Montage

15

La température

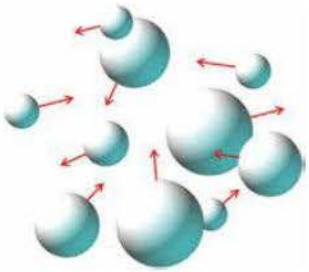
Au sommaire :

- une définition de la température ;
- savoir la mesurer ;
- le sketch complet ;
- la réalisation du circuit ;
- l'ajout d'un ventilateur au circuit ;
- un exercice complémentaire.

Chaud ou froid ?

Nous vivons dans un monde ou plutôt dans un environnement composé de matières diverses. Ces dernières peuvent en principe présenter trois états dits d'agrégation en physique. Un tel état d'agrégation peut être solide, liquide ou gazeux et dépend souvent d'une grandeur physique appelée température. Mais que signifie la température et comment se fait-elle sentir ou plutôt comment peut-on la mesurer ? Toute matière est composée d'infimes particules appelées atomes. Ceux-ci sont composés d'un nuage d'électrons (charge : négative) et d'un noyau formé de protons (charge positive) et de neutrons (charge : nulle). Ce ne sont pas là les plus petites particules, mais elles suffiront à expliquer au moyen de notre exemple ce qu'est la température.

Figure 15-1 ▶
Le mouvement des atomes

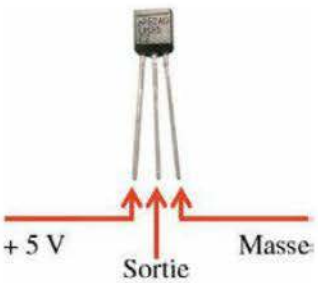


Ces infimes particules sont en perpétuel mouvement, errant apparemment sans but et dans des directions différentes. La température est donc un moyen de mesurer cette agitation thermique des atomes ou des molécules (assemblage de plusieurs atomes) d'une matière. Plus ils se déplacent rapidement, plus la probabilité est grande qu'ils entrent en collision. C'est alors que l'énergie cinétique se transforme en énergie calorifique. L'agitation thermique est donc un moyen de mesurer la température d'une matière.

Comment peut-on mesurer la température ?

On utilise des capteurs, qui convertissent la température mesurée en diverses valeurs de résistance ou de tension, desquelles on peut déduire la température ambiante. Vous avez déjà entendu parler d'une PTC et d'une NTC dans le chapitre 4 sur les bases de l'électronique. La résistance de ces composants varie en fonction de la température. Ils manquent cependant de précision, et leur courbe caractéristique n'est pas forcément linéaire. Aussi voudrais-je vous présenter un capteur de température qui fait fort bien les choses. Il a pour doux nom LM 35 et présente trois pattes de raccordement. Deux d'entre elles servent à l'alimentation, la troisième de sortie. Ce composant ressemble à un transistor au point de les confondre.

Figure 15-2 ▶
Capteur de température LM35 en boîtier plastique TO-92, avec son brochage



Ce capteur convertit la température mesurée en une valeur de tension analogique qui est proportionnelle à la température. Cela s'appelle un comportement de tension proportionnel à la température. Le capteur a une sensibilité de 10 mV/°C et une gamme de température comprise entre 0 et 100 °C. La formule pour calculer la température en fonction de la valeur mesurée à l'entrée analogique est la suivante :

$$Température\ [°C] = \frac{5.0 \cdot 100.0 \cdot analogPin}{1024.0}$$

Les valeurs de la formule se justifient ainsi :

- 5.0 : tension de référence Arduino de 5 V ;
- 100.0 : valeur maximale mesurable par le capteur de température ;
- 1024 : résolution de l'entrée analogique.

Nous allons maintenant envoyer la valeur mesurée à un sketch Processing et représenter graphiquement la courbe de température. Le tout ressemble à peu près à la figure 15-3. La température s'affiche sous la forme d'une valeur de température et sous celle d'une courbe graphique en fonction du temps.

