

Utiliser la communication série 1-Wire avec Arduino : l'exemple du capteur de température DS18B20.



Ateliers Arduino

par X. HINAULT

www.mon-club-elec.fr



Tous droits réservés – 2013.

Ce document légèrement payant est soumis au droit d'auteur et est réservé à l'usage personnel.

Afin d'encourager la production de supports didactiques de qualité, ce document est légèrement payant.

La licence d'utilisation est attribuée pour un usage personnel uniquement, dans le cercle familial. Mise en ligne et diffusion non autorisées.

Si vous n'êtes pas le détenteur de la licence attribuée pour l'usage de ce document, soyez sympa, merci d'acheter votre exemplaire personnel ici : <https://monclubelec.dpdcart.com/>

Pour tout problème lié à l'utilisation de ce document, veuillez envoyer une copie ici : support@mon-club-elec.fr

Pour obtenir tout autres types de licence d'utilisation (enseignement, commercial, etc...), veuillez contacter l'auteur ici : support@mon-club-elec.fr

Vous avez constaté une erreur ? une coquille ? N'hésitez pas à nous le signaler à cette adresse : support@mon-club-elec.fr

Truc d'utilisation : visualiser ce document en mode diaporama dans le visionneur PDF. Navigation avec les flèches HAUT / BAS ou la souris.

En mode fenêtre, activer le panneau latéral vous facilitera la navigation dans le document. Bonne lecture !

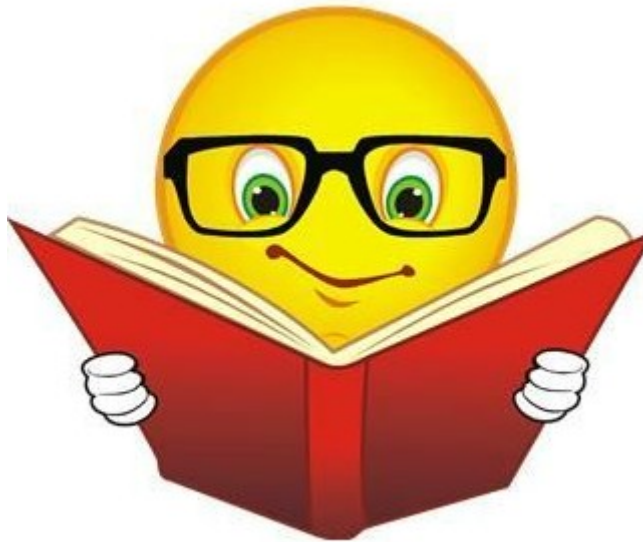
Lancer également le logiciel Arduino et connecter votre carte Arduino afin de pouvoir tester au fur et à mesure les codes d'exemples !

1. Intro

L'objectif ici est :

- de découvrir et d'apprendre à utiliser la communication série 1-wire
- de découvrir la librairie OneWire
- à titre d'exemple, vous allez apprendre comment interfacer votre carte Arduino avec un capteur de température 1-wire très pratique, le DS18B20.

... afin d'être en mesure d'utiliser le protocole de communication 1-wire avec Arduino.



Prêt ? C'est parti !

Pratique :

Les codes de cet atelier sont disponibles ici :

<https://github.com/sensor56/df1fb6962fa152a2cf65683facb427dd>

2. Matériel nécessaire pour les ateliers Arduino

Pour cet atelier, vous aurez besoin de tout ou partie des éléments suivants pour pouvoir réaliser les exemples proposés :

De l'espace de développement Arduino

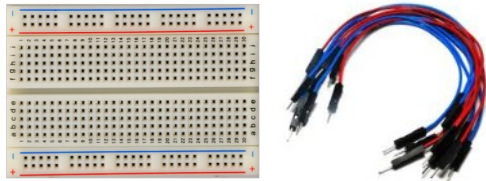


L'espace de développement Arduino associe :

- un ordinateur sous Windows, Mac Os X ou Gnu/Linux (Ubuntu)
- avec le logiciel Arduino installé (voir : <http://www.arduino.cc/>)
- un câble USB
- une carte Arduino UNO ou équivalente.

disponible chez : <http://shop.snootlab.com/> ou <http://www.gotronic.fr/>

Du nécessaire pour réaliser des montages sans soudure

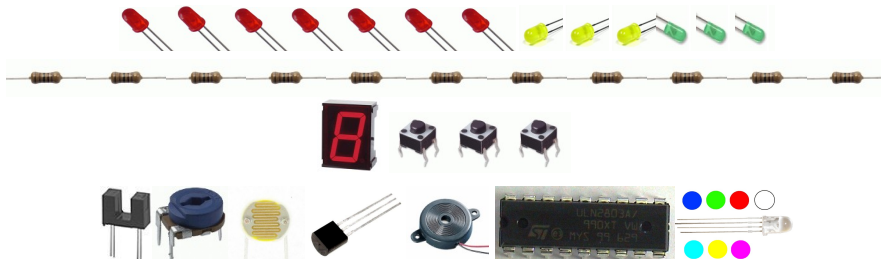


Pour réaliser des montages sans soudure, vous aurez besoin :

- d'une plaque d'essai ou breadboard moyenne (450 points)
- de quelques câbles souples (ou jumpers) mâle/mâle

disponible chez : <http://www.gotronic.fr/>

De quelques composants de base



Pour vous simplifier la vie, nous avons négocié ce kit pour vous !

Vous pouvez commander ce kit complet directement en 1 clic chez notre partenaire

<http://www.gotronic.fr/> avec le code express **701710**

Pour plus de détails, voir : http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ATELIERS

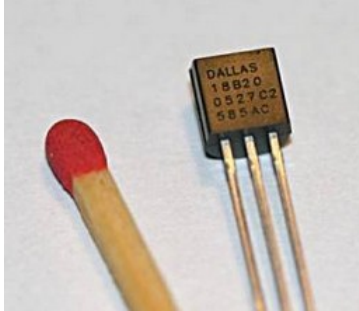
Pour les ateliers Arduino niveau débutant, vous devrez idéalement disposer des composants suivants :

- des LEDs 5mm Rouges(x20), Vertes (x5) et 3 Jaunes (x5)
- digit à cathode commune rouge 13mm (x1)
- Résistances (1/4w - 5%) de 270 Ohms (x20), 4,7K Ohms (x1), 1K Ohms (x1)
- mini bouton-poussoir (x3)
- Opto-fourche (x 1)
- Résistance variable linéaire 10K (x 1)
- Photo-résistance 7mm (x 1)
- Capteur de température LM35DZ (-55/+150°C - 10mV/°C) (x 1)
- Capsule son piézoélectrique (x 1)
- ULN 2803A (CI amplificateur 8 voies, 500mA/ voie) (x 1)
- LED 5mm multicolore RVB cathode commune (x 1)

3. **Matériel spécifique nécessaire pour cet atelier**

Pour cet atelier vous aurez besoin également :

D'un ou plusieurs capteurs de température DS18B20 à communication série 1-Wire



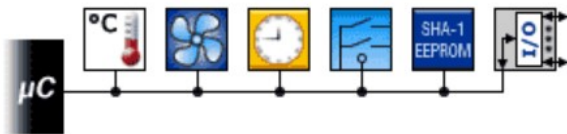
Il s'agit d'un capteur de température :

- peu cher (2 Euros)
- qui réalise en interne la conversion analogique numérique
- peut être connecté simplement sur **2 fils** (technique dite 1-wire)
- qui réalise une mesure de température sur la plage -55°C à $+125^{\circ}\text{C}$ **sans composant supplémentaire** (polyvalent donc...)
- qui renvoie le résultat de la mesure sous forme de données série sur 12 bits (précision de 0.0625°C)
- **plusieurs capteurs peuvent être utilisés simultanément sur les 2 mêmes fils** grâce à un système d'adressage de chaque capteur qui possède une adresse unique (64 bits ROM soit 8 octets).
- Facile à utiliser avec Arduino grâce à une librairie dédiée fournie

4. Technique : la communication 1-Wire : principe général

Intro

- La communication série 1-Wire (One-Wire) est une technologie développée par le fabricant de circuit intégré Dallas et qui permet de **communiquer de façon bi-directionnelle sur seulement 1 fil de communication** (auquel s'ajoute un fil de masse seulement = 0V) entre un microprocesseur et un ou plusieurs capteurs présents sur le fil de communication.
- Peu de composants utilisent ce protocole plutôt spécialisé, mais ce protocole est particulièrement intéressant car il est utilisé notamment par des capteurs de température qui **pourront ainsi être utilisés en grand nombre très facilement avec Arduino sur une seule broche**.



Les éléments clés de la communication 1-Wire

- Cette technique est assez géniale : **les données série vont circuler de façon bi-directionnelle sur un seul fil !**

En pratique, seul un fil de masse (=0V) sera utilisé en plus du fil de communication soit **2 fils en tout**. Ceci est rendu possible grâce à un condensateur interne dans les dispositifs 1-wire qui se charge via le fil de communication permettant d'éviter l'utilisation d'un fil de +.

A titre de comparaison, I2C utilise 2 fils de communication et 2 fils d'alimentation par dispositif soit 4 fils en tout.

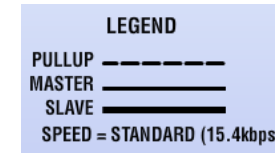
- Un « maître », le microprocesseur va communiquer avec les « esclaves », les capteurs connectés au fil de communication, appelé aussi « bus » 1-wire qui renverront les données voulues.
- Comme pour I2C, **chaque dispositif « esclave » aura une adresse individuelle** (= 1 numéro) permettant de l'identifier et qui est gravée une fois pour toute dans le composant.

Retenez l'essentiel :

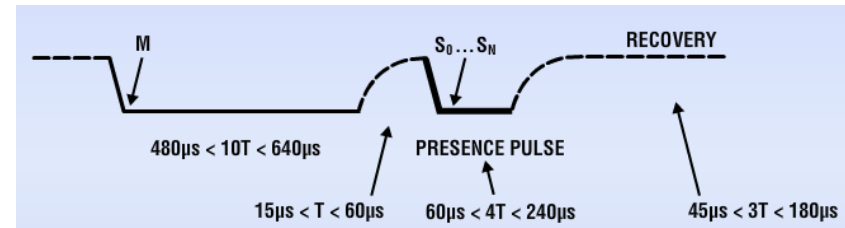
2 éléments qui discutent entre eux : le « maître » (microcontrôleur) et l'esclave (le dispositif 1-Wire)
1 seul fil de communication pour les 2 sens
pas de fil d' horloge !

