B.a.ba capteur : le capteur de température DS18B20

connexion au bus 1-Wire

Mathias Claussen (Allemagne)

Inclus dans de nombreux kits pour débutants, le capteur DS18B20 de Dallas Semiconductor facilite la découverte de la mesure de température. Avant de se lancer, il vaut mieux connaître un peu le bus 1-Wire et la façon d'y connecter le capteur. Nous allons rapidement explorer ici les bases de l'utilisation du DS18B20, pour vous permettre de débuter sereinement dans la mesure de température !

Si vous souhaitez enregistrer des températures à l'aide d'un microcontrôleur, vous avez le choix entre plusieurs capteurs et systèmes de bus. L'un des plus répandu est le DS18B20 de Dallas Semiconductor. Sa plage de mesure s'étend de -5 °C à 125 °C (-67° à +275 °F) avec une précision impressionnante de ±0,5 °C. Sa polyvalence lui permet de mesurer non seulement les températures ambiantes, mais aussi de surveiller les congélateurs et les chambres froides. Dans cet article, nous examinons de plus près son fonctionnement avec un microcontrôleur, avec des exemples comprenant le code source et les schémas de câblage pour montrer comment l'intégrer dans vos propres projets.

Le DS18B20

Le capteur DS18B20 utilise le système de bus 1-Wire, breveté par Dallas Semiconductor en 1989 et qui permet la connexion de plusieurs capteurs. Avec sa plage d'alimentation de 3,0 V à 5,5 V, on peut connecter directement le DS18B20 aux GPIO de nombreux appareils, de l'Arduino UNO au Raspberry Pi Pico. Le bus 1-Wire offre également un mode d'alimentation parasite, où l'énergie est prélevée sur la ligne de données qui alimente le capteur. Cela signifie que le bus 1-Wire ne nécessite qu'une seule broche sur un microcontrôleur (MCU) pour connecter plusieurs capteurs, ce qui justifie son choix, en particulier sur les cartes MCU les plus petites avec peu de GPIO. Bien qu'il existe d'autres capteurs 1-Wire, nous nous concentrerons ici sur le dispositif Dallas.

Figure 1. Le DS18B20 dans un boîtier TO-92.

Figure 2. Version étanche du DS18B20 avec câble.

Figure 3. Brochage du DS18B20.

Le capteur DS18B20 est disponible sous différentes formes, comme le montrent les **figures 1** et **2**. Grâce au bus 1-Wire, il est désormais très facile de connecter un plus grand nombre de capteurs. Bien que le protocole 1-Wire lui-même n'impose pas de limite au nombre de capteurs pouvant être connectés au bus, les limites sont déterminées par les propriétés électriques du bus.

Le bus 1-Wire

Pour connecter des capteurs au bus 1-Wire, trois fils sont nécessaires : la masse (GND), les données (DQ) et la tension d'alimentation (VCC). Le bus 1-Wire est bidirectionnel et fonctionne selon le concept contrôleur/cible (maître et esclave), où le contrôleur et la cible échangent des données via la ligne de données. Le brochage du capteur DS18B20 est présenté à la **figure 3**.

Tous les nœuds sur l'unique ligne de données utilisent un pilote à collecteur ouvert ; en cas de collision de données, lorsque deux nœuds essaient par erreur de parler en même temps, il en résulte simplement des données corrompues. Il ne peut jamais y avoir de conflit matériel lorsqu'un nœud essaie de forcer la ligne de données vers le haut alors qu'un autre essaie de la forcer vers le bas. Une résistance de rappel est nécessaire parce qu'aucun des nœuds ne peut forcer activement le bus vers VCC. La **figure 4** montre la connexion d'un capteur 1-Wire à un Arduino UNO.

Outre la connexion électrique, il faut un protocole pour le fonctionnement des appareils 1-Wire. Heureusement, la plupart des microcontrôleurs intègrent un UART qui peut être utilisé pour cela, sans nécessiter de matériel particulier. La **figure 5** présente un circuit approprié.

