Gros probléme d'emails avec Yahoo (/annonces/gros-probleme-demails-avec-yahoo/)

Yahoo refuse tous les emails du site. Si vous avez une adresse chez un autre prestataire, c'est le moment de l'utiliser 😉

En cas de soucis, n'hésitez pas à aller faire un tour sur la page de contact en bas de page.

par skywodd (/membres/skywodd/), le Feb. 21, 2017 at 7:27 p.m.

Mesurer une température avec un capteur 1-Wire DS18B20 et une carte Arduino / Genuino

Chaud devant

par skywodd (/membres/skywodd/) | Oct. 22, 2016 | © Licence (voir pied de page)

Catégories : Tutoriels (/articles/categories/tutoriels/) Arduino (/articles/categories/tutoriels/arduino/) | Mots clefs : Arduino (/articles/tags/arduino/) Genuino (/articles/tags/genuino/) Capteur (/articles/tags/capteur/) Température (/articles/tags/temperature/) OneWire (/articles/tags/onewire/) 1-Wire (/articles/tags/1-wire/) DS18B20 (/articles/tags/ds18b20/)

Cet article a été modifié pour la dernière fois le Oct. 24, 2016 at 10:36 a.m.

Dans ce tutoriel, nous allons apprendre ensemble à utiliser un capteur 1-Wire DS18B20 pour mesurer une température au moyen d'une carte Arduino / Genuino. Nous verrons aussi comment mesurer plusieurs températures en même temps avec plusieurs capteurs sur un même bus. En bonus, on verra comment faire plusieurs mesures en parallèle et comment réduire le temps entre deux mesures.

Sommaire

Bonjour à toutes et à tous !

Dans un précédent tutoriel (https://www.carnetdumaker.net/articles/mesurer-une-temperature-avec-un-capteur-lm35-et-une-carte-arduino-genuino/), je vous avais expliqué comment utiliser des capteurs analogiques de température. Ceux-ci sont fiables et faciles à mettre en oeuvre, mais ils ne sont pas exempts de défauts. Pour commencer, il faut une entrée analogique par capteur. Ensuite, il faut restreindre la longueur de câble pour éviter les parasites. Pour finir, il est nécessaire d'ajouter un peu d'électronique pour faire fonctionner chaque capteur.

On peut donc légitimement se poser la question suivante : existe-t-il des capteurs de température numériques, avec toute l'électronique de mesure intégrée dans un même composant ? La réponse est oui.

Il existe diverses références de capteurs de température numériques, celle que je vais vous présenter dans ce tutoriel est la plus connue : le capteur DS18B20 du fabricant Maxim (https://www.maximintegrated.com/en/products/analog/sensors-and-sensor-interface/DS18B20.html).

Le capteur DS18B20

(https://www.carnetdumaker.net/images/capteur-ds18b20-en-boitier-92/)

Capteur DS18B20 en boitier TO-92

Le capteur DS18B20 (http://datasheets.maxim-ic.com/en/ds/DS18B20.pdf) du fabricant Maxim (anciennement Dallas Semiconducteur) est un capteur de température numérique intégrant tout le nécessaire requis pour faire la mesure : capteur analogique, convertisseur analogique / numérique, électronique de communication et alimentation.

Il communique via un bus 1-Wire (https://fr.wikipedia.org/wiki/1-Wire) et possède une résolution numérique de 12 bits (programmable, voir chapitre bonus) avec une plage de mesure de -55°C à +125°C. La précision analogique du capteur est de 0,5°C entre -10°C et +85°C, ce qui rend ce capteur très intéressant pour une utilisation "normale".

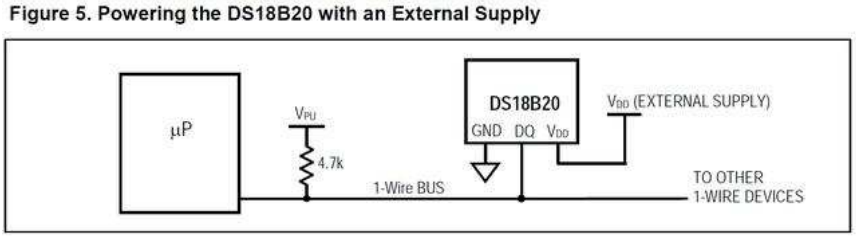
(https://www.carnetdumaker.net/images/capteur-ds18b20-en-format-sonde-etanche/)

Capteur DS18B20 en format "sonde" étanche

Le capteur DS18B20 existe dans le commerce en deux versions : en boitier TO-92 (format transistor, en photo un peu plus haut) pour des utilisations standards en intérieur, ou en format "sonde étanche" pour des applications en milieu humide / extérieur.

N.B. Les deux formats sont strictement identiques d'un point de vue technique.

Câblage du capteur



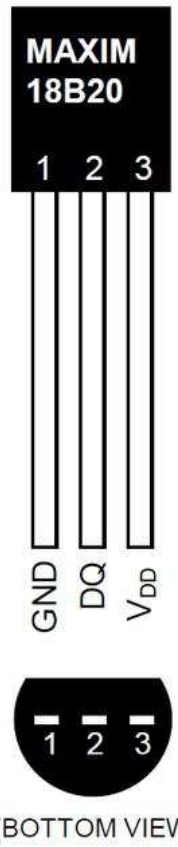
(https://www.carnetdumaker.net/images/cablage-dun-capteur-ds18b20/)

Illustration du câblage

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.

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.

PS Il existe aussi un mode "parasite" ne nécessitant que deux fils (masse et données), mais cela ne sera pas traité dans l'article. De plus, le mode parasite n'est pas le plus adapté pour faire de la mesure.