Le détail des signaux de la communication 1-Wire

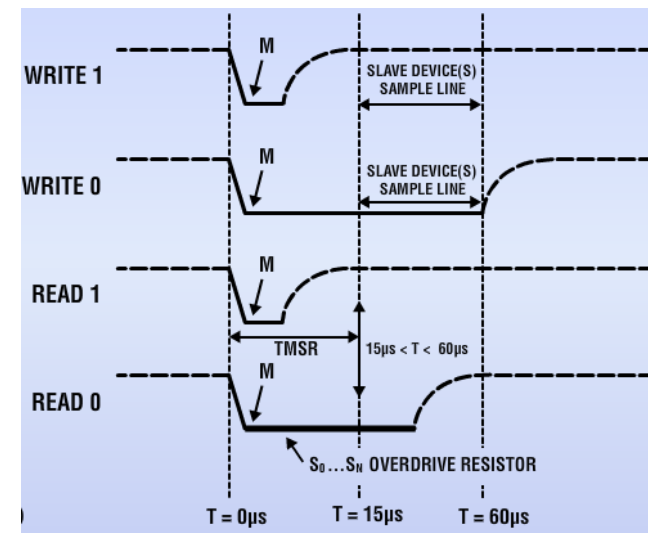
- Sur les schémas suivants, l'aspect du trait se modifie selon que le niveau sur le bus 1-Wire est appliqué par le maître ou l'esclave :



- Tout commence par un signal d'initialisation :



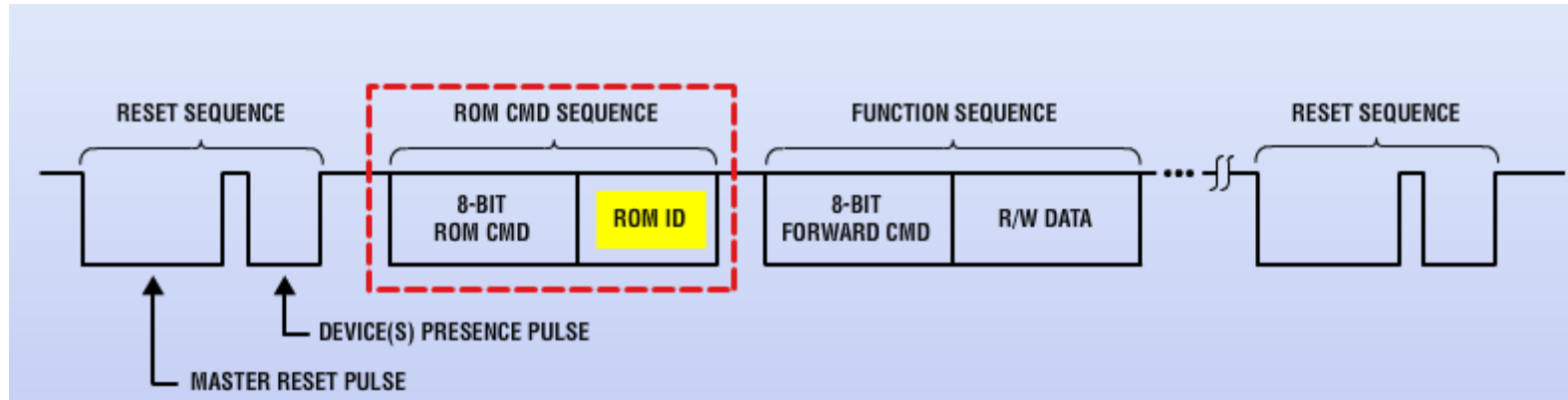
- Puis suivent les signaux d'écriture/lecture des données



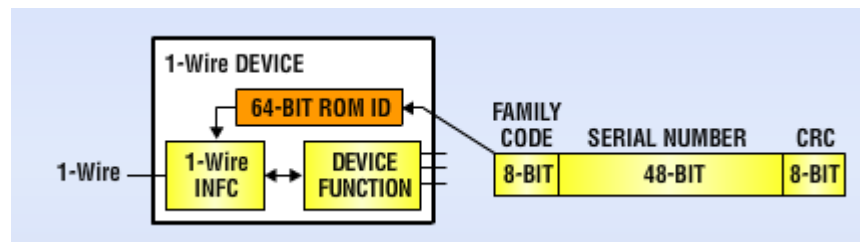
Une nouvelle fois, vous saisissez la complexité sous-jacente à l'utilisation d'un composant de ce type...

5. Technique : 1-Wire : Séquence de transmission des données

- Au final, c'est l'ensemble de la séquence suivante qui se déroule sur le bus 1-Wire :
 - séquence d'initialisation
 - envoi de la commande dite « ROM » avec notamment le numéro d'identification ou adresse
 - puis la séquence fonctionnelle avec notamment la lecture/écriture de la donnée voulue
 - puis l'ensemble se répète avec une nouvelle séquence d'initialisation, etc... :



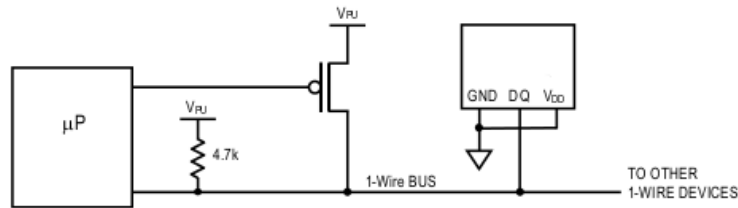
- En particulier, l'identifiant ou « ROM ID » (64 bits) est constitué :
 - du type du capteur (8 bits)
 - de l'adresse du capteur (48 bits)
 - d'un code de contrôle appelé CRC (8 bits)



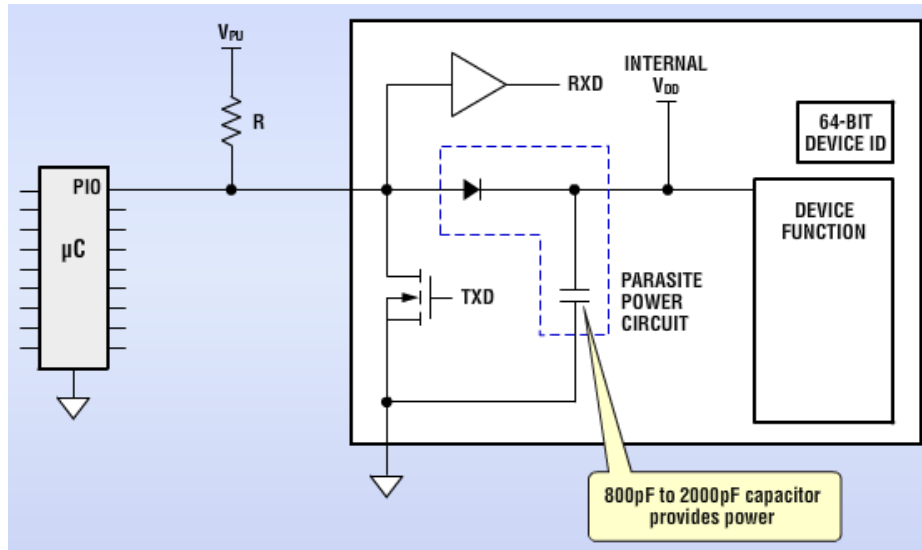
6. Info : Communication 1-Wire : Principe de l'alimentation par 1 seul fil de masse

Alimentation par un seul fil : comment est-ce possible ?

- Lorsque l'on découvre la communication 1-Wire, on se demande comment les dispositifs peuvent être alimentés... par un seul fil de masse (=0V) seulement en plus du fil de transmission série bidirectionnelle des données. Jusqu'à preuve du contraire, une alimentation, c'est au moins 2 fils, un plus (=5V) et une masse (=0V).
- La solution est suffisamment « géniale » pour que je vous en parle. Voici tout d'abord le câblage type d'un esclave 1-wire au bus :

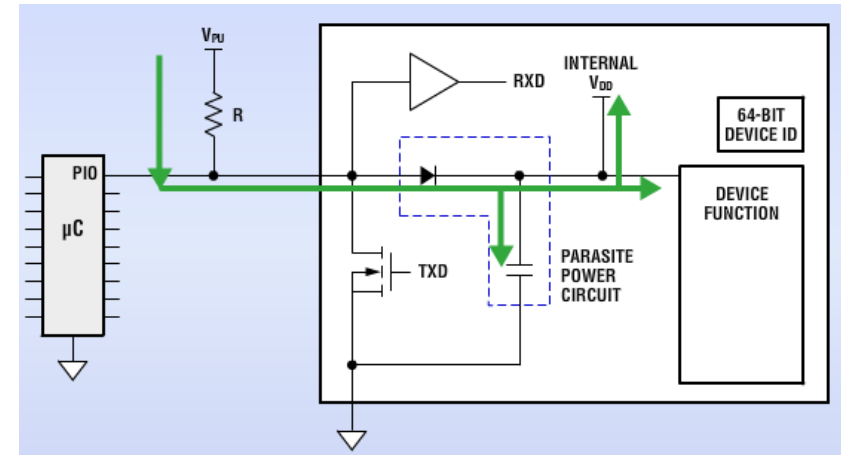


- Remarquer notamment la présence d'une résistance de « rappel au plus » de 4,7K sur la ligne de transmission des données séries.
- La « clé » du système « d'alimentation par un fil » repose sur la présence d'un condensateur interne au sein du dispositif 1-Wire :



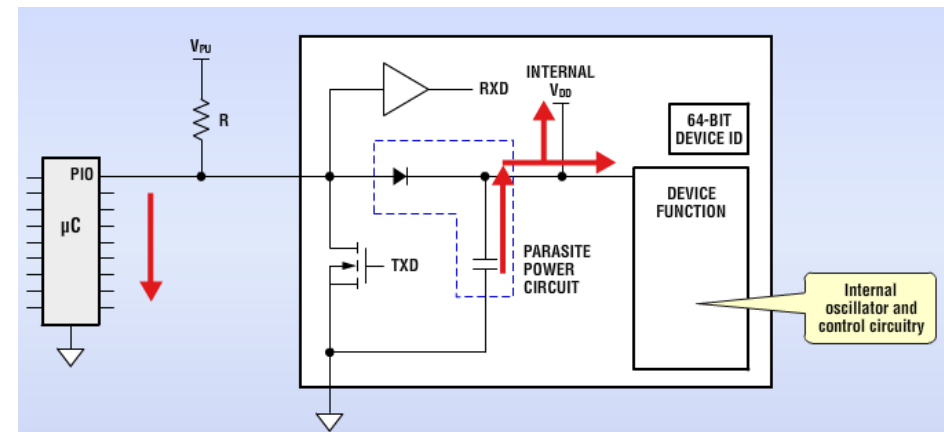
Charge lorsque le bus est au niveau HIGH

- Lorsque le bus est au niveau HAUT, le dispositif 1-Wire est alimenté... et le condensateur se charge au passage :



Décharge lorsque le bus est au niveau LOW

- Puis lorsque le bus passe au niveau BAS, le dispositif 1-wire n'est plus alimenté directement... mais par le condensateur qui agit alors comme une mini-batterie :



- Au final, comme le bus change d'état HAUT/BAS en permanence, le dispositif 1-Wire est alimenté correctement : bien vu non ?