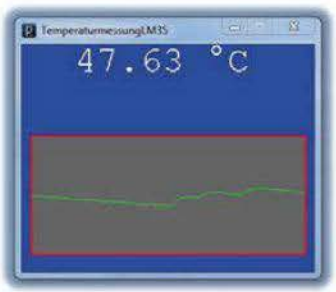


Figure 15-3
Courbe de température dans Processing

Composants nécessaires

	1 capteur de température LM35
	Plusieurs cavaliers flexibles de couleurs et de longueurs diverses

Code du sketch Arduino

```
#define sensorPin 0       //Connexion à la sortie du LM35
#define DELAY 10          //Bref temps d'attente
```

```
const int cycles = 20;  //Nombre de mesures

void setup(){
  Serial.begin(9600);
}
void loop(){
  float resultTemp = 0.0;
  for(int i = 0; i < cycles; i++){
    int analogValue = analogRead(sensorPin);
    float temperature = (5.0 * 100.0 * analogValue) / 1024;
    resultTemp += temperature; //Addition des valeurs mesurées
    delay(DELAY);
  }
  resultTemp /= cycles;        //Calcul de la moyenne
  Serial.println (resultTemp); //Envoi à l'interface série
}
```

Revue de code Arduino

La valeur déterminée par le capteur de température LM35 est calculée avec la formule ci-après :

```
float temperature = (5.0 * 100.0 * analogValue) / 1024;
```

et moyennée à l'aide d'une boucle `for`. Les valeurs mesurées y sont additionnées, puis la moyenne est calculée. Cette dernière est enfin transmise à l'interface série :

```
Serial.println(resultTemp);
```

Son traitement par Processing commence immédiatement.

Revue de code Processing

```
import processing.serial.*;
Serial mySerialPort;
float realTemperature;
int temperature, xPos;
int[] yPos;
PFont font;

void setup(){
  size(321, 250); smooth();
  println(Serial.list());
  mySerialPort = new Serial(this, Serial.list()[0], 9600);
  mySerialPort.bufferUntil('\n');
  yPos = new int[width];
```

```
  for(int i = 0; i < width; i++)
    yPos[i] = 250;
  font = createFont("Courier New", 40, false);
  textFont(font, 40); textAlign(RIGHT);
}

void draw(){
  background(0, 0, 255, 100);
  strokeWeight(2); stroke(255, 0, 0);
  fill(100, 100, 100); rect(10, 100, width - 20, 130);
  strokeWeight(1); stroke(0, 255, 0);
  int yPosPrev = 0, xPosPrev = 0;
  //Décaler les valeurs du tableau vers la gauche
  for(int x = 1; x < width; x++)
    yPos[x-1] = yPos[x];
  //Ajout des nouvelles coordonnées de la souris à
  //l'extrémité droite du tableau
  yPos[width-1] = temperature;
  //Affichage du tableau
  for(int x = 10; x < width - 10 ; x++)
    point(x, yPos[x]);
  fill(255);
  text(realTemperature + "°C", 250, 30);  //Celsius
  delay(100);
}

void serialEvent (Serial mySerialPort){
  String portStream = mySerialPort.readString();
  float data = float(portStream);
  realTemperature = data;
  temperature = heigh - (int)map(data, 0, 100, 0, 130) - 25;
  println(realTemperature);
}
```

Pour aller plus loin

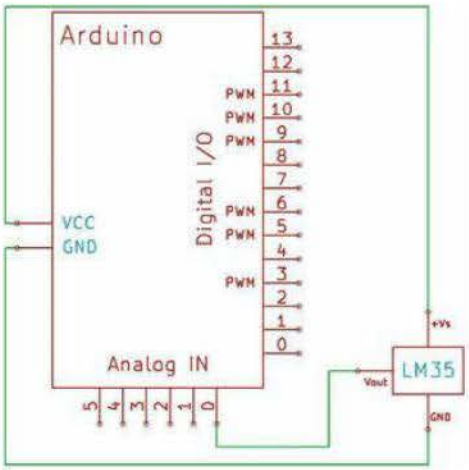
Si vous oubliez de refermer la fenêtre d'affichage de Processing que vous avez ouverte, la communication avec la carte Arduino est impossible. Pourquoi ? Tout simplement parce que Processing accède à l'interface série, dont votre carte Arduino a précisément besoin pour communiquer avec l'environnement de développement !

