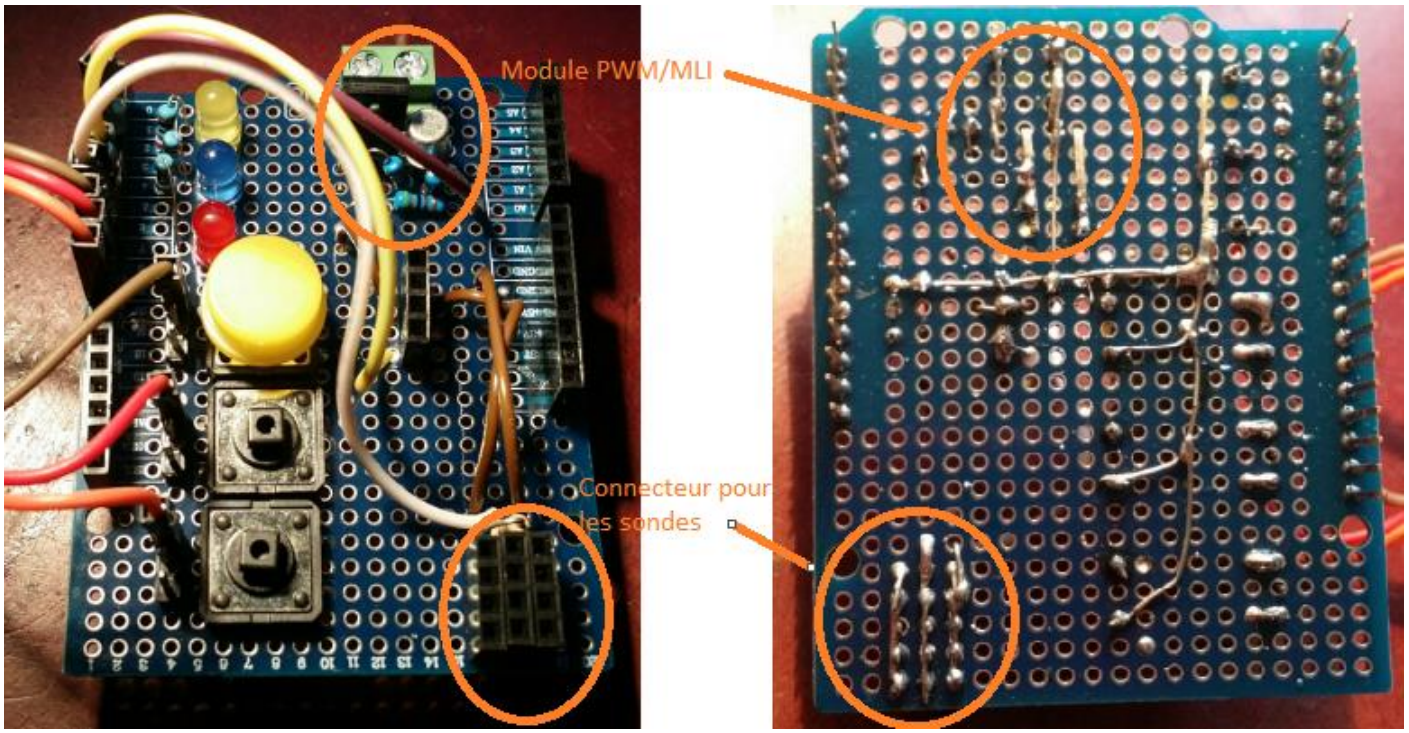
Illustration

Lors des essais

Les 4 fils qui se baladent en bas de la photo sont pour les 3 boutons poussoir.