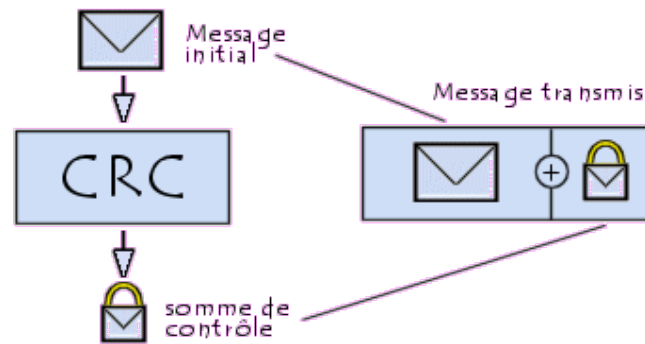
7. Pour info : Communication 1-Wire : Notion de contrôle par « Calcul de Redondance Cyclique »

- Les dispositifs 1-Wire intègrent de plus **un système permettant de contrôler que les données ont bien été transmises**, ceci en réalisant un calcul à partir des données reçues permettant d'être sûr que la transmission est correcte. Un peu à la manière d'une « somme de contrôle » quand on télécharge un fichier sur internet.
- Le système utilisé ici s'appelle le « **Calcul par Redondance Cyclique** » (ou CRC), un mot barbare pour désigner, je cite (wikipédia...) :

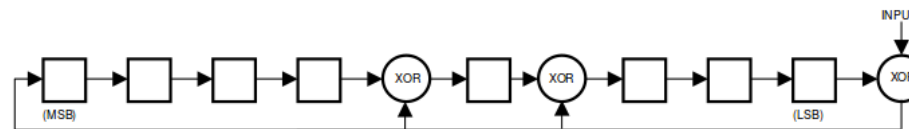
« En informatique et dans certains appareils numériques, **un contrôle de redondance cyclique ou CRC (Cyclic Redundancy Check) est un outil logiciel permettant de détecter les erreurs de transmission ou de transfert par ajout, combinaison et comparaison de données redondantes**, obtenues grâce à une procédure de hachage. Ainsi, une erreur de redondance cyclique peut survenir lors de la copie d'un support (disque dur, CD-Rom, DVD-Rom, clé USB, etc.) vers un autre support de sauvegarde. »

Les CRC sont évalués (échantillonnés) **avant et après** la transmission ou le transfert, **puis comparés** pour s'assurer que les données sont strictement identiques. Les calculs de CRC les plus utilisés sont conçus afin de pouvoir toujours détecter les erreurs de certains types, comme celles dues par exemple, aux interférences lors de la transmission. »

- Quoi ? Vous n'avez pas tout compris ? ... En clair, le message transmis par l'esclave va contenir une valeur de contrôle calculée que le « maître » va pouvoir recalculer de son côté puis la comparer avec la valeur reçue avec le message pour s'assurer de la cohérence entre le message émis et le message reçu. Voici un petit dessin qui explique la chose :



- L'algorithme interne utilisé par le générateur CRC d'un dispositif 1-wire est schématisé de la façon suivante :



**Enfin bref, non content de transmettre les données sur un fil dans les 2 sens, non content d'être alimenté par un seul fil de masse...
un dispositif 1-wire intègre en plus un système de contrôle de cohérence de la transmission des données !!**

Je ne sais pas pour vous, mais quand je vois qu'un petit capteur qui fait tout ça ne coûte que quelques euros... moi, je suis bluffé !

Et quand en plus, j'arrive à l'utiliser avec Arduino... alors là, « je suis aux anges » !

Avec toutes ces infos sur le bus 1-Wire, vous allez finir par avoir les neurones qui fument... **Je vous rassure, c'est juste pour info !**

A présent, nous allons voir un cas concret de composant 1-Wire et apprendre à l'utiliser avec Arduino : allez, on continue...

8. Exemple de dispositif 1-wire : le capteur de température DS18B20

Description

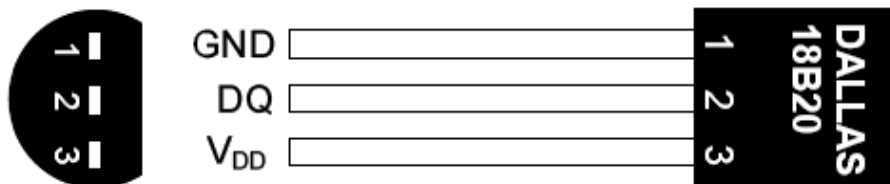
- Il s'agit d'un **capteur de température à communication 1-Wire**:
 - peu cher (2 Euros)
 - qui réalise en interne la conversion analogique numérique
 - peut être connecté simplement sur **2 fils** (technique dite 1-wire)
 - qui réalise une mesure de température sur la plage **-55°C à +125°C** sans composant supplémentaire (polyvalent donc...)
 - qui renvoie le résultat de la mesure sous forme de données série sur 12 bits (précision de 0.0625°C)
 - plusieurs capteurs peuvent être utilisés simultanément sur les 2 mêmes fils grâce à un système d'adressage de chaque capteur qui possède une adresse unique (64 bits ROM soit 8 octets).
 - Facile à utiliser avec Arduino grâce à une **bibliothèque dédiée** fournie

En un mot, vous allez pouvoir utiliser de 1 à 20 capteurs 18B20 sur une seule broche de la carte Arduino pour mesurer des températures !

Brochage

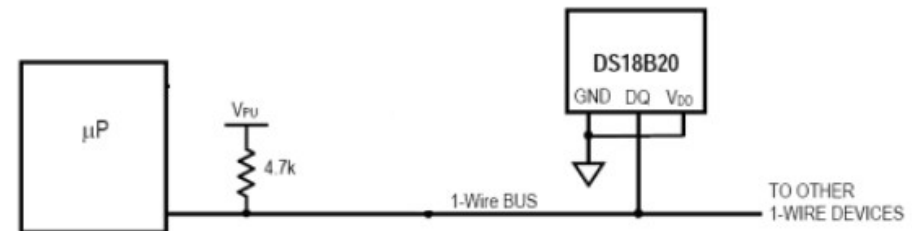
Ce capteur présente 3 broches :

- le **+5V** (Vdd)
- le **0V ou masse** (GND)
- la **broche de communication série "1-wire" entrée/sortie** (DQ) : cette broche est de type "drain ouvert" et devra être connectée au **+5V** par une **résistance de 4,7 K**.



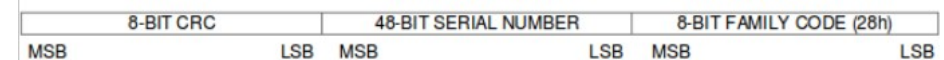
Montage type

- Pour assurer l'alimentation par la masse, on connecte les broches +5V et 0V ensemble à la masse :



Structure interne

- Le DS18B20 dispose d'une adresse 64 bits ayant la structure suivante :



- Le DS18B20 dispose de 8 octets contenant les données utiles :

Byte 0	Temperature LSB (50h)	} (85°C)
Byte 1	Temperature MSB (05h)	
Byte 2	TH Register or User Byte 1*	
Byte 3	TL Register or User Byte 2*	
Byte 4	Configuration Register*	
Byte 5	Reserved (FFh)	
Byte 6	Reserved	
Byte 7	Reserved (10h)	
Byte 8	CRC*	

- La température codée sur 12 bits sera exprimée sur 2 octets selon :

	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

Comme nous allons le voir à présent, une bibliothèque va nous permettre d'utiliser relativement simplement un tel composant 1-wire !

9. Les codes d'instructions du DS18B20

- Assez logiquement, comme la plupart des dispositifs à communication série, le DS18B20 est capable de reconnaître des instructions, sous forme d'un code hexadécimal, qui permettent au « maître » de dire ce qu'il veut : obtenir une mesure, lire les registres, etc...
- Voici le tableau résumant ces instructions tel qu'il est présent dans la documentation du capteur :

Table 3. DS18B20 Function Command Set

COMMAND	DESCRIPTION	PROTOCOL	1-Wire BUS ACTIVITY AFTER COMMAND IS ISSUED
TEMPERATURE CONVERSION COMMANDS			
Convert T	Initiates temperature conversion.	44h	DS18B20 transmits conversion status to master (not applicable for parasite-powered DS18B20s).
MEMORY COMMANDS			
Read Scratchpad	Reads the entire scratchpad including the CRC byte.	BEh	DS18B20 transmits up to 9 data bytes to master.
Write Scratchpad	Writes data into scratchpad bytes 2, 3, and 4 (T _H , T _L , and configuration registers).	4Eh	Master transmits 3 data bytes to DS18B20.
Copy Scratchpad	Copies T _H , T _L , and configuration register data from the scratchpad to EEPROM.	48h	None
Recall E ²	Recalls T _H , T _L , and configuration register data from EEPROM to the scratchpad.	B8h	DS18B20 transmits recall status to master.
Read Power Supply	Signals DS18B20 power supply mode to the master.	B4h	DS18B20 transmits supply status to master.