Pour la commande via UART, Analog Devices propose un article utile [1]. Alors que l'UART peut aider à réduire la charge du CPU, il n'est pas essentiel avec les MCU d'aujourd'hui. Même un petit ATtiny est assez puissant pour adresser les capteurs connectés via le bus 1-Wire grâce à une solution purement logicielle, ne nécessitant qu'une broche d'E/S. La plupart des bibliothèques qui fonctionnent avec le DS18B20 n'utilisent qu'un seul GPIO, ce qui rend l'utilisation d'un UART plutôt rare.

Identification du capteur

Lorsque plusieurs capteurs sont connectés à un fil de bus commun, chacun d'eux doit pouvoir être adressé individuellement afin qu'un seul capteur à la fois soit autorisé à écrire sur le bus et éviter que ses données ne soient brouillées ou corrompues par un autre capteur.

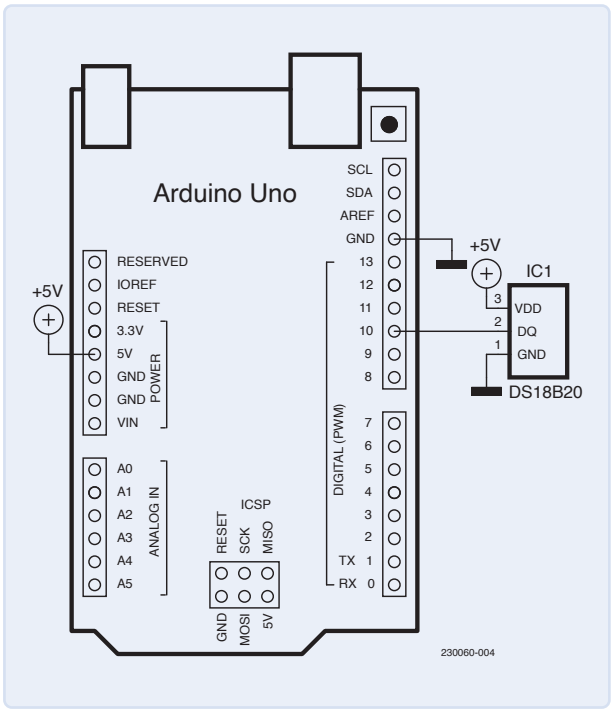

Figure 4. Schéma du circuit Arduino UNO avec un DS18B20.

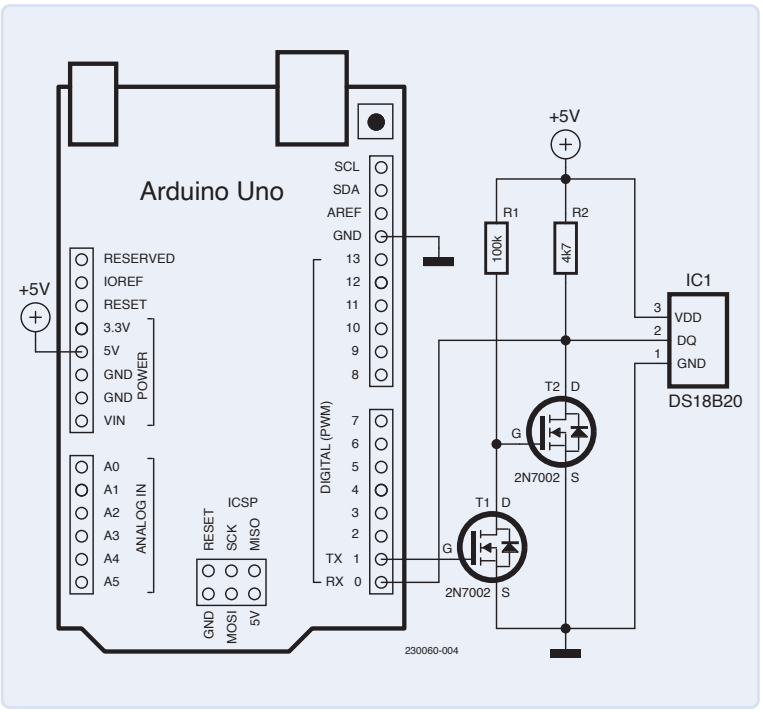

Figure 5. Schéma du bus 1-Wire du DS18B20 utilisant l'UART.