Ce port est donc bloqué par Processing et ne peut être libéré qu'en fermant la fenêtre d'affichage.

Schéma

Le schéma est, je dois le reconnaître, vraiment simple, mais nous allons bientôt lui ajouter quelque chose qui rendra le circuit plus fonctionnel.

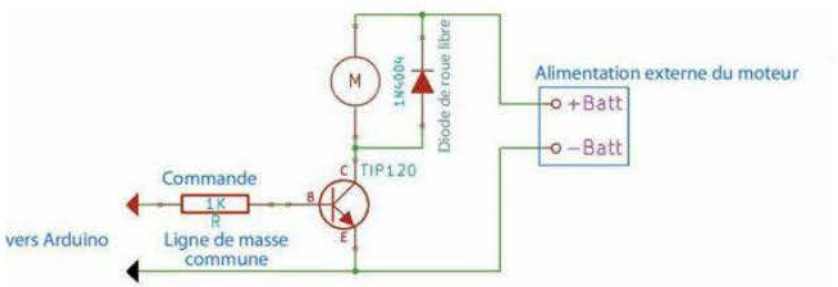
Figure 15-4 ▶
Capteur de température envoyant ses données à une entrée analogique



Sketch élargi (maintenant avec tout le reste)

Il est temps maintenant de construire quelque chose de bien avec le capteur de température. Que diriez-vous d'ajouter directement plusieurs composants au circuit ? Je pense qu'un ventilateur pour améliorer le climat ambiant et un afficheur pour donner les informations utiles seraient des projets intéressants. Le circuit et le sketch doivent être en mesure de mettre en route un moteur de ventilateur quand une certaine température est atteinte et de l'arrêter quand elle ne l'est plus. Nous touchons ici à l'art et la manière de commander un moteur. Ce dernier ayant assurément besoin de plus de courant et de tension pour fonctionner que la carte Arduino ne peut en fournir, il nous faut trouver autre chose. Vous avez appris, dans le chapitre 5 sur les circuits électroniques simples, comment un relais peut être commandé. Si vous remplacez le relais par un moteur, vous obtenez pratiquement une commande de moteur. Voyez la figure 15-5.

Figure 15-5 ▶
Commande d'un moteur



Dans ce circuit, j'ai utilisé un transistor plus fort, de type TIP 120. Il s'agit d'un transistor Darlington de puissance, en boîtier TO-220, capable de commuter un courant de collecteur $I_C = 5$ A et de supporter une tension collecteur-émetteur $U_{CE} = 60$ V.

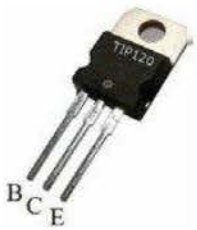


Figure 15-6
Transistor Darlington

La diode de roue libre ne doit pas être oubliée, bien sûr. C'est une 1N4004. Vous souvenez-vous encore pourquoi elle est obligatoire dans ce circuit ? Reportez-vous au chapitre 4 si vous avez un trou de mémoire. Vous devez impérativement utiliser cette diode et veiller à ce que sa polarité soit correcte, faute de quoi votre carte Arduino risque fort d'en pâtir. Nous souhaitons par ailleurs utiliser un affichage LCD pour indiquer la température actuelle. Il s'agit cette fois d'un affichage qui doit être commandé via un bus *I²C* (bus de données série pouvant être relié à différents types d'appareils électroniques). Il est de type I2C/TWILCD1602.