- En clair, voici la signification des codes d'instructions (0x signifie que la valeur est hexadécimale) :
 - 0x44 : Lancer une mesure de la température
 - 0xBE : Lecture des 9 registres du capteur
 - 0x4E : Ecrire dans les registres d'alarmes et de configuration du capteur
 - 0x48 : Copie la valeur des alarmes et de la configuration dans l'Eeprom du capteur
 - 0xB8 : Recharge la valeur des alarmes et de la configuration stockées dans l'Eeprom du capteur
 - 0xB4 : Information sur le statut d'alimentation du capteur
- Certains de ces codes nous serviront pour la suite. Nous en reparlerons le moment voulu.

10. Utiliser un DS18B20 (capteur 1-wire) avec la librairie Arduino OneWire

La librairie Arduino OneWire

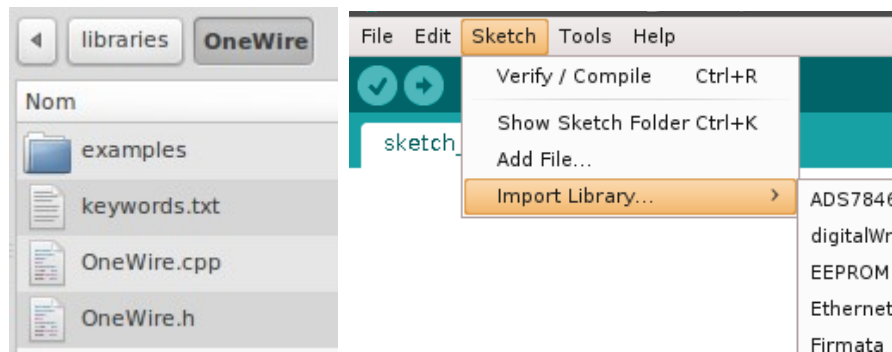
- La librairie OneWire permet d'utiliser un DS18B20 avec Arduino en fournissant toutes les fonctionnalités utiles, notamment en ce qui concerne le contrôle de validité de la mesure (CRC), l'initialisation du capteur, la lecture du résultat de la mesure.

Télécharger la librairie

- La librairie OneWire est disponible ici : http://www.pjrc.com/teensy/td_libs_OneWire.html
- La documentation de la librairie est disponible ici : http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.LibrairieOneWire

Installation

- Télécharger l'archive. au format zip ou autre. L'extraire
- Vérifier que le nom du répertoire de la librairie est strictement le même que le nom du fichier *.h ou *.cpp principal. Corriger au besoin. Ici le nom est **OneWire**
- Copier/coller le répertoire de la librairie dans le répertoire libraries de votre répertoire Arduino
- Relancer Arduino et vérifier que la librairie est présente dans le menu **Sketch > ImportLibrary**.



Le constructeur principal

- On déclare l'objet OneWire avec le constructeur :

```
OneWire myWire(broche) ; // déclare capteur et broche
```

Fonctions de la librairie

Fonctions d'initialisation

- `myWire.search(addrArray)`
- `myWire.reset_search()`
- `myWire.skip()`

Fonctions de communication

- `myWire.reset()`
- `myWire.select(addrArray)`
- `myWire.write(num)`
- `myWire.write(num, 1)`
- `myWire.read()`

Fonction de contrôle des données

- `myWire.crc8(dataArray, length)`

Fonction de contrôle de de l'alimentation

- `myWire.depower()`

Pour le détail des fonctions, voir :

http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.LibrairieOneWire

Code d'exemple

```
#include <OneWire.h> // librairie pour capteur OneWire

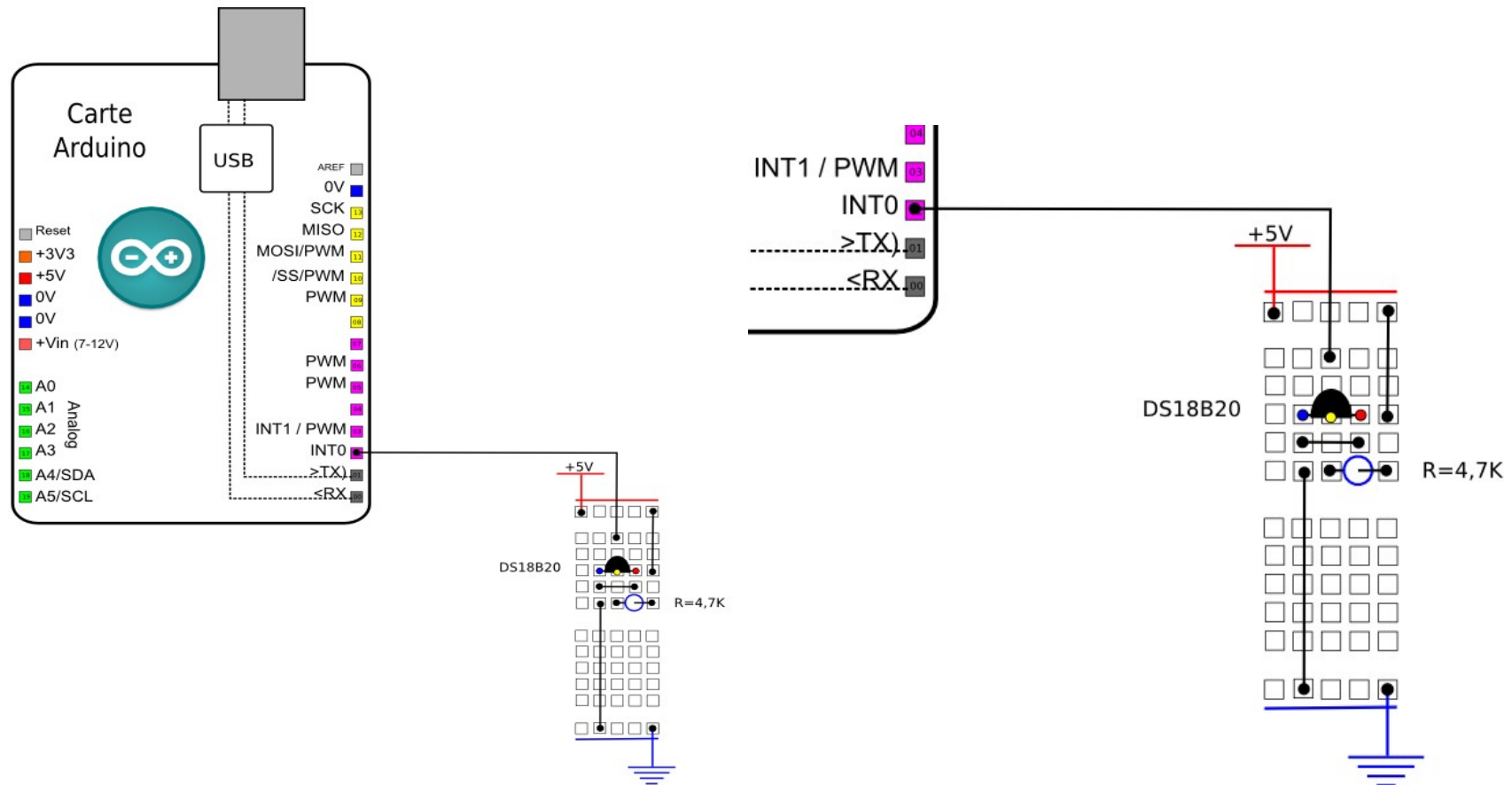
OneWire myWire(broche) ; // déclare capteur et broche

void setup () {
    // ici les fonctions d'initialisation
}

void loop () {
    // ici les fonctions de contrôle et de lecture des données
}
```

11. Exemple 1-wire : DS18B20 : Détecter le capteur et obtenir son adresse : le montage

- On connecte le DS18B20 pour une connexion 2 fils avec :
 - la broche +V connectée à la broche 0V
 - la broche 0V connectée à la masse
 - la broche de donnée connectée à la broche Arduino voulue, ici la 2
 - et la broche de donnée également raccordée au + à l'aide d'une résistance de rappel au plus (pull-up) de 4,7K.



12. Exemple 1-wire : DS18B20 : Détecter le capteur et obtenir son adresse : le programme

Ce qu'on va faire ici...

- Pour prendre en main notre capteur, on va se contenter de le détecter et d'afficher son adresse 64 bits afin d'en extraire les éléments utiles.

Entête déclarative

Inclusion des bibliothèques utiles

- On commence par inclure les bibliothèques
 - on inclut la bibliothèque **OneWire** qui permet d'utiliser un dispositif 1-wire. Cette bibliothèque doit avoir été installée dans le répertoire <Libraries>

Déclaration broche utilisée

- On déclare la broche utilisée avec le capteur. **Une même broche pourra être utilisée pour autant de capteurs qu'on le souhaite !**

Variables utiles

- On déclare un tableau de 8 octets qui va servir à stocker l'adresse 64 bits du capteur (8 octets x 8 bits = 64 bits) .