(https://www.carnetdumaker.net/images/pinout-dun-capteur-ds18b20/)

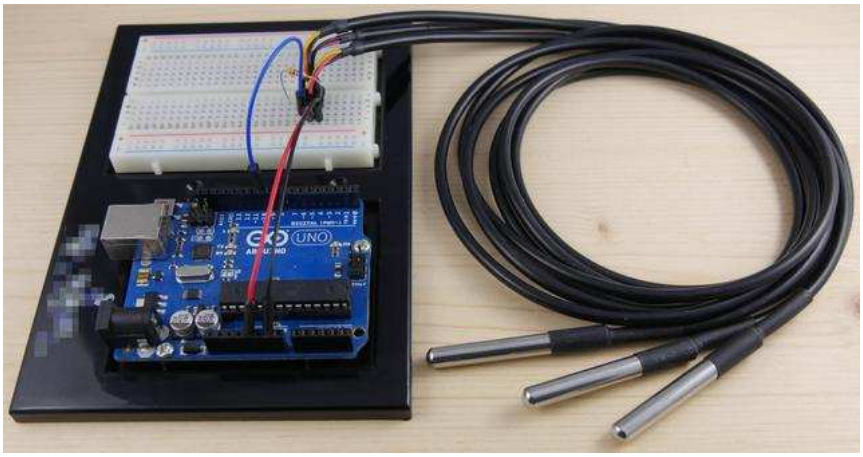
Illustration du pinout

Pour les utilisateurs de la version en boitier TO-92, la broche de données est toujours au centre du composant. En prenant le composant avec le méplat face à vous, la masse est à gauche et l'alimentation à droite.

Pour les utilisateurs de la version "sonde", le fil rouge représente l'alimentation, le fil noir représente la masse et le fil jaune représente la broche de données.

Le montage

Le montage du capteur avec une carte Arduino est relativement simple à mettre en oeuvre.

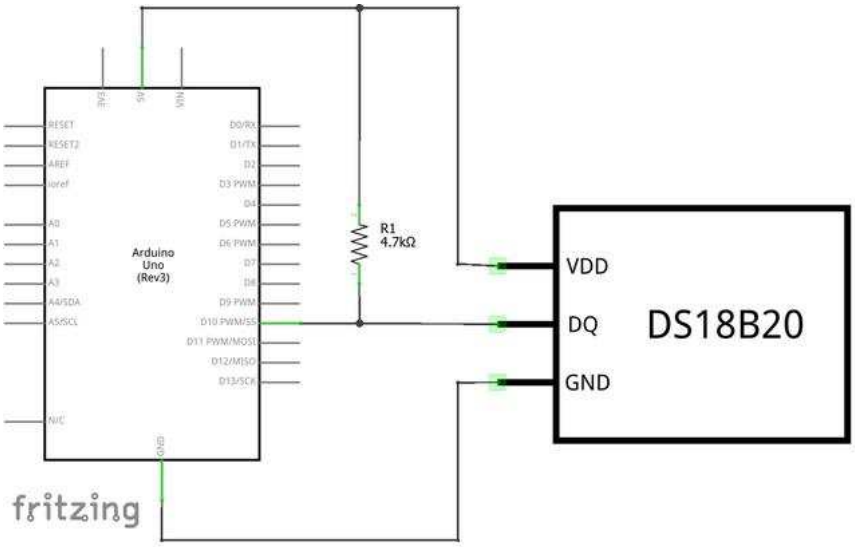


(https://www.carnetdumaker.net/images/photographie-du-montage-de-test-pour-le-scanneur-1-wire-arduino/)

Le montage

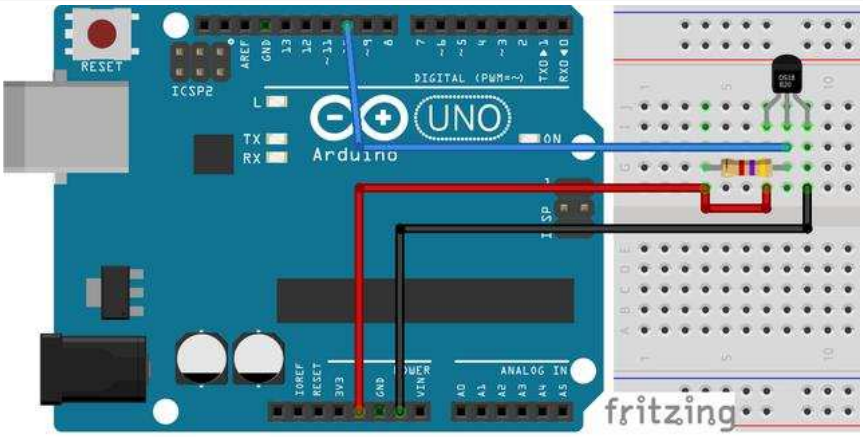
Pour réaliser ce montage, il va nous falloir :

- Une carte Arduino UNO (et son câble USB),
- Une résistance de 4.7K ohms, code couleur jaune – violet – rouge,
- Un ou plusieurs capteurs DS18B20,
- Une plaque d'essai et des fils pour câbler notre montage.



(https://www.carnetdumaker.net/images/schema-du-montage-de-test-du-scanneur-de-bus-1-wire-arduino/)

Schéma du montage



(https://www.carnetdumaker.net/images/vue-prototypage-du-montage-pour-capteur-1-wire-ds18b20-avec-une-arduino/)

Vue prototypage du montage

On commence le montage en reliant ensemble la masse des capteurs avec la masse `GND` de la carte Arduino. On fait ensuite de même avec l'alimentation `5V` de la carte Arduino et l'alimentation des capteurs.

Il ne reste alors plus qu'à relier la broche de données des capteurs à une broche de la carte Arduino. Pour cet article, j'utilise la broche `D10` de la carte Arduino, mais vous pouvez utiliser n'importe quelle autre broche si vous le souhaitez.

Pour finir le montage, il convient de placer une résistance de 4.7K ohms entre l'alimentation `5V` de la carte Arduino et la broche de données du bus 1-Wire.

N.B. La valeur de cette résistance est importante. Ce doit être une résistance de 4.7K ohms. Une résistance plus grande limitera le bon fonctionnement des capteurs 1-Wire et une résistance plus faible endommagera les capteurs 1-Wire.