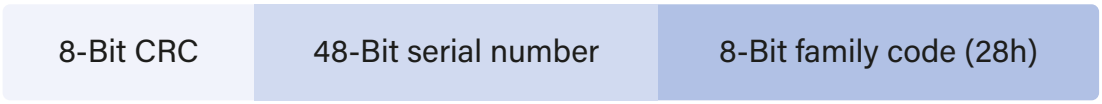

Figure 6. Format du message UUID du DS18B20.

Pour cela, chaque participant au bus 1-Wire dispose d'un code d'identification unique, d'une largeur de 64 bits (UUID). Il se compose d'un code sur 8 bits indiquant la famille du capteur, d'un numéro de série sur 48 bits et d'une somme de contrôle sur 8 bits (CRC) (**figure 6**). Le numéro de série ne figure toutefois nulle part sur le capteur, mais réside en interne, stocké sur 8 octets en ROM. Il peut être récupéré par logiciel à l'aide d'une fonction de recherche sur le bus 1-Wire. On peut trouver l'algorithme de recherche sur la page d'Analog Devices [2].

Exemple de code pour Arduino

À partir du schéma de la **figure 4**, la configuration reste très simple. Le capteur utilise une alimentation de 5 V comme l'Arduino UNO. Le GPIO de l'Arduino UNO, connecté à la ligne de données ou DQ, nécessite une résistance de rappel de 4,7 kΩ. La bibliothèque pour bus 1-Wire que nous utilisons ici est la bibliothèque *OneWire* de Paul Stoffregen [3]. Le code d'exemple de la bibliothèque montre aussi comment lire les données des capteurs DS18B20 (**listage 1**), ce que nous pouvons maintenant examiner plus en détail.

Pour utiliser la bibliothèque, il faut d'abord l'inclure avec `#include <OneWire.h>` au début du croquis. Ensuite, avec `OneWire ds(10)`, on affecte la broche 10 au fonctionnement de la ligne de données du bus 1-Wire. Avec `ds.search(addr)`, on recherche le capteur suivant sur le bus. La recherche est terminée lorsque tous les capteurs ont été identifiés. Les identifications de tous les capteurs trouvés sont stockées dans `addr`.

Maintenant que le capteur a été identifié, on peut interagir avec lui. Le premier octet, `addr[0]`, contient le code indiquant la famille. Il peut servir à déterminer si un capteur de température DS18B20, DS18S20 ou DS1822 a été trouvé, ou s'il s'agit d'un autre type de dispositif de bus 1-Wire.

La séquence suivante apparaît à la ligne 69 :

```
ds.reset();
ds.select(addr);
ds.write(0x44, 1);
```

Une réinitialisation de tous les participants au bus est effectuée une fois à l'aide de `ds.reset()`. Ensuite, en utilisant `ds.select(addr)`, le capteur découvert est adressé. Pour demander une lecture de température au capteur, il faut d'abord lancer une mesure. La commande correspondante est `ds.write(0x44, 1)`, où 0x44 est la commande reconnue par le capteur. Avec le deuxième paramètre (ici, `1`), la broche d'E/S est forcée au niveau haut pour alimenter les capteurs tels que le DS18S20 en mode parasite. Le résultat de la mesure de température est disponible après 750 à 1000 ms. Dans cet exemple, on utilise un délai pour attendre la fin du traitement. Lorsque la mesure de la température est terminée et que le résultat est prêt, on peut le récupérer. Le capteur est à nouveau adressé à l'aide de `ds.select(addr)` et on lui demande de lire la valeur stockée dans sa mémoire de travail (`ds.write(0xBE)`). Les neuf octets stockés ici, comprenant la valeur de la température, vont maintenant être envoyés.