Objet utile

- On déclare un objet OneWire qui représente le capteur 1-wire utilisé, ici le DS18B20, en précisant la broche utilisée.

```
// --- Inclusion des bibliothèques utilisées ---
#include <OneWire.h> // bibliothèque pour capteur OneWire

// --- Déclaration des constantes ---
// --- constantes des broches ---

const int broche_OneWire=2; //déclaration constante de broche

// --- Déclaration des variables globales ---
byte adresse[8]; // Tableau de 8 octets pour stockage du code d'adresse 64 bits du composant One Wire

// --- Déclaration des objets utiles pour les fonctionnalités utilisées ---
OneWire capteur(broche_OneWire); // crée un objet One Wire sur la broche voulue
```

Fonction **setup()**

Initialisation série

- On initialise le port série à 115200 Bauds

Détection du capteur

- Un capteur est détecté à l'aide de la fonction **search()** qui renvoie **true** si un capteur est présent, **False** sinon.
- On attend qu'un capteur soit détecté en plaçant la fonction **search** au sein d'une boucle **while()**
- Une fois qu'un capteur est présent/détecté, on affiche un message

Affichage de l'adresse 64 bits au format hexadécimal

- La fonction **search()** précédemment appelée a reçu en paramètre le tableau de 8 octets déclaré dans l'entête : l'adresse 64 bits va automatiquement être stockée dans ce tableau.
- Ensuite, à l'aide d'une simple boucle **for**, on défile les octets du tableau et on les affiche au format hexadécimal :

```
void setup() { // debut de la fonction setup()

// --- ici instructions à exécuter au démarrage ---

Serial.begin(115200); // initialise connexion série à 115200 bauds
// IMPORTANT : régler le terminal côté PC avec la même valeur de transmission

//---- détection des capteurs présents sur le bus One Wire

Serial.println("*** Capteur present sur le bus 1-wire *** ");

while (capteur.search(adresse)!= true) { } // attend qu'un capteur soit détecté
// la fonction search renvoie la valeur VRAI si un élément 1-wire est trouvé. Stocke son adresse dans le tableau adresse
// adresse correspond à l'adresse de début du tableau adresse[8] déclaré ...

Serial.print (" 1 capteur 1-wire present avec code adresse 64 bits : ");

//--- affichage des 64 bits d'adresse au format hexadécimal
for(int i = 0; i < 8; i++) { // l'adresse renvoyée par la fonction search est stockée sur 8 octets

    if (adresse[i]<16) Serial.print('0'); // pour affichage des 0 poids fort au format hexadécimal
    Serial.print(adresse[i], HEX); // affiche 1 à 1 les 8 octets du tableau adresse au format hexadécimal
    Serial.print(" ");

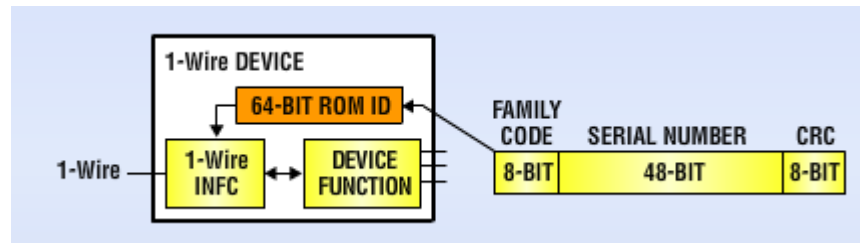
} // fin for

Serial.println();
```

Fonction **setup()** (suite)

Vérification du type du capteur

- Comme nous l'avons dit précédemment, l'adresse 64 bits est constituée de la façon suivante :



- Le premier octet correspond au type du capteur et vaut 0x28 pour un capteur DS18B20, 0x10 pour un DS18S20, 0x22 pour un DS1820. Ici, on se contente de vérifier que nous avons bien à faire à un 18B20 (code 0x28) :

Note : un nombre sous la forme **0x123** correspond au nombre hexadécimal **123**

Vérification du code de contrôle

- La librairie OneWire intègre une fonction, la fonction **crc8()** qui permet de contrôler la validité de toute donnée reçue en provenance du capteur. Il suffit de vérifier la concordance entre le code CRC reçu et le code fourni par la fonction **crc8** pour être sûr que la transmission des données reçues est valide. Ici, on teste tout simplement la validité du code CRC de l'adresse reçue :

```
//---- test du type de capteur ----
// le type du capteur est donné par le 1er octet du code adresse 64 bits
// Valeur 0x28 pour capteur type DS18B20, 0x10 pour type DS18S20, 0x22 pour type DS1820
if (adresse[0]==0x28) Serial.println ("Type : Capteur temperature DS18B20.");

//----- contrôle du code CRC ----
// le dernier octet de l'adresse 64bits est un code de contrôle CRC
// à l'aide de la fonction crc8 on peut vérifier si ce code est valide
if (capteur.crc8( adresse, 7) == adresse[7]) // vérification validité code CRC de l'adresse 64 bits
// le code CRC de l'adresse 64 bits est le 8ème octet de l'adresse (index 7 du tableau)
{
    Serial.println ("Verification du code CRC de l'adresse 64 bits de ce capteur : VALIDE !");
}
else
{
    Serial.println ("Verification du code CRC de l'adresse 64 bits de ce capteur : NON VALIDE !");
}

Serial.println("-----");

} // fin de la fonction setup()
```

Pour plus de détails, voir : http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.LibrairieOneWirecrc8

Fonction **loop()**

- Ici, la fonction loop() est laissée vide

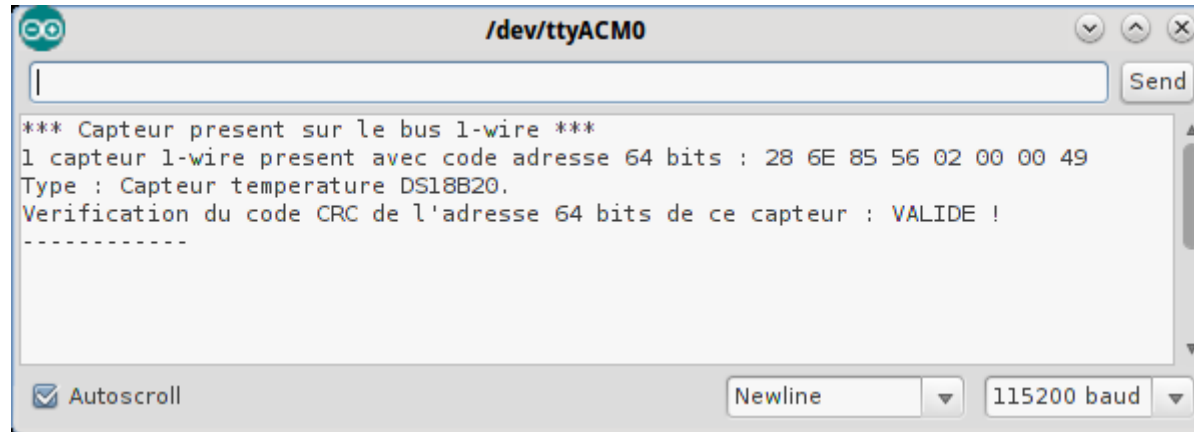
```
void loop(){ // debut de la fonction loop()

// --- ici instructions à exécuter par le programme principal ---

} // fin de la fonction loop() - le programme recommence au début de la fonction loop sans fin
```


Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne ...



```
*** Capteur present sur le bus 1-wire ***
1 capteur 1-wire present avec code adresse 64 bits : 28 6E 85 56 02 00 00 49
Type : Capteur temperature DS18B20.
Verification du code CRC de l'adresse 64 bits de ce capteur : VALIDE !
-----
```

Récupération de l'adresse 64 bits du capteur, test du type du capteur, vérification de la validité du code de contrôle CRC...
C'est cool non de pouvoir faire ça aussi simplement non ?



Il est sympa ce petit capteur.... Allez, on continue sur notre lancée !
Et je ne vais pas me contenter de vous montrer comment faire une mesure :
je vais vous montrer en détail comment ce capteur fonctionne, car c'est très instructif je trouve, et pas trop compliqué...

13. Exemple 1-wire : DS18B20 : Afficher le contenu des registres de données : le programme

Ce qu'on va faire ici...

- Une fois que l'on est capable de détecter le capteur 1-wire et d'afficher son adresse 64 bits afin d'en extraire les éléments utiles, nous allons pouvoir à présent obtenir le contenu des registres de données.
- Je rappelle ici que ce capteur dispose de 9 registres de données (appelé aussi le « scratchpad ») qui sont décrits dans la documentation du capteur :

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*	

- Rappelons également que le capteur est capable de « comprendre » certaines instructions : le code lire les registres du capteurs est **0xBE**.
- Nous reprenons ici le même montage que précédemment.

Entête déclarative

Inclusion des bibliothèques utiles

- On commence par inclure les bibliothèques
 - on inclut la bibliothèque **OneWire** qui permet d'utiliser un dispositif 1-wire. Cette bibliothèque doit avoir été installée dans le répertoire <Libraries>

Déclaration broche utilisée

- On déclare la broche utilisée avec le capteur. **Une même broche pourra être utilisée pour autant de capteurs qu'on le souhaite !**

Variables utiles

- On déclare un tableau de 8 octets qui va servir à stocker l'adresse 64 bits du capteur (8 octets x 8 bits = 64 bits) .
- On déclare également un tableau de 12 octets qui va servir à stocker les 9 registres de la RAM et 3 registres d'Eeprom du capteur.

Codes d'instruction

- On déclare également une constante hexadécimale correspondant à l'instruction « Lecture des Registres », soit le code hexadécimal 0xBE

Objet utile

- On déclare un objet OneWire qui représente le capteur 1-wire utilisé, ici le DS18B20, en précisant la broche utilisée.

```
// --- Inclusion des bibliothèques utilisées ---
#include <OneWire.h> // bibliothèque pour capteur OneWire

// --- Déclaration des constantes ---
// --- constantes des broches ---

const int broche_OneWire=2; //déclaration constante de broche

// --- Déclaration des variables globales ---
byte adresse[8]; // Tableau de 8 octets pour stockage du code d'adresse 64 bits du composant One Wire
byte data[12]; // Tableau de 12 octets pour lecture des 9 registres de RAM et des 3 registres d'EEPROM du capteur One Wire

//---- code d'instruction du capteur
const int modeLecture=0xBE;

// --- Déclaration des objets utiles pour les fonctionnalités utilisées ---
OneWire capteur(broche_OneWire); // crée un objet One Wire sur la broche voulue
```

Fonction **setup()**

Initialisation série

- On initialise le port série à 115200 Bauds

Détection du capteur

- Un capteur est détecté à l'aide de la fonction **search()** qui renvoie **true** si un capteur est présent, **False** sinon.
- On attend qu'un capteur soit détecté en plaçant la fonction **search** au sein d'une boucle **while()**
- Une fois qu'un capteur est présent/détecté, on affiche un message

