Capteurs DHT11 et DHT22

Ce capteur permet de mesurer la température et l’humidité. Trois broches : VCC, DATA et GND. La broche DATA doit être connectée sur une entrée numérique de l’Arduino (Ex, broche 7).

Plages DHT11:

Température : 0 à 50 °C   , précision 2 °C
Humidité   : 20 à 90 %   , précision 5 %

Plages DHT22 :

Température : -40 °c à 80°C , précision 0.5 °C
Humidité   : 0 à 100 %   , précision de 2à 5 %

Code :

Fichier dht1122.h

```
#ifndef dht1122_h
#define dht1122_h
/*
 * Source : https://www.carnetdumaker.net/
 *
*/
#include <Arduino.h>

class dht1122
{
 public:
 static const byte DHT_SUCCESS=0;  // OK
 static const byte DHT_TIMEOUT_ERROR = 1;  // Temps d'attente dépassé
 static const byte DHT_CHECKSUM_ERROR = 2; // Données reçues erronées
 static byte readDHT11(byte pin, float* temperature, float* humidity);
 static byte readDHT22(byte pin, float* temperature, float* humidity);
 private:
 static byte readDHTxx(byte pin, byte* data, unsigned long start_time, unsigned
long timeout);
};
#endif
```

Fichier dht1122.cpp

```
#include <Arduino.h>
#include "dht1122.h"
/**
 * Lit la température et le taux d'humidité mesuré par un capteur DHT11.
 *
 * @param pin Broche sur laquelle est câblée le capteur.
 * @param temperature Pointeur vers la variable stockant la température.
 * @param humidity Pointeur vers la variable stockant le taux d'humidité.
 * @return DHT_SUCCESS si aucune erreur, DHT_TIMEOUT_ERROR en cas de timeout, ou
DHT_CHECKSUM_ERROR en cas d'erreur de checksum.
 */
byte dht1122::readDHT11(byte pin, float* temperature, float* humidity) {

  /* Lit le capteur */
  byte data[5];
  byte ret = readDHTxx(pin, data, 18, 1000);

  /* Détecte et retourne les erreurs de communication */
```

```
  if (ret != DHT_SUCCESS)
    return ret;

  /* Calcul la vraie valeur de la température et de l'humidité */
  *humidity = data[0];
  *temperature = data[2];
  /* Ok */
  return DHT_SUCCESS;
}
/**
 * Lit la température et le taux d'humidité mesuré par un capteur DHT22.
 *
 * @param pin Broche sur laquelle est câblée le capteur.
 * @param temperature Pointeur vers la variable stockant la température.
 * @param humidity Pointeur vers la variable stockant le taux d'humidité.
 * @return DHT_SUCCESS si aucune erreur, DHT_TIMEOUT_ERROR en cas de timeout, ou
DHT_CHECKSUM_ERROR en cas d'erreur de checksum.
 */
byte dht1122::readDHT22(byte pin, float* temperature, float* humidity) {

  /* Lit le capteur */
  byte data[5];
  byte ret = readDHTxx(pin, data, 1, 1000);

  /* Détecte et retourne les erreurs de communication */
  if (ret != DHT_SUCCESS)
    return ret;

  /* Calcul la vraie valeur de la température et de l'humidité */
  float fh = data[0];
  fh *= 256;
  fh += data[1];
  fh *= 0.1;
  *humidity = fh;

  float ft = data[2] & 0x7f;
  ft *= 256;
  ft += data[3];
  ft *= 0.1;
  if (data[2] & 0x80) {
    ft *= -1;
  }
  *temperature = ft;

  /* Ok */
  return DHT_SUCCESS;
}

/**
 * Fonction bas niveau permettant de lire la température et le taux d'humidité
(en valeurs brutes) mesuré par un capteur DHTxx.
 */
byte dht1122::readDHTxx(byte pin, byte* data, unsigned long start_time, unsigned
long timeout) {
  data[0] = data[1] = data[2] = data[3] = data[4] = 0;
  // start_time est en millisecondes
  // timeout est en microsecondes

  /* Conversion du numéro de broche Arduino en ports / masque binaire "bas
niveau" */
  uint8_t bit = digitalPinToBitMask(pin);
  uint8_t port = digitalPinToPort(pin);
  volatile uint8_t *ddr = portModeRegister(port);   // Registre MODE (INPUT /