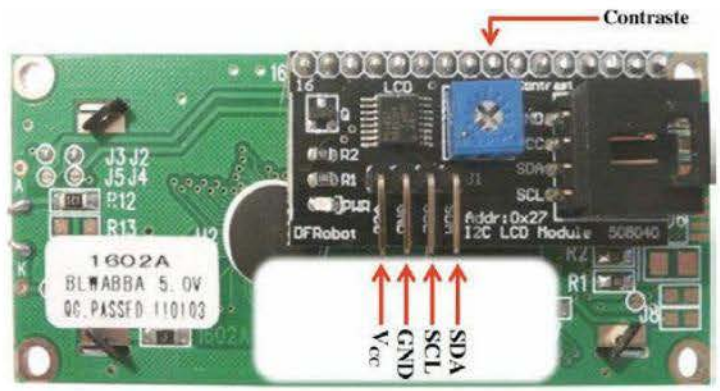


Figure 15-7
Envers d'un affichage LCD 1602

La commande de cet affichage va être présentée au moyen du sketch utilisé. Venons-en maintenant au schéma complet, qui a l'air déjà beaucoup plus conséquent.

Composants nécessaires

1 capteur de température LM35

1 transistor de puissance TIP 120

1 résistance d'1 kΩ

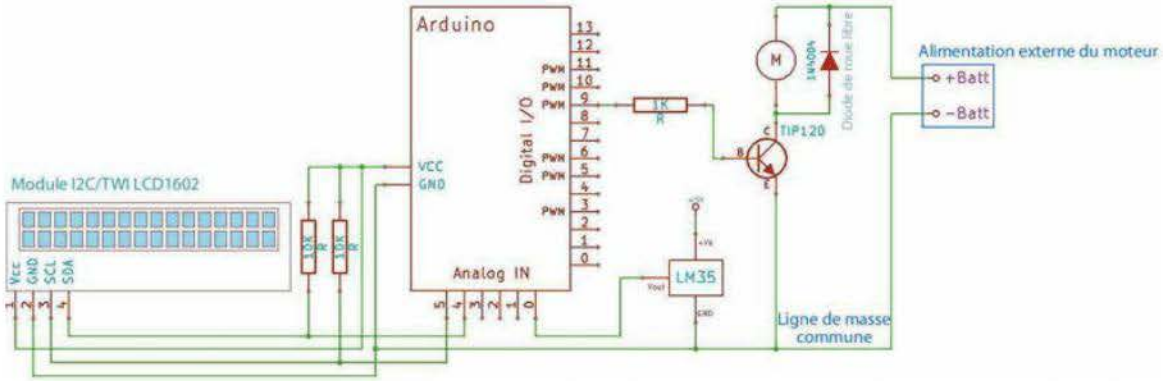
2 résistances de 10 kΩ

1 diode 1N4004

Module afficheur LCD I2C/TWI LCD1602

Moteur de ventilateur, 12 V par exemple

Plusieurs cavaliers flexibles de couleurs et de longueurs diverses



▲ **Figure 15-8**
Circuit complet avec capteur, affichage et moteur ou plutôt ventilateur

Bon ! Nous avons à gauche le LCD I2C avec les résistances pull-up, au centre notre Arduino et à droite le capteur de température LM35. Complètement à droite se trouve la commande du moteur avec le transistor TIP 120 et la diode de roue libre 1N4004. Voyons maintenant le code du sketch :

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#define sensorPin 0       //Connexion à la sortie du LM35
#define DELAY1 10         //Bref temps d'attente lors de la mesure
#define DELAY2 500        //Bref temps d'attente lors de l'affichage
#define motorPin 9        //Broche commande du ventilateur
#define threshold 25      //Température de commutation du ventilateur
                          //(25 degrés Celsius)
#define hysteresis 0.5    //Valeur d'hystérésis (0.5 degré Celsius)
const int cycles = 20;    //Nombre de mesures
LiquidCrystal_I2C lcd(0x27, 16, 2);  //Adresse I2C : 0x27 pour
                                     //16 caractères/2 lignes