La mémoire du DS18B20

Figure 7. DS18B20 Memory Map

SCRATCHPAD (POWER-UP STATE)		
Byte 0	Temperature LSB (50h)	(85°C)
Byte 1	Temperature MSB (05h)	
Byte 2	T_H Register or User Byte 1*	
Byte 3	T_L Register or User Byte 2*	
Byte 4	Configuration Register*	
Byte 5	Reserved (FFh)	
Byte 6	Reserved	
Byte 7	Reserved (10h)	
Byte 8	CRC*	

*Power-up state depends on value(s) stored in EEPROM.

(https://www.carnetdumaker.net/images/scratchpad-du-capteur-ds18b20/)

Structure du scratchpad

Comme tout périphérique 1-Wire, le DS18B20 contient un "scratchpad" qui est une sorte de mémoire tampon sécurisée ou l'on peut venir lire et / ou écrire des données. C'est dans cette mémoire qu'on vient lire les données de mesures et écrire les informations de configuration du capteur.

Le scratchpad du capteur DS12B20 est divisé en quatre parties :

- Le résultat de la dernière mesure de température (deux octets),
- Deux octets à usages divers (le capteur dispose d'un mode "alarme", mais cela ne sera pas traité dans ce tutoriel),
- Le registre de configuration du capteur,
- Une somme de contrôle.

Ce qui nous intéresse, ce sont les deux premiers octets qui contiennent le résultat de la mesure de température. Le reste ne nous intéresse pas.

Figure 2. Temperature Register Format

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
LS BYTE	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
MS BYTE	S	S	S	S	S	2^6	2^5	2^4

S = SIGN

(https://www.carnetdumaker.net/images/illustration-de-la-structure-des-registres-du-capteur-ds18b20/)

Structure du registre de température

Table 1. Temperature/Data Relationship

TEMPERATURE (°C)	DIGITAL OUTPUT (BINARY)	DIGITAL OUTPUT (HEX)
+125	0000 0111 1101 0000	07D0h
+85*	0000 0101 0101 0000	0550h
+25.0625	0000 0001 1001 0001	0191h
+10.125	0000 0000 1010 0010	00A2h
+0.5	0000 0000 0000 1000	0008h
0	0000 0000 0000 0000	0000h
-0.5	1111 1111 1111 1000	FFF8h
-10.125	1111 1111 0101 1110	FF5Eh
-25.0625	1111 1110 0110 1111	FE6Fh
-55	1111 1100 1001 0000	FC90h

*The power-on reset value of the temperature register is +85°C.

(https://www.carnetdumaker.net/images/illustration-de-la-structure-des-registres-du-capteur-ds18b20-suite/)

Structure du registre de température (suite)

Les deux octets forment un nombre flottant sur 11 bits, plus le signe (`+` / `-`).

Pour avoir la température en degré Celsius il faut appliquer la formule suivante : `temperature_celsius = ((MSB << 8) + LSB) * 0.0625`.

N.B. A noter que le calcul doit être réalisé avec des nombres signés (pouvant être négatif ou positif), faut de quoi, les températures négatives créeront des problèmes.

PS On verra dans les chapitres bonus d'où provient cette constante `0.0625` 😉

Lire un ou plusieurs capteurs par recherche

Commençons par le cas d'usage le plus simple : vous avec un unique capteur DS18B20 sur votre bus 1-Wire.

```
1  /* Dépendance pour le bus 1-Wire */
2  #include <OneWire.h>
```

Tout d'abord on commence notre code en important la bibliothèque de code OneWire (https://www.pjrc.com/teensy/td_libs_OneWire.html).

N.B. Cette bibliothèque de code n'est pas fournie de base avec le logiciel Arduino, il vous faudra l'installer par vous même en suivant les instructions dans le lien un peu plus haut.

```
1  /* Broche du bus 1-Wire */
2  const byte BROCHE_ONEWIRE = 7;
3
4  /* Code de retour de la fonction getTemperature() */
5  enum DS18B20_RCODES {
6    READ_OK,  // Lecture ok
7    NO_SENSOR_FOUND,  // Pas de capteur
8    INVALID_ADDRESS,  // Adresse reçue invalide
9    INVALID_SENSOR  // Capteur invalide (pas un DS18B20)
10 };
```

On déclare ensuite deux constantes. Une première constante pour le numéro de broche de notre bus 1-Wire. Et une seconde constante (une énumération de constantes pour être exact) contenant les divers codes d'erreurs que la fonction de lecture de la température peut retourner.

```
1  /* Création de l'objet OneWire pour manipuler le bus 1-Wire */
2  OneWire ds(BROCHE_ONEWIRE);
```

On crée ensuite notre objet OneWire qui permettra de communiquer avec les capteurs.

PS Plusieurs bus 1-Wire peuvent être créés en parallèle sur différentes broches. N'hésitez donc pas à faire plusieurs bus pour chaque type de périphérique 1-Wire utilisé. Cela vous rendra la vie plus facile.

```
1  /** Fonction setup() **/
2  void setup() {
3
4    /* Initialisation du port série */
5    Serial.begin(115200);
6  }
```

Vient ensuite la fonction `setup()` qui ne fait qu'initialiser le port série pour que l'on puisse avoir un retour de la température lue.