Avec le DS18B20, la température est disponible après lecture sous forme d'une valeur sur 16 bits, composée de deux octets de registre. Si on utilise un capteur de type DS18S20 au lieu d'un DS18B20, les valeurs de registre ont un poids différent, si bien que leur position dans le registre doit être modifiée. L'exemple de code en tient compte et effectue les 3 décalages vers la gauche nécessaires sur la valeur brute.

Le DS18B20 fournit la température avec une résolution de 12 bits et prend 750 ms pour la conversion. Il peut, au choix, fournir des valeurs sur 11, 10 ou 9 bits, ce qui permet d'obtenir un temps de conversion plus rapide. Avec une résolution de 9 bits, la valeur des bits de poids faible n'est pas définie, ils peuvent donc être mis à zéro. La température relevée par le capteur est toujours exprimée en degrés Celsius ; votre logiciel devra effectuer la conversion en degrés Fahrenheit si nécessaire.

DS18B20 - La guerre des clones

Le DS18B20 est un capteur répandu, souvent inclus dans divers kits de capteurs et kits de début pour microcontrôleurs. Ces capteurs sont aussi très bon marché ; vous pouvez en trouver un pour seulement 30 centimes sur certains sites de vente en ligne. Les distributeurs tels que Mouser ou Farnell vous feront cependant payer plus de 4 euros pour un seul DS18B20.

La popularité de ce capteur a incité certaines usines malhonnêtes de fabrication de puces à produire leurs propres versions, qui ressemblent au capteur original et semblent se comporter comme lui, mais qui peuvent s'écarter considérablement des caractéristiques de l'original après un examen plus approfondi. Les distributeurs reconnus s'approvisionnent en composants auprès de fournisseurs certifiés.

Pour vous assurer que vous disposez d'un capteur authentique, la page GitHub de Chris Petrich [4] fournit des croquis Arduino pour tester votre capteur, ainsi que de plus amples informations sur chaque clone particulier et ses déviations par rapport à la conception originale.

Résumé

Il est facile de connecter un capteur de température à un microcontrôleur avec le DS18B20 et 1-Wire. Comme nous l'avons vu, en utilisant la plateforme Arduino, il y a tout un tas de bibliothèques

et d'exemples de code source pour alléger le processus. Avec cette configuration, une seule broche du MCU est nécessaire pour communiquer avec le capteur, et plusieurs capteurs peuvent être connectés, ce qui permet à un petit MCU avec un GPIO libre de prendre en charge plusieurs capteurs simultanément.

VF : Denis Lafourcade — 230060-04

Des questions, des commentaires ?

Contactez Elektor (redaction@elektor.fr).

Produits

- **Kit capteur Elektor 37-en-1 (SKU 16843)**
  https://elektor.fr/16843
- **Cytron Maker UNO (SKU 18634)**
  https://elektor.fr/18634
- **Kit de développement MakePython ESP32 (SKU 20137)**
  https://elektor.fr/20137

LIENS

[1] Using a UART to implement a 1-wire-bus-master, Analog Devices : https://analog.com/en/technical-articles/using-a-uart-to-implement-a-1wire-bus-master.html
[2] 1-Wire search algorithm, Analog Devices : https://analog.com/en/app-notes/1wire-search-algorithm.html
[3] OneWireLibrary par Paul Stoffregen sur GitHub : https://github.com/PaulStoffregen/OneWire
[4] Chris Petrich, "Your DS18B20 temperature sensor is likely a fake, counterfeit, clone...," GitHub : https://github.com/cpetrich/counterfeit_DS18B20