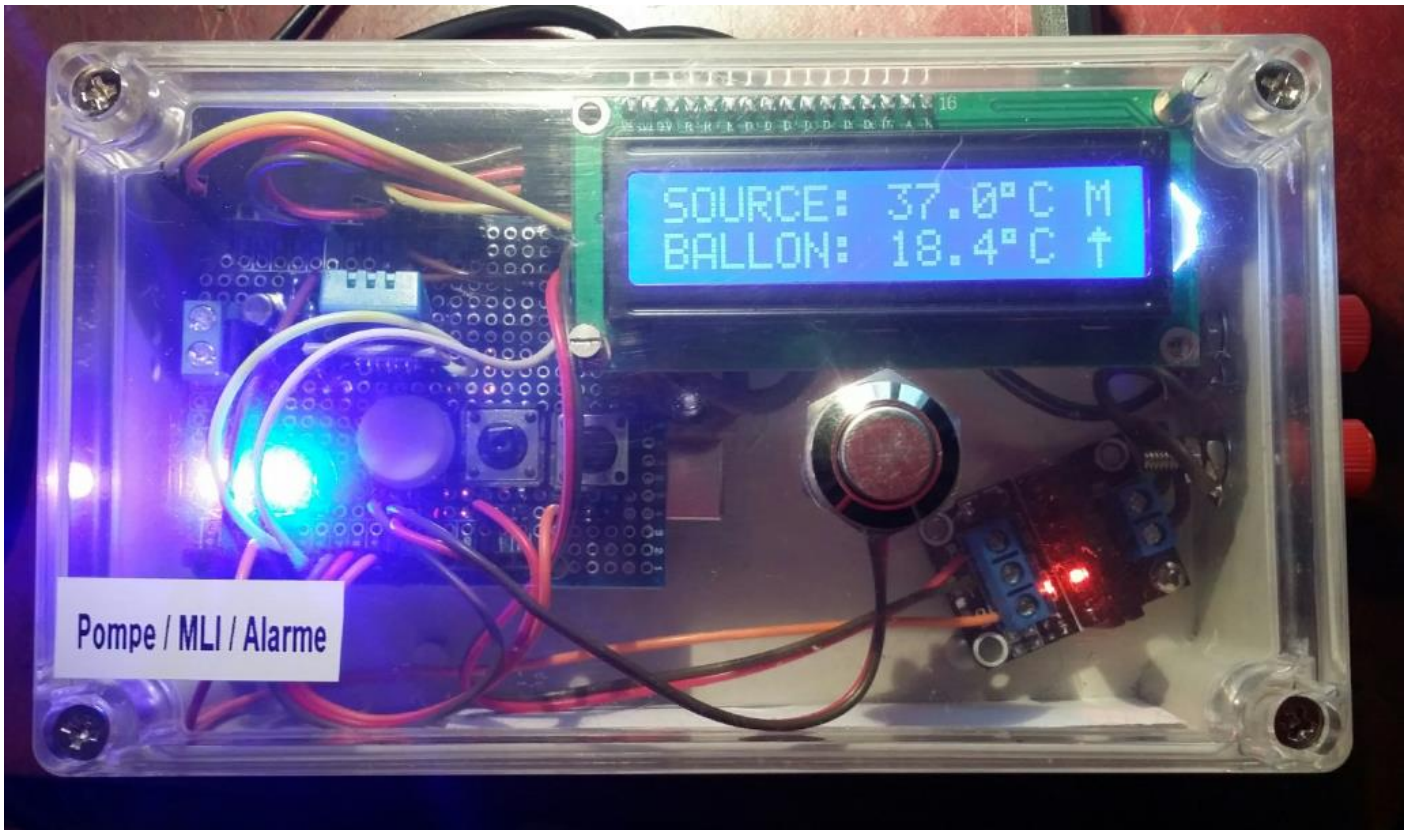


Implantation des composants sur le shield



Le connecteur au centre ainsi que le fil jaune ne concernent pas ce projet : il s'agit d'un connecteur pour une sonde DHT11 utilisé dans le cadre d'un autre projet.

Mise en boitier



Mode d'emploi

A l'allumage de l'appareil, après la séquence de démarrage du programme le contexte est affiché : la température de la source de chauffage, la température du cumulus, en plus 2 indicateurs :

En haut à droite une lettre :

- A pour pompe à l'arrêt,
- M pour pompe en marche,
- G pour mode gel : la pompe est en marche,
- F pour mise en marche forcée de la pompe,

En bas à droite un symbol :

- Flèche haut lorsque la température du cumulus augmente,
- Flèche bas lorsque la température du cumulus diminue,
- Signe égal lorsque la température est stable.

Le montage dispose de 3 voyants :

- Voyant rouge : alarme température maximale atteinte,
- Voyant pompe en marche,
- Voyant MLI.

Le montage dispose de 3 touches : "valider" "+" et "-".

Avec ces sondes qui mettent presque 1/2 seconde chacune pour relever une température, période durant lequel il n'est pas évident de générer une interruption, la manière d'entrer dans les menus de configuration obéit au protocole suivant : on commence par un appui long sur la touche "valider". Après un certain temps - le temps qu'il reste pour accomplir le relevé en cours des températures - apparaît le message que nous entrons dans le menu. Il

faut alors rapidement rappuyer brièvement sur la touche "valider", puis à chaque nouvel appui le menu passe en revue les différentes rubriques. A chaque fois il est possible de changer la donnée avec les touches "+" et "-". Il y a un timeout qui fait qu'après un délai d'inactivité le programme revient seul au contexte initial.

Il faut un peu d'entrainement : si le 2ème appuie intervient trop vite ou pas assez, ou trop brièvement ou trop longuement on rate son coup et on revient au contexte.