N.B. Le (ou les) bus 1-Wire s'initialise(nt) d'eux même lors de la création de l'objet OneWire.

```
1  /** Fonction loop() **/
2  void loop() {
3    float temperature;
4
5    /* Lit la température ambiante à ~1Hz */
6    if (getTemperature(&temperature, true) != READ_OK) {
7      Serial.println(F("Erreur de lecture du capteur"));
8      return;
9    }
10
11   /* Affiche la température */
12   Serial.print(F("Temperature : "));
13   Serial.print(temperature, 2);
14   Serial.write(176); // Caractère degré
15   Serial.write('C');
16   Serial.println();
17 }
```

Vient ensuite la fonction `loop()` qui permet de lire la température du capteur et de l'afficher sur le port série.

```
/**
 * Fonction de lecture de la température via un capteur DS18B20.
 */
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 bus 1-Wire ci nécessaire (requis pour la lecture du premier capteur) */
  if (reset_search) {
    ds.reset_search();
  }

  /* Recherche le prochain capteur 1-Wire disponible */
  if (!ds.search(addr)) {
    // Pas de capteur
    return NO_SENSOR_FOUND;
  }

  /* Vérifie que l'adresse a été correctement reçue */
  if (OneWire::crc8(addr, 7) != addr[7]) {
    // Adresse invalide
    return INVALID_ADDRESS;
  }

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

  /* Reset le bus 1-Wire et sélectionne le capteur */
  ds.reset();
  ds.select(addr);

  /* Lance une prise de mesure de température et attend la fin de la mesure */
  ds.write(0x44, 1);
  delay(800);

  /* Reset le bus 1-Wire, sélectionne le capteur et envoie une demande de lecture du scratchpad */
  ds.reset();
  ds.select(addr);
  ds.write(0xBE);

 /* Lecture du scratchpad */
  for (byte i = 0; i < 9; i++) {
    data[i] = ds.read();
  }

  /* Calcul de la température en degré Celsius */
  *temperature = ((data[1] << 8) | data[0]) * 0.0625;

  // Pas d'erreur
  return READ_OK;
}
```

Vient ensuite le gros du code : la fonction `getTemperature()`.

Celle-ci prend en argument un pointeur vers une variable de type `float` qui servira à stocker la température lue, ainsi qu'un booléen permettant de savoir s'il est nécessaire de redémarrer la recherche de capteurs du début. En retour, la fonction retourne un code d'erreur.

L'algorithme de cette fonction est comme suit :

- Si le booléen `reset_search` est vrai, on recommence la recherche de capteur de zéro. Ce booléen doit être à `true` lors de la lecture du premier capteur.
- Ensuite on cherche un capteur sur le bus 1-Wire.
- On vérifie que l'adresse reçue est correcte et qu'il s'agit bien d'un DS18B20.

  PS Comme vous pouvez le voir, le fait d'avoir un bus 1-Wire par type de capteurs permet de simplifier énormément le code.
- On sélectionne le capteur trouvé et on lui envoie la commande magique `0x44` qui déclenche une prise de mesure.
- On attend 800 millisecondes le temps que la mesure se fasse.
- On resélectionne le capteur et cette fois on lui envoie la commande `0xBE` pour lire le scratchpad.
- On lit le scratchpad et on calcul la valeur de la température.

Le code complet avec commentaires :

```
/**
 * Exemple de code pour lire un unique capteur DS18B20 sur un bus 1-Wire.
 */

/* Dépendance pour le bus 1-Wire */
```

```
#include <OneWire.h>


/* Broche du bus 1-Wire */
const byte BROCHE_ONEWIRE = 7;

/* Code de retour de la fonction getTemperature() */
enum DS18B20_RCODES {
  READ_OK,  // Lecture ok
  NO_SENSOR_FOUND,  // Pas de capteur
  INVALID_ADDRESS,  // Adresse reçue invalide
  INVALID_SENSOR  // Capteur invalide (pas un DS18B20)
};


/* Création de l'objet OneWire pour manipuler le bus 1-Wire */
OneWire ds(BROCHE_ONEWIRE);


/**
 * Fonction de lecture de la température via un capteur DS18B20.
 */
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 bus 1-Wire ci nécessaire (requis pour la lecture du premier capteur) */
  if (reset_search) {
    ds.reset_search();
  }

  /* Recherche le prochain capteur 1-Wire disponible */
  if (!ds.search(addr)) {
    // Pas de capteur
    return NO_SENSOR_FOUND;
  }

  /* Vérifie que l'adresse a été correctement reçue */
  if (OneWire::crc8(addr, 7) != addr[7]) {
    // Adresse invalide
    return INVALID_ADDRESS;
  }

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

  /* Reset le bus 1-Wire et sélectionne le capteur */
  ds.reset();
  ds.select(addr);

  /* Lance une prise de mesure de température et attend la fin de la mesure */
  ds.write(0x44, 1);
  delay(800);

  /* Reset le bus 1-Wire, sélectionne le capteur et envoie une demande de lecture du scratchpad */
  ds.reset();
  ds.select(addr);
  ds.write(0xBE);

 /* Lecture du scratchpad */
  for (byte i = 0; i < 9; i++) {
    data[i] = ds.read();
  }

  /* Calcul de la température en degré Celsius */
  *temperature = ((data[1] << 8) | data[0]) * 0.0625;

  // Pas d'erreur
  return READ_OK;
}


/** Fonction setup() **/
void setup() {

  /* Initialisation du port série */
  Serial.begin(115200);
}


/** Fonction loop() **/
void loop() {
  float temperature;
```

```
93
94    /* Lit la température ambiante à ~1Hz */
95    if (getTemperature(&temperature, true) != READ_OK) {
96      Serial.println(F("Erreur de lecture du capteur"));
97      return;
98    }
99
100   /* Affiche la température */
101   Serial.print(F("Temperature : "));
102   Serial.print(temperature, 2);
103   Serial.write(176); // Caractère degré
104   Serial.write('C');
105   Serial.println();
106 }
```

L'extrait de code ci-dessus est disponible en téléchargement sur cette page (https://www.carnetdumaker.net/snippets/96/) (le lien de téléchargement en .zip contient le projet Arduino prêt à l'emploi).

Passons maintenant au second cas d'usage ultra classique : vous avez plusieurs capteurs sur un même bus 1-Wire.