Listage 1. Exemple DS18B20 avec Arduino.

```
001        #include <OneWire.h>
002
003        // Exemple de température avec DS18S20, DS18B20, DS1822 OneWire
004        //
005        // http://www.pjrc.com/teensy/td_libs_OneWire.html
006        //
007        // La bibliothèque DallasTemperature peut faire tout ce travail pour vous !
008        // https://github.com/milesburton/Arduino-Temperature-Control-Library
009
010        OneWire ds(10);  sur la broche 10 (une résistance de 4k7 est nécessaire)
011
012        void setup(void) {
013          Serial.begin(9600);
014        }
015
016        void loop(void) {
017
018          byte i;
019          byte present = 0;
020          byte type_s;
021          byte data[9];
022          byte addr[8];
023          float celsius, fahrenheit;
024
025          if (!ds.search(addr)) {
026            Serial.println("No more addresses.");
027            Serial.println();
```

suite à la page suivante

```
028           ds.reset_search();
029           delay(250);
030           return;
031         }
032
033
034         Serial.print("ROM =");
035         for(i = 0; i < 8; i++) {
036           Serial.write(' ');
037           Serial.print(addr[i], HEX);
038         }
039
040         if (OneWire::crc8(addr, 7) != addr[7]) {
041             Serial.println("CRC is not valid!");
042             return;
043         }
044
045         Serial.println();
046         // le premier octet en ROM identifie le type
047         switch (addr[0]) {
048
049           case 0x10:
050             Serial.println("  Chip = DS18S20");  // ou l'ancienne DS1820
051             type_s = 1;
052             break;
053
054           case 0x28:
055             Serial.println("  Chip = DS18B20");
056             type_s = 0;
057             break;
058
059           case 0x22:
060             Serial.println("  Chip = DS1822");
061             type_s = 0;
062             break;
063
064           default:
065             Serial.println("Device is not a DS18x20 family device.");
066             return;
067         }
068
069         ds.reset();
070         ds.select(addr);
071        ds.write(0x44, 1); // début de conversion, avec alimentation en mode parasite à la fin
072         delay(1000);       // peut-être que 750ms suffisent, peut-être pas
073         // nous pourrions faire un ds.depower() ici, mais le reset s'en chargera.
074
075         present = ds.reset();
076         ds.select(addr);
077         ds.write(0xBE);         // Lecture de la mémoire de travail
078
079         Serial.print("  Data = ");
080         Serial.print(present, HEX);
081         Serial.print(" ");
082         for (i = 0; i < 9; i++) {           // nous avons besoin de 9 octets
083           data[i] = ds.read();
084           Serial.print(data[i], HEX);
085           Serial.print(" ");
086         }
087         Serial.print(" CRC=");
088         Serial.print(OneWire::crc8(data, 8), HEX);
```

```
089         Serial.println();
090         // Conversion des données en température réelle
091         // comme le résultat est un entier signé sur 16 bits, il doit
092         // être stocké dans un type "int16_t", qui est toujours sur 16 bits,
093         // même lorsqu'il est compilé sur un processeur 32 bits.
094
095         int16_t raw = (data[1] << 8) | data[0];
096         if (type_s) {
097           raw = raw << 3; // résolution 9 bits par défaut
098           if (data[7] == 0x10) {
099             // le résidu donne une résolution complète de 12 bits
100             raw = (raw & 0xFFF0) + 12 - data[6];
101           }
102         } else {
103           byte cfg = (data[4] & 0x60);
104          // à une résolution plus faible, les bits de poids faible sont indéfinis, alors mettons-les à zéro
105           if (cfg == 0x00) raw = raw & ~7;  // 9 bit resolution, 93.75 ms
106           else if (cfg == 0x20) raw = raw & ~3; // 10 bit res, 187.5 ms
107           else if (cfg == 0x40) raw = raw & ~1; // 11 bit res, 375 ms
108           // la résolution par défaut est 12 bits, le temps de conversion est de 750 ms
109         }
110
111         celsius = (float)raw / 16.0;
112         fahrenheit = celsius * 1.8 + 32.0;
113         Serial.print("  Temperature = ");
114         Serial.print(celsius);
115         Serial.print(" Celsius, ");
116         Serial.print(fahrenheit);
117         Serial.println(" Fahrenheit");
118       }
```