Affichage de l'adresse 64 bits au format hexadécimal

- La fonction **search()** précédemment appelée a reçu en paramètre le tableau de 8 octets déclaré dans l'entête : l'adresse 64 bits va automatiquement être stockée dans ce tableau.
- Ensuite, à l'aide d'une simple boucle **for**, on défile les octets du tableau et on les affiche au format hexadécimal :

```
void setup() { // debut de la fonction setup()

// --- ici instructions à exécuter au démarrage ---

Serial.begin(115200); // initialise connexion série à 115200 bauds
// IMPORTANT : régler le terminal côté PC avec la même valeur de transmission

//---- détection des capteurs présents sur le bus One Wire

Serial.println("*** Capteur present sur le bus 1-wire *** ");

while (capteur.search(adresse)!= true) { } // attend qu'un capteur soit détecté
// la fonction search renvoie la valeur VRAI si un élément 1-wire est trouvé. Stocke son adresse dans le tableau adresse
// adresse correspond à l'adresse de début du tableau adresse[8] déclaré ...

Serial.print (" 1 capteur 1-wire present avec code adresse 64 bits : ");

//--- affichage des 64 bits d'adresse au format hexadécimal
for(int i = 0; i < 8; i++) { // l'adresse renvoyée par la fonction search est stockée sur 8 octets

    if (adresse[i]<16) Serial.print('0'); // pour affichage des 0 poids fort au format hexadécimal
    Serial.print(adresse[i], HEX); // affiche 1 à 1 les 8 octets du tableau adresse au format hexadécimal
    Serial.print(" ");

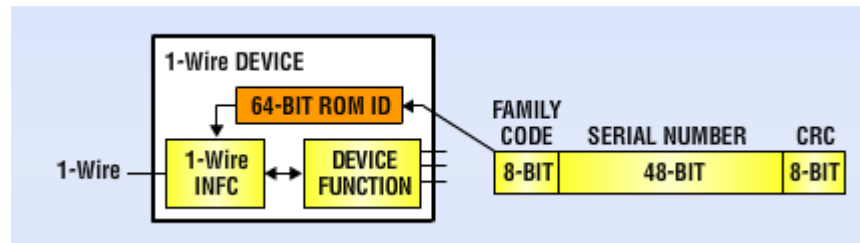
} // fin for

Serial.println();
```

Fonction **setup()** (suite)

Vérification du type du capteur

- Comme nous l'avons dit précédemment, l'adresse 64 bits est constituée de la façon suivante :



- Le premier octet correspond au type du capteur et vaut 0x28 pour un capteur DS18B20, 0x10 pour un DS18S20, 0x22 pour un DS1820. Ici, on se contente de vérifier que nous avons bien à faire à un 18B20 (code 0x28) :

Note : un nombre sous la forme **0x123** correspond au nombre hexadécimal **123**

Vérification du code de contrôle

- La bibliothèque OneWire intègre une fonction, la fonction **crc8()** qui permet de contrôler la validité de toute donnée reçue en provenance du capteur. Il suffit de vérifier la concordance entre le code CRC reçu et le code fourni par la fonction **crc8** pour être sûr que la transmission des données reçues est valide. Ici, on teste tout simplement la validité du code CRC de l'adresse reçue :

```
//---- test du type de capteur ----
// le type du capteur est donné par le 1er octet du code adresse 64 bits
// Valeur 0x28 pour capteur type DS18B20, 0x10 pour type DS18S20, 0x22 pour type DS1820
if (adresse[0]==0x28) Serial.println ("Type : Capteur temperature DS18B20.");

//----- contrôle du code CRC -----
// le dernier octet de l'adresse 64bits est un code de contrôle CRC
// à l'aide de la fonction crc8 on peut vérifier si ce code est valide
if (capteur.crc8( adresse, 7) == adresse[7]) // vérification validité code CRC de l'adresse 64 bits
// le code CRC de l'adresse 64 bits est le 8ème octet de l'adresse (index 7 du tableau)
{
    Serial.println ("Verification du code CRC de l'adresse 64 bits de ce capteur : VALIDE !");
}
else
{
    Serial.println ("Verification du code CRC de l'adresse 64 bits de ce capteur : NON VALIDE !");
}

Serial.println("-----");
```

Pour plus de détails, voir : http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.LibrairieOneWirecrc8

Fonction **setup()** (suite)

- A présent on lance une lecture des registres en envoyant l'instruction de lecture vers le capteur avec la séquence **reset()** - **select()** - **write()** utilisant les fonctions de la librairie **OneWire()**
- Une fois fait, on lit les données reçues avec l'instruction **read()**
- Les données sont stockées dans le tableau d'octets déclaré précédemment : avec une simple boucle, on défile les 9 premiers octets et on affiche la valeur binaire. Le but ici est essentiellement didactique pour vous montrer comment ça fonctionne.

```
//----- passer en mode LECTURE -----
capteur.reset(); // initialise le bus 1-wire avant la communication avec un capteur donné
capteur.select(adresse); // sélectionne le capteur ayant l'adresse 64 bits contenue dans le tableau envoyé à la fonction
capteur.write(modeLecture,1); // passe en mode lecture de la RAM du capteur

// ----- lire les 9 octets de la RAM (appelé Scratchpad) ----

for ( int i = 0; i < 9; i++) {           // 9 octets de RAM stockés dans 9 octets
    data[i] = capteur.read();           // lecture de l'octet de rang i stocké dans tableau data
}

// ----- affichage du contenu des différents octets -----
Serial.println("");
Serial.println("---- lecture des 9 registres de la RAM du capteur ---- ");
Serial.print("Octet 0 (Temperature bits poids faible)=", Serial.println(data[0],BIN));
Serial.print("Octet 1 (Temperature bits poids fort)=", Serial.println(data[1],BIN));
Serial.print("Octet 2 (Alarme haute)=", Serial.println(data[2],BIN));
Serial.print("Octet 3 (Alarme basse)=", Serial.println(data[3],BIN));
Serial.print("Octet 4 (Registre de configuration)=", Serial.println(data[4],BIN));
Serial.print("Octet 5 (Reserve)=", Serial.println(data[5],BIN));
Serial.print("Octet 6 (Reserve)=", Serial.println(data[6],BIN));
Serial.print("Octet 7 (Reserve)=", Serial.println(data[7],BIN));
Serial.print("Octet 8 (code CRC mesure)=", Serial.println(data[8],BIN));
```

Fonction **setup** () (suite)

- A présent, comme on l'a fait pour l'adresse 64 bits, nous allons tester la cohérence des données reçues à l'aide du code CRC inclut dans les 9 registres : le code CRC est le 9ème. Nous allons passer sa valeur à la fonction `crc8()` comme fait précédemment, qui renverra `True` si le code est valide :

```
//----- test de validité des valeurs reçues par contrôle du code CRC -----  
  
Serial.println("");  
Serial.println("---- test de controle de validite des donnees recues ---- ");  
  
// le dernier (9ème) octet de la RAM est un code de contrôle CRC  
// à l'aide de la fonction crc8 on peut vérifier si ce code est valide  
if (capteur.crc8( data, 8) == data[8]) // vérification validité code CRC des valeurs reçues  
{  
    Serial.println ("Verification du code CRC des Registres : VALIDE !");  
}  
else  
{  
    Serial.println ("Verification du code CRC des Registres : NON VALIDE !");  
}  
  
} // fin de la fonction setup()
```

Pour plus de détails, voir : http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.LibrairieOneWirecrc8

Fonction **loop()**

- Ici, la fonction `loop()` est laissée vide

```
void loop(){ // debut de la fonction loop()

// --- ici instructions à exécuter par le programme principal ---

} // fin de la fonction loop() - le programme recommence au début de la fonction loop sans fin
```

Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne ...



```
*** Capteur present sur le bus 1-wire ***
1 capteur 1-wire present avec code adresse 64 bits : 28 6E 85 56 02 00 00 49
Type : Capteur temperature DS18B20.
Verification du code CRC de l'adresse 64 bits de ce capteur : VALIDE !
-----

---- lecture des 9 registres de la RAM du capteur ----
Octet 0 (Temperature bits poids faible)=1010000
Octet 1 (Temperature bits poids fort)=101
Octet 2 (Alarme haute)=1001011
Octet 3 (Alarme basse)=1000110
Octet 4 (Registre de configuration)=1111111
Octet 5 (Reserve)=1111111
Octet 6 (Reserve)=1100
Octet 7 (Reserve)=10000
Octet 8 (code CRC mesure)=11100

---- test de controle de validite des donnees recues ----
Verification du code CRC des Registres : VALIDE !
```

Vous venez d'accéder aux registres internes du capteur et vous avez vérifié la validité des données : cool non ?

Au fait, vous avez peut-être envie de réaliser une mesure de... température : je vous rassure, on y arrive.

14. Exemple 1-wire : DS18B20 : Lancer une mesure, lire et contrôler et afficher la mesure obtenue.

Ce qu'on va faire ici...

- Bien, arriver à ce stade, « ça va être du gâteau ! »
- Je vous rappelle ici que les 9 registres de données du capteur :

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*	

- Si vous êtes attentif, vous remarquerez que les octets 0 et 1 sont les octets qui stockent le résultat de la dernière mesure.
- Nous allons avoir 2 choses supplémentaires à faire ici :
 - lancer une mesure avant de lire les registres, ce qui sera fait avec le code d'instruction 0x44
 - puis convertir le résultat exprimé sur 2 octets en une valeur décimale en degrés.
- Nous reprenons ici le même montage que précédemment.

Entête déclarative

Inclusion des bibliothèques utiles

- On commence par inclure les bibliothèques
 - on inclut la bibliothèque **OneWire** qui permet d'utiliser un dispositif 1-wire. Cette bibliothèque doit avoir été installée dans le répertoire <Libraries>

Déclaration broche utilisée

- On déclare la broche utilisée avec le capteur. **Une même broche pourra être utilisée pour autant de capteurs qu'on le souhaite !**

Variables utiles

- On déclare un tableau de 8 octets qui va servir à stocker l'adresse 64 bits du capteur (8 octets x 8 bits = 64 bits) .
- On déclare également un tableau de 12 octets qui va servir à stocker les 9 registres de la RAM et 3 registres d'Eeprom du capteur.
- On déclare 2 variables pour la mesure de la température.