N.B. Dans cette version du code, l'ordre de lecture des capteurs est matériellement fixé par les adresses des divers capteurs. On verra dans le chapitre suivant comment fixer soi-même l'ordre de lecture.

```
1   /** Fonction loop() **/
2   void loop() {
3     float temperature[3];
4
5     /* Lit les températures des trois capteurs */
6     if (getTemperature(&temperature[0], true) != READ_OK) {
7       Serial.println(F("Erreur de lecture du capteur 1"));
8       return;
9     }
10    if (getTemperature(&temperature[1], false) != READ_OK) {
11      Serial.println(F("Erreur de lecture du capteur 2"));
12      return;
13    }
14    if (getTemperature(&temperature[2], false) != READ_OK) {
15      Serial.println(F("Erreur de lecture du capteur 3"));
16      return;
17    }
18
19    /* Affiche les températures */
20    Serial.print(F("Temperatures : "));
21    Serial.print(temperature[0], 2);
22    Serial.write(176); // Caractère degré
23    Serial.print(F("C, "));
24    Serial.print(temperature[1], 2);
25    Serial.write(176); // Caractère degré
26    Serial.print(F("C, "));
27    Serial.print(temperature[2], 2);
28    Serial.write(176); // Caractère degré
29    Serial.println('C');
30  }
```

Pour lire plusieurs capteurs de suite, il suffit de passer `true` en second argument lors du premier appel à `getTemperature()` puis `false` pour les appels suivants. Cela aura pour effet de redémarrer la recherche de capteur lors de la lecture du premier capteur. L'ordre de lecture des capteurs restera ainsi toujours le même.

Le code complet avec commentaires :

```
1   /**
2    * Exemple de code pour lire plusieurs capteurs DS18B20 sur un même bus 1-Wire.
3    */
4
5   /* Dépendance pour le bus 1-Wire */
6   #include <OneWire.h>
7
8
9   /* Broche du bus 1-Wire */
10  const byte BROCHE_ONEWIRE = 7;
11
12  /* Code de retour de la fonction getTemperature() */
13  enum DS18B20_RCODES {
14    READ_OK,
15    NO_SENSOR_FOUND,
16    INVALID_ADDRESS,
17    INVALID_SENSOR
18  };
19
20
21  /* Création de l'objet OneWire pour manipuler le bus 1-Wire */
22  OneWire ds(BROCHE_ONEWIRE);
23
24
25  /**
```

```
 * Fonction de lecture de la température via un capteur DS18B20.
 */
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 bus 1-Wire ci nécessaire (requis pour la lecture du premier capteur) */
  if (reset_search) {
    ds.reset_search();
  }

  /* Recherche le prochain capteur 1-Wire disponible */
  if (!ds.search(addr)) {
    // Pas de capteur
    return NO_SENSOR_FOUND;
  }

  /* Vérifie que l'adresse a été correctement reçue */
  if (OneWire::crc8(addr, 7) != addr[7]) {
    // Adresse invalide
    return INVALID_ADDRESS;
  }

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

  /* Reset le bus 1-Wire et sélectionne le capteur */
  ds.reset();
  ds.select(addr);

  /* Lance une prise de mesure de température et attend la fin de la mesure */
  ds.write(0x44, 1);
  delay(800);

  /* Reset le bus 1-Wire, sélectionne le capteur et envoie une demande de lecture du scratchpad */
  ds.reset();
  ds.select(addr);
  ds.write(0xBE);

 /* Lecture du scratchpad */
  for (byte i = 0; i < 9; i++) {
    data[i] = ds.read();
  }

  /* Calcul de la température en degré Celsius */
  *temperature = ((data[1] << 8) | data[0]) * 0.0625;

  // Pas d'erreur
  return READ_OK;
}


/** Fonction setup() **/
void setup() {

  /* Initialisation du port série */
  Serial.begin(115200);
}


/** Fonction loop() **/
void loop() {
  float temperature[3];

  /* Lit les températures des trois capteurs */
  if (getTemperature(&temperature[0], true) != READ_OK) {
    Serial.println(F("Erreur de lecture du capteur 1"));
    return;
  }
  if (getTemperature(&temperature[1], false) != READ_OK) {
    Serial.println(F("Erreur de lecture du capteur 2"));
    return;
  }
  if (getTemperature(&temperature[2], false) != READ_OK) {
    Serial.println(F("Erreur de lecture du capteur 3"));
    return;
  }

  /* Affiche les températures */
  Serial.print(F("Temperatures : "));
  Serial.print(temperature[0], 2);
  Serial.write(176); // Caractère degré
  Serial.print(F("C, "));
```

```
  Serial.print(temperature[1], 2);
  Serial.write(176); // Caractère degré
  Serial.print(F("C, "));
  Serial.print(temperature[2], 2);
  Serial.write(176); // Caractère degré
  Serial.println('C');
}
```

L'extrait de code ci-dessus est disponible en téléchargement sur cette page (https://www.carnetdumaker.net/snippets/97/) (le lien de téléchargement en .zip contient le projet Arduino prêt à l'emploi).

Lire un ou plusieurs capteurs par adressage direct

Dans le chapitre précédent, on a vu comment lire un ou plusieurs capteurs. Cependant, il reste un souci : l'ordre de lecture des capteurs est lié aux adresses des capteurs, qui sont physiquement gravées dans le capteur lors de la fabrication.

On est ainsi obligé de comprendre l'ordre de lecture des capteurs avant de pouvoir les utiliser correctement. Et en cas d'ajout, remplacement ou suppression de capteur sur le bus, tout doit être revérifié. Pas terrible.

La solution consiste à ne pas chercher les capteurs sur le bus, mais a directement communiqué avec les capteurs dont on connait à l'avance les adresses. Pour connaitre les adresses de vos capteurs, je vous invite à jeter un oeil à mon scanneur de bus 1-Wire (https://www.carnetdumaker.net/articles/faire-un-scanneur-de-bus-1-wire-avec-une-carte-arduino/) 😉

```
/* Dépendance pour le bus 1-Wire */
#include <OneWire.h>


/* Broche du bus 1-Wire */
const byte BROCHE_ONEWIRE = 7;

/* Adresses des capteurs de température */
const byte SENSOR_ADDRESS_1[] = { 0x28, 0x9E, 0x9C, 0x1F, 0x00, 0x00, 0x80, 0x04 };
const byte SENSOR_ADDRESS_2[] = { 0x28, 0x1D, 0x9B, 0x1F, 0x00, 0x00, 0x80, 0xE6 };
const byte SENSOR_ADDRESS_3[] = { 0x28, 0x0F, 0x91, 0x1F, 0x00, 0x00, 0x80, 0x6E };


/* Création de l'objet OneWire pour manipuler le bus 1-Wire */
OneWire ds(BROCHE_ONEWIRE);
```

On reprend tout depuis le début :

- On inclut la bibliothèque OneWire pour pouvoir communiquer avec les capteurs.
- On déclare la broche que l'on utilise pour le bus.
- On déclare toutes les adresses de nos capteurs (ici j'ai inclus les adresses de trois de mes capteurs).