void setup(){
   pinMode(motorPin, OUTPUT);
   lcd.init();       //Initialisation du LCD
   lcd.backlight();  //Activation du rétroéclairage
}

void loop(){
   float resultTemp = 0.0;
   for(int i = 0; i < cycles; i++){
      int analogValue = analogRead(sensorPin);
      float temperature = (5.0 * 100.0 * analogValue) / 1024;
      resultTemp += temperature;  //Addition des valeurs mesurées
      delay(DELAY1);
   }
   resultTemp /= cycles; //Calcul de la moyenne
   lcd.clear();    //Méthode clear pour effacer le contenu du LCD
```

```
    lcd.print("Temp: "); //Méthode print pour écrire sur le LCD
    lcd.print(resultTemp);
    #if ARDUINO < 100
    lcd.print(0xDO + 15, BYTE); //Caractère degré (Arduino 0022)
    #else
    lcd.write(0xDO + 15); //Caractère degré (Arduino 1.00)
    #endif
    lcd.print("C");
    lcd.setCursor(0, 1);  //Méthode setCursor pour positionner
                          //le curseur du LCD
    lcd.print("Moteur: ");
    if(resultTemp > (threshold + hysterese))
       digitalWrite(motorPin, HIGH);
    else if(resultTemp < (threshold - hysterese))
       digitalWrite(motorPin, LOW);
    lcd.print(digitalRead(motorPin) == HIGH?"en marche":"stop");
    delay(DELAY2);
}
```

La détermination de la température est effectuée de la même manière et se trouve être la même que dans l'exemple précédent.

Soit vous jouez encore les cachottiers, soit vous avez carrément oublié : dans ce code de sketch se trouve également un élément de programme dont vous ne nous avez pas parlé. Que signifie la ligne `const int cycles = 20;` ? Ce qui me chagrine là-dedans, c'est le mot `const`.

Bien vu, Ardus ! Cette fois, j'allais vraiment oublier ! Je dois ici développer un peu, mais vous allez voir que c'est vraiment facile à comprendre. Nous avons affaire à une autre forme de déclaration de variable. Vous connaissez par conséquent maintenant trois formes écrites, que je vous rappelle au moyen d'un exemple :