Codes d'instruction

- On déclare également une constante hexadécimale correspondant à l'instruction « Lecture des Registres », soit le code hexadécimal 0xBE

Objet utile

- On déclare un objet OneWire qui représente le capteur 1-wire utilisé, ici le DS18B20, en précisant la broche utilisée.

```
// --- Inclusion des bibliothèques utilisées ---
#include <OneWire.h> // bibliothèque pour capteur OneWire

// --- Déclaration des constantes ---
// --- constantes des broches ---

const int broche_OneWire=2; //déclaration constante de broche

// --- Déclaration des variables globales ---
byte adresse[8]; // Tableau de 8 octets pour stockage du code d'adresse 64 bits du composant One Wire
byte data[12]; // Tableau de 12 octets pour lecture des 9 registres de RAM et des 3 registres d'EEPROM du capteur One Wire

int temperature=0; // variable entière pour stocker la température brute
float temperaturef=0.0; // variable décimale pour stocker la température réelle

//---- code d'instruction du capteur
const int modeLecture=0xBE;
const int lancerMesure=0x44;

// --- Déclaration des objets utiles pour les fonctionnalités utilisées ---
OneWire capteur(broche_OneWire); // crée un objet One Wire sur la broche voulue
```

Fonction **setup()**

Initialisation série

- On initialise le port série à 115200 Bauds

Détection du capteur

- Un capteur est détecté à l'aide de la fonction **search()** qui renvoie **true** si un capteur est présent, **False** sinon.
- On attend qu'un capteur soit détecté en plaçant la fonction **search** au sein d'une boucle **while()**
- Une fois qu'un capteur est présent/détecté, on affiche un message

Affichage de l'adresse 64 bits au format hexadécimal

- La fonction **search()** précédemment appelée a reçu en paramètre le tableau de 8 octets déclaré dans l'entête : l'adresse 64 bits va automatiquement être stockée dans ce tableau.
- Ensuite, à l'aide d'une simple boucle **for**, on défile les octets du tableau et on les affiche au format hexadécimal :

```
void setup() { // debut de la fonction setup()

// --- ici instructions à exécuter au démarrage ---

Serial.begin(115200); // initialise connexion série à 115200 bauds
// IMPORTANT : régler le terminal côté PC avec la même valeur de transmission

//---- détection des capteurs présents sur le bus One Wire

Serial.println("*** Capteur present sur le bus 1-wire *** ");

while (capteur.search(adresse)!= true) { } // attend qu'un capteur soit détecté
// la fonction search renvoie la valeur VRAI si un élément 1-wire est trouvé. Stocke son adresse dans le tableau adresse
// adresse correspond à l'adresse de début du tableau adresse[8] déclaré ...

Serial.print (" 1 capteur 1-wire present avec code adresse 64 bits : ");

//--- affichage des 64 bits d'adresse au format hexadécimal
for(int i = 0; i < 8; i++) { // l'adresse renvoyée par la fonction search est stockée sur 8 octets

    if (adresse[i]<16) Serial.print('0'); // pour affichage des 0 poids fort au format hexadécimal
    Serial.print(adresse[i], HEX); // affiche 1 à 1 les 8 octets du tableau adresse au format hexadécimal
    Serial.print(" ");

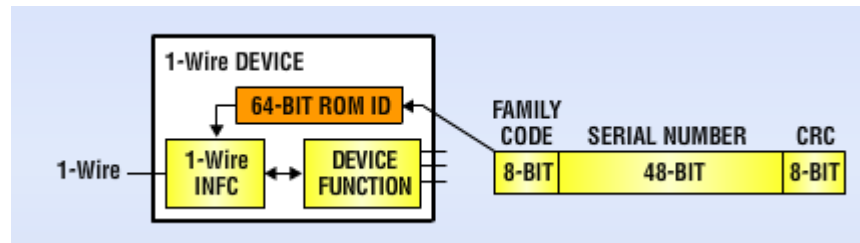
} // fin for

Serial.println();
```

Fonction **setup()** (suite)

Vérification du type du capteur

- Comme nous l'avons dit précédemment, l'adresse 64 bits est constituée de la façon suivante :



- Le premier octet correspond au type du capteur et vaut 0x28 pour un capteur DS18B20, 0x10 pour un DS18S20, 0x22 pour un DS1820. Ici, on se contente de vérifier que nous avons bien à faire à un 18B20 (code 0x28) :

Note : un nombre sous la forme **0x123** correspond au nombre hexadécimal **123**

Vérification du code de contrôle

- La bibliothèque OneWire intègre une fonction, la fonction **crc8()** qui permet de contrôler la validité de toute donnée reçue en provenance du capteur. Il suffit de vérifier la concordance entre le code CRC reçu et le code fourni par la fonction **crc8** pour être sûr que la transmission des données reçues est valide. Ici, on teste tout simplement la validité du code CRC de l'adresse reçue :

```
//---- test du type de capteur ----
// le type du capteur est donné par le 1er octet du code adresse 64 bits
// Valeur 0x28 pour capteur type DS18B20, 0x10 pour type DS18S20, 0x22 pour type DS1820
if (adresse[0]==0x28) Serial.println ("Type : Capteur temperature DS18B20.");

//----- contrôle du code CRC -----
// le dernier octet de l'adresse 64bits est un code de contrôle CRC
// à l'aide de la fonction crc8 on peut vérifier si ce code est valide
if (capteur.crc8( adresse, 7) == adresse[7]) // vérification validité code CRC de l'adresse 64 bits
// le code CRC de l'adresse 64 bits est le 8ème octet de l'adresse (index 7 du tableau)
{
    Serial.println ("Verification du code CRC de l'adresse 64 bits de ce capteur : VALIDE !");
}
else
{
    Serial.println ("Verification du code CRC de l'adresse 64 bits de ce capteur : NON VALIDE !");
}

Serial.println("-----");
} // fin de la fonction setup()
```

Pour plus de détails, voir : http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.LibrairieOneWirecrc8

Fonction **loop()**

- On commence par lancer une mesure avec l'instruction 0x44 à l'aide de la séquence avec la séquence **reset()** - **select()** - **write()** utilisant les fonctions de la librairie **OneWire()**
- Puis ensuite, on attend une seconde pour laisser le temps au capteur de réaliser la mesure.

```
void loop(){ // debut de la fonction loop()

//----- lancer une mesure -----
capteur.reset(); // initialise le bus 1-wire avant la communication avec un capteur donné
capteur.select(adresse); // sélectionne le capteur ayant l'adresse 64 bits contenue dans le tableau envoyé à la fonction
capteur.write(lancerMesure,1); // lance la mesure et alimente le capteur par la broche de donnée

//----- pause d'une seconde -----
delay(1000); // au moins 750 ms
// il faudrait mettre une instruction capteur.depower ici, mais le reset va le faire
```

Fonction `loop()` (suite)

- Puis on lance une lecture des registres en envoyant l'instruction de lecture vers le capteur avec la séquence `reset()` - `select()` - `write()` utilisant les fonctions de la librairie `OneWire()`
- Une fois fait, on lit les données reçues avec l'instruction `read()`
- Ensuite, on réalise le test de validité des données à l'aide de la fonction `crc8()` et du code CRC reçu (octet 8)

```
// ----- lire les 9 octets de la RAM (appelé Scratchpad) ----  
  
for ( int i = 0; i < 9; i++) {           // 9 octets de RAM stockés dans 9 octets  
    data[i] = capteur.read();           // lecture de l'octet de rang i stocké dans tableau data  
}  
  
//----- test de validité des valeurs reçues par contrôle du code CRC ----  
  
Serial.println("");  
Serial.println("---- Resultat de mesure de la temperature ---- ");  
  
// le dernier (9ème) octet de la RAM est un code de contrôle CRC  
// à l'aide de la fonction crc8 on peut vérifier si ce code est valide  
if (capteur.crc8( data, 8) == data[8]) // vérification validité code CRC des valeurs reçues  
{  
    Serial.println ("Verification du code CRC des Registres : VALIDE !");  
}  
else  
{  
    Serial.println ("Verification du code CRC des Registres : NON VALIDE !");  
}
```

Fonction **loop()** (suite)

- A présent, nous allons lire les 2 octets contenant la valeur de la mesure. Je rappelle que les 2 octets de la mesure ont le format suivant :

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
LS BYTE	2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰
	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