  N.B. Chaque capteur a une adresse unique, inutile donc de copier-coller bêtement mes adresses, elles ne fonctionneront pas avec vos capteurs. Utilisez mon scanneur de bus 1-Wire en lien ci-dessus.
- On déclare l'objet OneWire qui permet de communiquer avec les capteurs.

```
/** Fonction setup() **/
void setup() {

  /* Initialisation du port série */
  Serial.begin(115200);
}
```

La fonction `setup()` est la même que précédemment, pas de changement de ce côté-là du code.

```
/** Fonction loop() **/
void loop() {
  float temperature[3];

  /* Lit les températures des trois capteurs */
  temperature[0] = getTemperature(SENSOR_ADDRESS_1);
  temperature[1] = getTemperature(SENSOR_ADDRESS_2);
  temperature[2] = getTemperature(SENSOR_ADDRESS_3);

  /* Affiche les températures */
  Serial.print(F("Temperatures : "));
  Serial.print(temperature[0], 2);
  Serial.write(176); // Caractère degré
  Serial.print(F("C, "));
  Serial.print(temperature[1], 2);
  Serial.write(176); // Caractère degré
  Serial.print(F("C, "));
  Serial.print(temperature[2], 2);
  Serial.write(176); // Caractère degré
  Serial.println('C');
}
```

Pour la fonction `loop()`, pas de gros changement, j'ai juste simplifier la fonction `getTemperature()` qui retourne désormais directement la température (les codes d'erreurs n'étant plus vraiment utile).

```
/**
 * Fonction de lecture de la température via un capteur DS18B20.
 */
float getTemperature(const byte addr[]) {
  byte data[9];
  // data[] : Données lues depuis le scratchpad
  // addr[] : Adresse du module 1-Wire détecté

  /* Reset le bus 1-Wire et sélectionne le capteur */
  ds.reset();
  ds.select(addr);

  /* Lance une prise de mesure de température et attend la fin de la mesure */
  ds.write(0x44, 1);
  delay(800);

  /* Reset le bus 1-Wire, sélectionne le capteur et envoie une demande de lecture du scratchpad */
  ds.reset();
  ds.select(addr);
  ds.write(0xBE);

 /* Lecture du scratchpad */
  for (byte i = 0; i < 9; i++) {
    data[i] = ds.read();
  }

  /* Calcul de la température en degré Celsius */
  return ((data[1] << 8) | data[0]) * 0.0625;
}
```

L'algorithme de fonctionnement de la nouvelle version de la fonction `getTemperature()` est très identique à l'ancien algorithme, la partie recherche de capteurs en moins.

Le code complet avec commentaires :

```
/**
 * Exemple de code pour lire plusieurs capteurs DS18B20 sur un même bus 1-Wire via leur adresse unique.
 */

/* Dépendance pour le bus 1-Wire */
#include <OneWire.h>


/* Broche du bus 1-Wire */
const byte BROCHE_ONEWIRE = 7;

/* Adresses des capteurs de température */
const byte SENSOR_ADDRESS_1[] = { 0x28, 0x9E, 0x9C, 0x1F, 0x00, 0x00, 0x80, 0x04 };
const byte SENSOR_ADDRESS_2[] = { 0x28, 0x1D, 0x9B, 0x1F, 0x00, 0x00, 0x80, 0xE6 };
const byte SENSOR_ADDRESS_3[] = { 0x28, 0x0F, 0x91, 0x1F, 0x00, 0x00, 0x80, 0x6E };


/* Création de l'objet OneWire pour manipuler le bus 1-Wire */
OneWire ds(BROCHE_ONEWIRE);


/**
 * Fonction de lecture de la température via un capteur DS18B20.
 */
float getTemperature(const byte addr[]) {
  byte data[9];
  // data[] : Données lues depuis le scratchpad
  // addr[] : Adresse du module 1-Wire détecté

  /* Reset le bus 1-Wire et sélectionne le capteur */
  ds.reset();
  ds.select(addr);

  /* Lance une prise de mesure de température et attend la fin de la mesure */
  ds.write(0x44, 1);
  delay(800);

  /* Reset le bus 1-Wire, sélectionne le capteur et envoie une demande de lecture du scratchpad */
  ds.reset();
  ds.select(addr);
  ds.write(0xBE);

 /* Lecture du scratchpad */
  for (byte i = 0; i < 9; i++) {
    data[i] = ds.read();
  }

  /* Calcul de la température en degré Celsius */
  return ((data[1] << 8) | data[0]) * 0.0625;
}


/** Fonction setup() **/
void setup() {

  /* Initialisation du port série */
  Serial.begin(115200);
}


/** Fonction loop() **/
void loop() {
  float temperature[3];

  /* Lit les températures des trois capteurs */
  temperature[0] = getTemperature(SENSOR_ADDRESS_1);
  temperature[1] = getTemperature(SENSOR_ADDRESS_2);
  temperature[2] = getTemperature(SENSOR_ADDRESS_3);

  /* Affiche les températures */
  Serial.print(F("Temperatures : "));
  Serial.print(temperature[0], 2);
  Serial.write(176); // Caractère degré
  Serial.print(F("C, "));
  Serial.print(temperature[1], 2);
  Serial.write(176); // Caractère degré
  Serial.print(F("C, "));
  Serial.print(temperature[2], 2);
  Serial.write(176); // Caractère degré
  Serial.println('C');
}
```

L'extrait de code ci-dessus est disponible en téléchargement sur cette page (https://www.carnetdumaker.net/snippets/98/) (le lien de téléchargement en .zip contient le projet Arduino prêt à l'emploi).