OUTPUT)
```

```
  volatile uint8_t *out = portOutputRegister(port); // Registre OUT (écriture)
  volatile uint8_t *in = portInputRegister(port);   // Registre IN (lecture)

  /* Conversion du temps de timeout en nombre de cycles processeur */
  unsigned long max_cycles = microsecondsToClockCycles(timeout);

  /* Evite les problèmes de pull-up */
  *out |= bit;  // PULLUP
  *ddr &= ~bit; // INPUT
  delay(100);   // Laisse le temps à la résistance de pullup de mettre la ligne
de données à HIGH

  /* Réveil du capteur */
  *ddr |= bit;  // OUTPUT
  *out &= ~bit; // LOW
  delay(start_time); // Temps d'attente à LOW causant le réveil du capteur
  // N.B. Il est impossible d'utilise delayMicroseconds() ici car un délai
  // de plus de 16 millisecondes ne donne pas un timing assez précis.

  /* Portion de code critique - pas d'interruptions possibles */
  noInterrupts();

  /* Passage en écoute */
  *out |= bit;  // PULLUP
  delayMicroseconds(40);
  *ddr &= ~bit; // INPUT

  /* Attente de la réponse du capteur */
  timeout = 0;
  while(!(*in & bit)) { /* Attente d'un état LOW */
    if (++timeout == max_cycles) {
        interrupts();
        return DHT_TIMEOUT_ERROR;
      }
  }

  timeout = 0;
  while(*in & bit) { /* Attente d'un état HIGH */
    if (++timeout == max_cycles) {
        interrupts();
        return DHT_TIMEOUT_ERROR;
      }
  }

  /* Lecture des données du capteur (40 bits) */
  for (byte i = 0; i < 40; ++i) {

    /* Attente d'un état LOW */
    unsigned long cycles_low = 0;
    while(!(*in & bit)) {
      if (++cycles_low == max_cycles) {
        interrupts();
        return DHT_TIMEOUT_ERROR;
      }
    }

    /* Attente d'un état HIGH */
    unsigned long cycles_high = 0;
    while(*in & bit) {
      if (++cycles_high == max_cycles) {
        interrupts();
        return DHT_TIMEOUT_ERROR;
      }
    }
```

```
    /* Si le temps haut est supérieur au temps bas c'est un "1", sinon c'est un
"0" */
    data[i / 8] <<= 1;
    if (cycles_high > cycles_low) {
      data[i / 8] |= 1;
    }
  }

  /* Fin de la portion de code critique */
  interrupts();

  /*
   * Format des données :
   * [1, 0] = humidité en %
   * [3, 2] = température en degrés Celsius
   * [4] = checksum (humidité + température)
   */

  /* Vérifie la checksum */
  byte checksum = (data[0] + data[1] + data[2] + data[3]) & 0xff;
  if (data[4] != checksum)
    return DHT_CHECKSUM_ERROR; /* Erreur de checksum */
  else
    return DHT_SUCCESS;
}
```

Programme de Test

```
/**
   Exemple de code pour la lecture d'un capteur DHT11 ou DHT22.
*/
#include <dht1122.h>

const byte BROCHE_CAPTEUR = 7;

void setup()
{
  Serial.begin(9600);
  Serial.println("\nDemo DHT11 avec methodes statiques");
  pinMode(BROCHE_CAPTEUR, INPUT_PULLUP);
}

void loop()
{
  float temperature, humidity;
  // N.B. Remplacer readDHT11 par readDHT22 en fonction du capteur utilisé !
  switch (dht1122::readDHT11(BROCHE_CAPTEUR, &temperature, &humidity))
  {
    case dht1122::DHT_SUCCESS:
      Serial.print("Temperature ");
      Serial.print(temperature);
      Serial.print("° Humidite ");
      Serial.print(humidity);
      Serial.println("%");
      break;
    case dht1122::DHT_TIMEOUT_ERROR:
      Serial.println("Pas de reponse !");
      break;
    case dht1122::DHT_CHECKSUM_ERROR:
      Serial.println("Pb de communication !");
      break;
  }
  /* Pas plus d'une mesure par seconde */
```

```
  // N.B. Avec le DHT22 il est possible de réaliser deux mesures par seconde
  delay(60000);
}
```