```
1. int grandeurs = 47;
2. #define grandeurs 47
3. const int grandeurs = 47;
```

Les trois variantes initialisent visiblement une variable appelée `grandeurs` avec la valeur `47`. Alors où est la différence ? Il doit bien y en avoir une, sinon à quoi bon des formes écrites différentes.

Variante 1

Bon ! La première variante `int grandeurs = 47;` fait réserver par le compilateur un emplacement dans la mémoire vive RAM pour y

consigner la valeur 47. De la mémoire supplémentaire est donc nécessaire et occupée.

Variante 2

Cette variante utilise la directive de prétraitement `#define`, qui affecte une valeur uniquement à un nom, que le compilateur remplace partout dans le code de sketch lors de la transformation. Aucune mémoire supplémentaire n'est ainsi attribuée pour gérer une variable. Mais vous devez vous demander, pour cette forme écrite, quel type de donnée est employé, car il n'est pas indiqué comme dans le premier exemple. Duquel pourrait-il bien s'agir ?

Variante 3

Si le mot-clé `const` est utilisé devant la déclaration de variable, alors la variable en question n'est plus une variable mais une constante, dont la valeur ne peut plus être modifiée pour la durée du sketch. Il s'agit presque d'une variable en *lecture seule*. Qu'en pensez-vous maintenant si je vous dis que cette variante n'utilise pas de mémoire ? On est sûr que la variable ne sera pas modifiée, aussi pourquoi en occuperait-elle ? Mais en quoi est-elle différente de la variante `#define` ? C'est très simple : on peut indiquer ici un certain type de donnée.

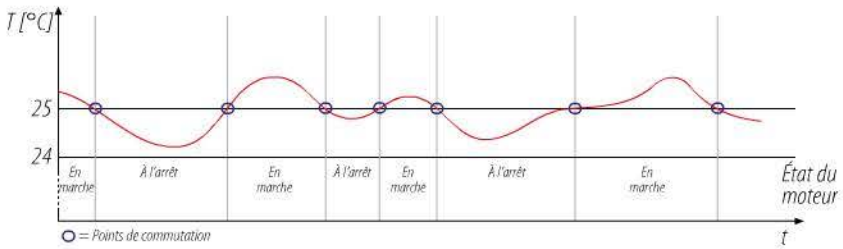
Sur Internet, tout comme dans de nombreux livres, on passe sans cesse de l'une à l'autre des trois possibilités. De quelle variante faut-il se servir ? Si la mémoire est juste et si une indication explicite du type de donnée est nécessaire, la variante 3 est alors recommandée. Revenons maintenant à notre circuit. Le mieux est de vous montrer l'afficheur LCD.

Vous pouvez y lire sans problème la température et l'état du moteur.

Holà, stop, arrêtez ! J'ai bien compris jusqu'ici le fonctionnement du sketch, mais je n'ai aucune idée de ce qu'est une hystérésis.

Vous ne pouvez rien me reprocher cette fois, car j'allais y venir. Imaginez la situation suivante : le ventilateur doit, tout comme dans notre exemple, se mettre en marche à 25 °C et apporter un peu d'air frais à ceux qui transpirent sur leur Arduino. La température ambiante n'étant cependant pas constante à 100 %, le capteur est lui aussi soumis à certaines fluctuations. Un état est donc par exemple atteint, dans lequel la température mesurée varie constamment entre 24,8 et 25,2 °C. Autrement dit, le ventilateur n'arrête pas de s'allumer et de s'éteindre. Plutôt énervant à la longue ! Voyons maintenant la figure 15-9 de plus près.

Figure 15-9 ▶
En cas de température fluctuant autour de la valeur de consigne, l'état du moteur change constamment.



C'est là que l'*hystérésis* (le mot vient du grec et signifie retard) entre en jeu. On peut expliquer ainsi le comportement d'une régulation avec hystérésis : la variable de sortie, qui commande ici le moteur, ne dépend pas seulement de la variable d'entrée délivrée par le capteur. L'état de la variable de sortie, qui régnait auparavant, joue aussi un rôle important. Dans notre exemple, nous avons une valeur-seuil de 25 °C et une hystérésis de 0,5 °C. Voyons maintenant de plus près la régulation du ventilateur :

```
if(resultTemp > (threshold + hysterese))
    digitalWrite(motorPin, HIGH);
else if(resultTemp < (threshold - hysterese))
    digitalWrite(motorPin, LOW);
```

Quand le ventilateur se met-il en marche ?

Si la condition :

```
(resultTemp > (threshold + hysterese)) …
```

est remplie, le ventilateur commence à tourner. C'est le cas ici quand la température mesurée est supérieure à 25 + 0,5 °C.

Quand le ventilateur s'arrête-t-il ?

Si la condition :