Bonus : Lire simultanément plusieurs capteurs

Les plus attentifs auront remarqué un défaut dans les codes ci-dessus : une mesure prend 800 millisecondes, deux mesures prennent 1.6 seconde, trois mesures 2.4 secondes, etc. Mesurer dix températures prendrait 8 secondes !

Comme avec beaucoup d'algorithmes bloquants, il est possible de découper l'algorithme en plusieurs algorithmes plus courts non bloquants.

```
/**
 * Fonction de démarrage de la prise de mesure de la température via un capteur DS18B20.
 */
void startTemperatureMeasure(const byte addr[]) {
  // addr[] : Adresse du module 1-Wire détecté

  /* Reset le bus 1-Wire et sélectionne le capteur */
  ds.reset();
  ds.select(addr);

  /* Lance une prise de mesure de température et attend la fin de la mesure */
  ds.write(0x44, 1);
}

/**
 * Fonction de récupération de la prise de mesure de la température via un capteur DS18B20.
 */
float readTemperatureMeasure(const byte addr[]) {
  byte data[9];
  // data[] : Données lues depuis le scratchpad
  // addr[] : Adresse du module 1-Wire détecté

  /* Reset le bus 1-Wire, sélectionne le capteur et envoie une demande de lecture du scratchpad */
  ds.reset();
  ds.select(addr);
  ds.write(0xBE);

 /* Lecture du scratchpad */
  for (byte i = 0; i < 9; i++) {
    data[i] = ds.read();
  }

  /* Calcul de la température en degré Celsius */
  return ((data[1] << 8) | data[0]) * 0.0625;
}
```

Il est donc ici possible de découper la fonction getTemperature en deux fonctions : une pour déclencher la mesure de température et une pour lire le résultat de la mesure.

```
/** Fonction loop() **/
void loop() {
  float temperature[3];

  /* Lit les températures des trois capteurs */
  startTemperatureMeasure(SENSOR_ADDRESS_1);
  startTemperatureMeasure(SENSOR_ADDRESS_2);
  startTemperatureMeasure(SENSOR_ADDRESS_3);
  delay(800);
  temperature[0] = readTemperatureMeasure(SENSOR_ADDRESS_1);
  temperature[1] = readTemperatureMeasure(SENSOR_ADDRESS_2);
  temperature[2] = readTemperatureMeasure(SENSOR_ADDRESS_3);

  /* Affiche les températures */
  Serial.print(F("Temperatures : "));
  Serial.print(temperature[0], 2);
  Serial.write(176); // Caractère degré
  Serial.print(F("C, "));
  Serial.print(temperature[1], 2);
  Serial.write(176); // Caractère degré
  Serial.print(F("C, "));
  Serial.print(temperature[2], 2);
  Serial.write(176); // Caractère degré
  Serial.println('C');
}
```

Il devient alors possible de déclencher la mesure de multiples capteurs, puis d'attendre une seule fois et ensuite de lire toutes les mesures en une fois.

Le code complet avec commentaires :

```
/**
 * Exemple de code pour lire plusieurs capteurs DS18B20 sur un même bus 1-Wire via leur adresse unique sans délai.
 */

/* Dépendance pour le bus 1-Wire */
#include <OneWire.h>

```

```
 8
 9  /* Broche du bus 1-Wire */
10  const byte BROCHE_ONEWIRE = 7;
11
12  /* Adresses des capteurs de température */
13  const byte SENSOR_ADDRESS_1[] = { 0x28, 0x9E, 0x9C, 0x1F, 0x00, 0x00, 0x80, 0x04 };
14  const byte SENSOR_ADDRESS_2[] = { 0x28, 0x1D, 0x9B, 0x1F, 0x00, 0x00, 0x80, 0xE6 };
15  const byte SENSOR_ADDRESS_3[] = { 0x28, 0x0F, 0x91, 0x1F, 0x00, 0x00, 0x80, 0x6E };
16
17
18  /* Création de l'objet OneWire pour manipuler le bus 1-Wire */
19  OneWire ds(BROCHE_ONEWIRE);
20
21
22  /**
23   * Fonction de démarrage de la prise de mesure de la température via un capteur DS18B20.
24   */
25  void startTemperatureMeasure(const byte addr[]) {
26    // addr[] : Adresse du module 1-Wire détecté
27
28    /* Reset le bus 1-Wire et sélectionne le capteur */
29    ds.reset();
30    ds.select(addr);
31
32    /* Lance une prise de mesure de température et attend la fin de la mesure */
33    ds.write(0x44, 1);
34  }
35
36  /**
37   * Fonction de récupération de la prise de mesure de la température via un capteur DS18B20.
38   */
39  float readTemperatureMeasure(const byte addr[]) {
40    byte data[9];
41    // data[] : Données lues depuis le scratchpad
42    // addr[] : Adresse du module 1-Wire détecté
43
44    /* Reset le bus 1-Wire, sélectionne le capteur et envoie une demande de lecture du scratchpad */
45    ds.reset();
46    ds.select(addr);
47    ds.write(0xBE);
48
49   /* Lecture du scratchpad */
50    for (byte i = 0; i < 9; i++) {
51      data[i] = ds.read();
52    }
53
54    /* Calcul de la température en degré Celsius */
55    return ((data[1] << 8) | data[0]) * 0.0625;
56  }
57
58
59  /** Fonction setup() **/
60  void setup() {
61
62    /* Initialisation du port série */
63    Serial.begin(115200);
64  }
65
66
67  /** Fonction loop() **/
68  void loop() {
69    float temperature[3];
70
71    /* Lit les températures des trois capteurs */
72    startTemperatureMeasure(SENSOR_ADDRESS_1);
73    startTemperatureMeasure(SENSOR_ADDRESS_2);
74    startTemperatureMeasure(SENSOR_ADDRESS_3);
75    delay(800);
76    temperature[0] = readTemperatureMeasure(SENSOR_ADDRESS_1);
77    temperature[1] = readTemperatureMeasure(SENSOR_ADDRESS_2);
78    temperature[2] = readTemperatureMeasure(SENSOR_ADDRESS_3);
79
80    /* Affiche les températures */
81    Serial.print(F("Temperatures : "));
82    Serial.print(temperature[0], 2);
83    Serial.write(176); // Caractère degré
84    Serial.print(F("C, "));
85    Serial.print(temperature[1], 2);
86    Serial.write(176); // Caractère degré
87    Serial.print(F("C, "));
88    Serial.print(temperature[2], 2);
89    Serial.write(176); // Caractère degré
90    Serial.println('C');
91  }
```

L'extrait de code ci-dessus est disponible en téléchargement sur cette page (https://www.carnetdumaker.net/snippets/99/) (le lien de téléchargement en .zip contient le projet Arduino prêt à l'emploi).

Bonus : Réduire la résolution pour diminuer le temps de mesure

Table 2. Thermometer Resolution Configuration

R1	R0	RESOLUTION (BITS)	MAX CONVERSION TIME	
0	0	9	93.75ms	(t_{CONV}/8)
0	1	10	187.5ms	(t_{CONV}/4)
1	0	11	375ms	(t_{CONV}/2)
1	1	12	750ms	(t_{CONV})

(https://www.carnetdumaker.net/images/illustration-du-registre-de-configuration-du-capteur-ds18b20/)

Illustration du registre de configuration

Il y a une relation entre la résolution de la mesure et la durée de la mesure. Plus la résolution est grande, plus la mesure prend de temps.

En diminuant la résolution de la mesure, il est possible de faire plus de mesures dans une même période de temps. Par exemple, en passant d'une résolution de 12 bits à seulement 9 bits, il est possible de faire huit mesures au lieu d'une seule.

Pour modifier la résolution d'un capteur dont l'adresse est connue, vous pouvez utiliser la fonction ci-dessous :

```
/* Résolution disponibles */
const byte RESOLUTION_12_BITS = 0b01111111;
const byte RESOLUTION_11_BITS = 0b01011111;
const byte RESOLUTION_10_BITS = 0b00111111;
const byte RESOLUTION_9_BITS = 0b00011111;

/** Change la résolution du capteur spécifié */
void changeResolution(const byte addr[], byte resolution) {

  /* Reset le bus 1-Wire, sélectionne le capteur et envoie une demande d'écriture du scratchpad */
  ds.reset();
  ds.select(addr);
  ds.write(0x4E);

  /* Ecrit dans le scratchpad */
  ds.write(0x00);
  ds.write(0x00);
  ds.write(resolution);

  /* Fin d'écriture */
  ds.reset();
}
```

Mettre à jour le reste du code

Le changement résolution du capteur entraîne non seulement une modification du temps de mesure, mais aussi de la précision finale de la mesure (en même temps, c'est le but). Il est donc nécessaire de modifier la constante utilisée pour calculer la température finale.

Résolution	Constante de calcul	Temps de mesure
9 bits	0.5 (°C)	93.75ms
10 bits	0.25 (°C)	187.5ms
11 bits	0.125 (°C)	375ms
12 bits	0.0625 (°C)	750ms

Si vous ne modifiez pas la constante de calcul, les mesures seront complètement fausses.

Comme vous êtes des lecteurs fort sympathiques, voici une petite modification de code qui fait tout le boulot pour vous :

```
/* A mettre en début de programme avec les autres constantes */
const float RESOLUTION_CONSTANTS[4] = {
  0.5, 0.25, 0.125, 0.0625
};

// Remplacer "0.0625" par "RESOLUTION_CONSTANTS[data[4] >> 5]"
// Exemple : ((data[1] << 8) | data[0]) * RESOLUTION_CONSTANTS[data[4] >> 5];
```

Cette petite modification utilise le fait que le scratchpad contient la valeur courante du registre de configuration. Il est donc possible d'appliquer le bon coefficient de calcul en fonction de la résolution automatiquement.

PS Je n'ai pas inclus cette modification dans les codes ci-dessus, car celle-ci n'est utile que ci vous modifiez la résolution du capteur, ce qui est un cas d'usage assez peu courant.

Conclusion

Ce tutoriel est désormais terminé.

Si ce tutoriel vous a plu, n'hésitez pas à le commenter sur le forum, à le partager sur les réseaux sociaux et à soutenir le site si cela vous fait plaisir.

Articles précédents

- Faire un scanneur de bus 1-Wire avec une carte Arduino (/articles/faire-un-scanneur-de-bus-1-wire-avec-une-carte-arduino/)

Articles en relation avec celui-ci

- Mesurer une température avec un capteur LM35 et une carte Arduino / Genuino (/articles/mesurer-une-temperature-avec-un-capteur-lm35-et-une-carte-arduino-genuino/)

Cliquez ici pour accéder aux commentaires de l'article. (/forum/topics/90-mesurer-une-temperature-avec-un-capteur-1-wire-ds18b20-et-une-carte-arduino-genuino/)

Qui sommes-nous ? (/pages/qui-sommes-nous/)
Pourquoi ce site ? (/pages/pourquoi-ce-site/)
Nos engagements (/pages/nos-engagements/)
Foire aux questions (/pages/faq/)
Conditions générales d'utilisation (/pages/conditions-generales-d-utilisation/)
Plan du site (/pages/plan-du-site/)

Nous contacter (/pages/nous-contacter/)
Mentions légales (/pages/mentions-legales/)
Utilisation des cookies (/pages/cookies/)

@CarnetDuMaker (https://twitter.com/carnetdumaker)
+CarnetDuMaker (https://plus.google.com/102700422941341090773/about)
Page CarnetDuMaker (https://www.facebook.com/CarnetDuMaker/)
Chaine CarnetDuMaker (https://www.youtube.com/channel/UCAafmzWNcunTWO5d5cli2yw)
Github TamiaLab (https://github.com/TamiaLab)
The cake is a lie