```
resultTemp < (threshold - hysterese)
```

est remplie, le ventilateur s'arrête de tourner, ici en l'occurrence quand la température est inférieure à 25 – 0,5 °C. Pour résumer :

- le ventilateur est en marche si : température > 25,5 °C ;
- le ventilateur est à l'arrêt si : température < 24,5 °C.

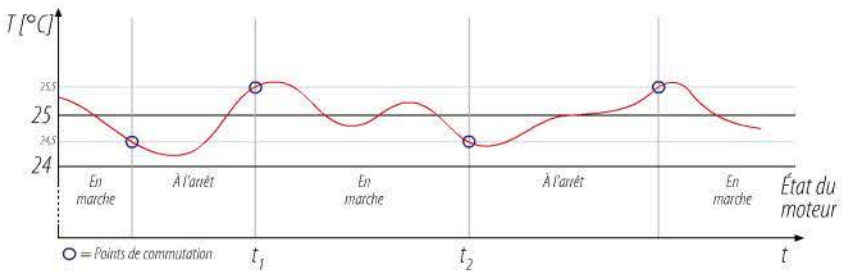


Figure 15-10
En cas de fluctuation de la température autour de la valeur de consigne, l'état du moteur ne change pas constamment.

Si vous regardez la courbe entre les points t_1 et t_2, vous verrez que la température passe constamment au-dessus et en dessous des 25 °C. Sans commande avec hystérésis, nous aurions sans cesse *moteur en marche* et *moteur à l'arrêt*. La réalisation complète du circuit est donc celle de la figure 15-11.

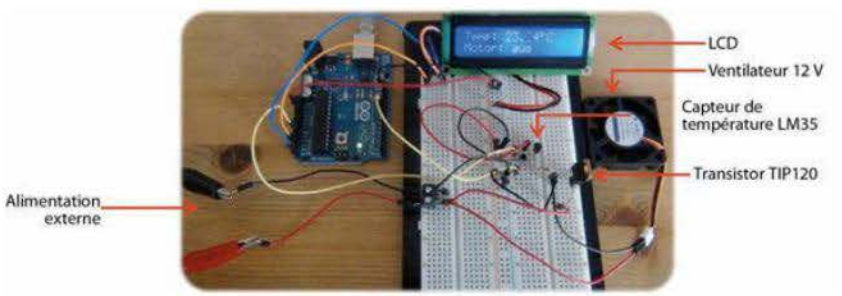


Figure 15-11
Réalisation complète du circuit

Attention !
Vous devez être particulièrement soigneux du fait que vous travaillez avec une source de tension externe. Comme je l'ai déjà expliqué, vous devez raccorder ensemble les deux points de masse de la carte Arduino et de la source de tension externe. Mais surtout pas les potentiels positifs ! Vous ne devez en aucun cas confondre ces deux potentiels et devez veiller à ce qu'aucun court-circuit ne se produise. Vérifiez le câblage du circuit avant de tout mettre en marche. Mieux vaut trop que pas assez...

Problèmes courants

Si le ventilateur ne se met pas en marche alors que la température-seuil plus la valeur d'hystérésis est atteinte, éteignez tout et vérifiez ce qui suit.

- Le câblage est-il correct ?
- Pas de court-circuit éventuel ?
- La masse commune à la carte Arduino et à la source de tension externe est-elle établie ?
- La diode de roue libre est-elle montée dans le bon sens ?
- Si on ne voit rien sur le LCD, le contraste n'est-il pas trop faible ?

Qu'avez-vous appris ?

- Dans ce montage, vous avez appris comment le capteur de température fonctionne et convertit des valeurs de température en valeurs de tension correspondantes, qui peuvent être exploitées à l'entrée analogique de votre carte Arduino.
- Vous avez utilisé un affichage I2C/TWI LCD1602, commandé via le bus I^2C, pour indiquer la valeur de température.
- Pour que le ventilateur fonctionne correctement, vous avez dû recourir à une alimentation externe, elle-même connectée au transistor de puissance TIP 120.
- Vous avez appris comment une diode 1N4004 sert, en tant que diode de roue libre, à protéger votre carte Arduino.

Exercice complémentaire

Agrandissez votre circuit de manière à pouvoir augmenter ou diminuer la valeur-seuil au moyen par exemple de deux boutons-poussoirs supplémentaires. Le LCD est censé attirer l'attention en se mettant à clignoter une fois cette valeur-seuil atteinte. Pour plus d'informations sur la bibliothèque et sur la gamme d'instructions du LCD, recherchez les mots suivants sur Google :

- I2C/TWI LCD 1602 ;
- Dfrobot.

Pour aller plus loin

Il existe bien entendu beaucoup d'autres capteurs de température. En voici une sélection :

- TMP75 (avec bus I²C) ;
- AD22100 (capteur de température analogique) ;
- DHT11 (capteur de température et humidité avec microcontrôleur 8 bits intégré) ;
- DS1820 (capteur de température numérique 1-Wire).