- et par un jeu de décalage des bits (on ne garde que les bits utiles de poids fort, on décale vers 8 bits à gauche et on ajoute les bits de poids faible) : on obtient la valeur finale entière correspondante qui est stockée dans une seule variable int.
- Puis, on réalise la conversion de la valeur pour obtenir la valeur décimale en degrés Celsius,
- ce qui donne :

```
//----- réalisation d'une mesure de température---
data[1]=data[1] & B10000111; // met à 0 les bits de signes inutiles
temperature=data[1]; // bits de poids fort
temperature=temperature<<8; // décalage de 8 bits vers la gauche = vers les bits de poids fort
temperature=temperature+data[0]; // bits de poids faible ici mis dans les 8 premiers bits

Serial.print ("Mesure brute =");
Serial.println (temperature);

// --- en mode 12 bits, la résolution est de 0.0625°C - cf datasheet DS18B20
temperaturef=float(temperature)*6.25;
temperaturef=temperaturef/100.0;

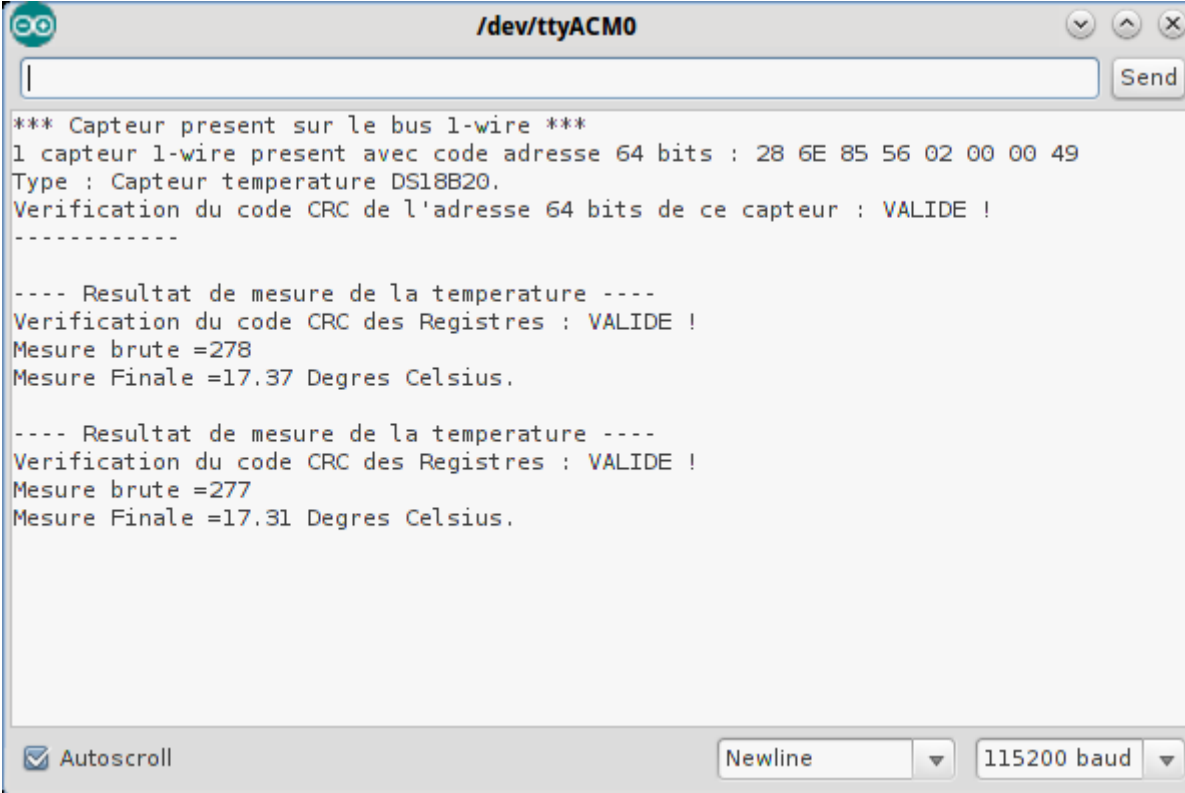
Serial.print ("Mesure Finale =");
Serial.print (temperaturef,2);
Serial.println (" Degres Celsius. ");

delay(1000); // entre 2 mesures

} // fin de la fonction loop() - le programme recommence au début de la fonction loop sans fin
```

Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne ...



```
*** Capteur present sur le bus 1-wire ***
1 capteur 1-wire present avec code adresse 64 bits : 28 6E 85 56 02 00 00 49
Type : Capteur temperature DS18B20.
Verification du code CRC de l'adresse 64 bits de ce capteur : VALIDE !
-----

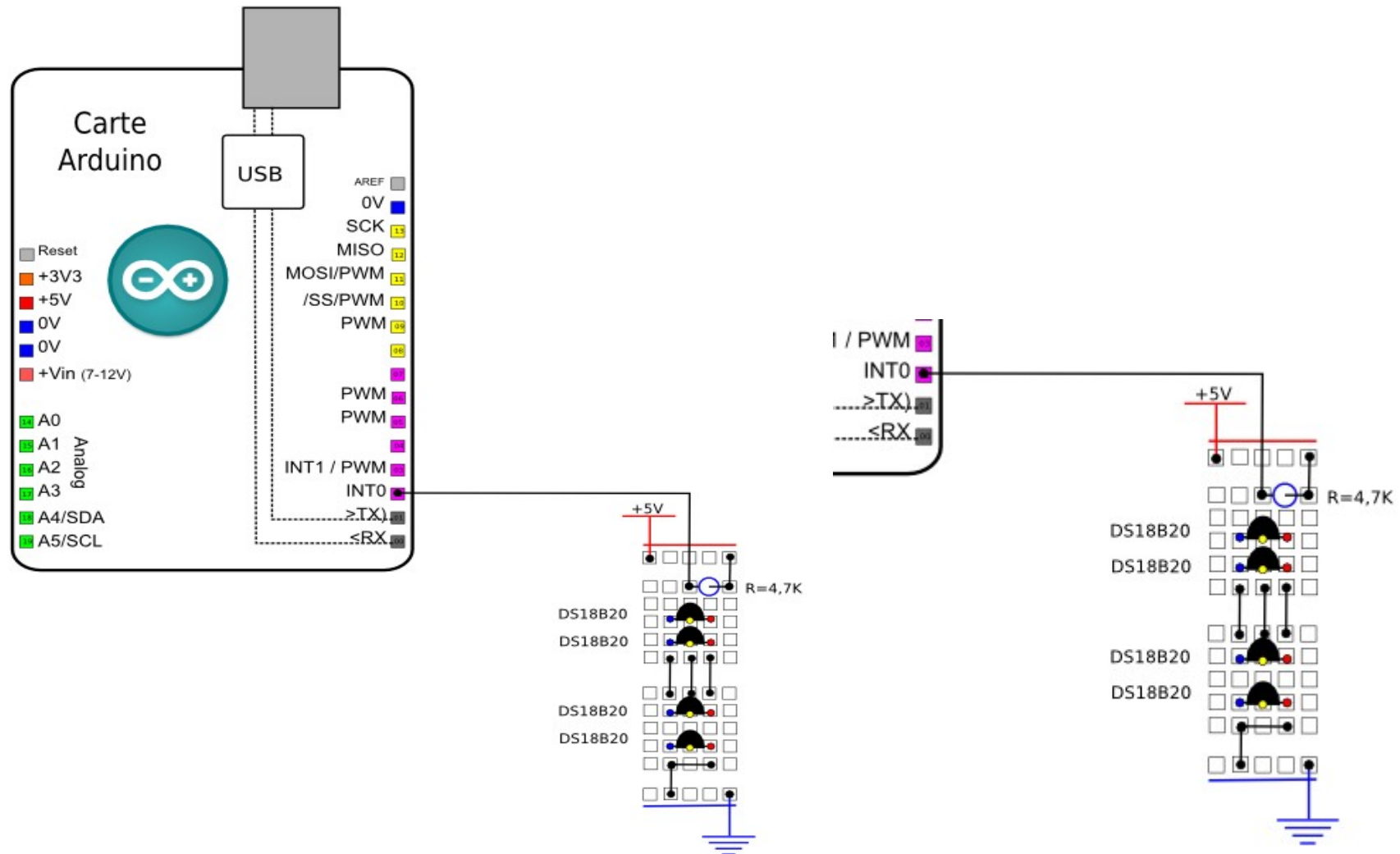
---- Resultat de mesure de la temperature ----
Verification du code CRC des Registres : VALIDE !
Mesure brute =278
Mesure Finale =17.37 Degres Celsius.

---- Resultat de mesure de la temperature ----
Verification du code CRC des Registres : VALIDE !
Mesure brute =277
Mesure Finale =17.31 Degres Celsius.
```

Cette fois vous y êtes : vous avez une mesure de la température !
Reste à tester l'utilisation de plusieurs capteurs simultanément.

15. Exemple 1-wire : DS18B20 : Détecter des capteurs multiples et obtenir leurs adresses : le montage

- Ce qui est bien, mais alors très bien, avec les capteurs 1-wire, c'est qu'il est très simple et possible de les utiliser à plusieurs sur la même broche Arduino. Il suffit pour cela de les connecter en parallèle entre eux, tout simplement, et une seule résistance de rappel suffit pour l'ensemble, ce qui donne :



16. Exemple 1-wire : DS18B20 : Détecter des capteurs multiples et obtenir leurs adresses : le programme

Ce qu'on va faire ici...

- Nous allons ici nous contenter de détecter tous les capteurs présents et de récupérer leur adresse 64 bits, ainsi que réaliser un test de contrôle de la bonne transmission série de l'adresse pour chaque capteur.

Entête déclarative

Inclusion des librairies utiles

- On commence par inclure les librairies
 - on inclut la librairie **OneWire** qui permet d'utiliser un dispositif 1-wire. Cette librairie doit avoir été installée dans le répertoire <Libraries>

Déclaration broche utilisée

- On déclare la broche utilisée avec le capteur. **Une même broche pourra être utilisée pour autant de capteurs qu'on le souhaite !**

Variables utiles

- On déclare un tableau de 8 octets qui va servir à stocker l'adresse 64 bits du capteur (8 octets x 8 bits = 64 bits) .
- On déclare également une variable de comptage des capteurs présents sur le bus One-Wire.

Objet utile

- On déclare un objet OneWire qui représente le capteur 1-wire utilisé, ici le DS18B20, en précisant la broche utilisée.

```
// --- Inclusion des librairies utilisées ---
#include <OneWire.h> // librairie pour capteur OneWire

// --- Déclaration des constantes ---

// --- constantes des broches ---

const int broche_OneWire=2; //declaration constante de broche

// --- Déclaration des variables globales ---
byte adresse[8]; // Tableau de 8 octets pour stockage du code d'adresse 64 bits du composant One Wire

int compt=0; // variable de comptage du nombre de capteurs sur le bus One Wire

// --- Déclaration des objets utiles pour les fonctionnalités utilisées ---
OneWire capteur(broche_OneWire); // crée un objet One Wire sur la broche voulue
```

Fonction **setup()**

Initialisation série

- On initialise le port série à 115200 Bauds

Détection du capteur

- Un capteur est détecté à l'aide de la fonction **search()** qui renvoie **true** si un capteur est présent, **False** sinon.
- A la différence du programme n'utilisant qu'un capteur, ici, on exécute le même code tant qu'un capteur est détecté en plaçant la fonction **search** au sein d'une boucle **while()**, avec à chaque fois qu'un capteur est présent/détecté, l'affichage d'un message et de son adresse 64 bits

Affichage de l'adresse 64 bits au format hexadécimal

- La fonction **search()** précédemment appelée a reçu en paramètre le tableau de 8 octets déclaré dans l'entête : l'adresse 64 bits va automatiquement être stockée dans ce tableau.
- Ensuite, à l'aide d'une simple boucle **for**, on défile les octets du tableau et on les affiche au format hexadécimal :

```
void setup() { // debut de la fonction setup()

Serial.begin(115200); // initialise connexion série à 115200 bauds
// IMPORTANT : régler le terminal côté PC avec la même valeur de transmission

//---- détection des capteurs présents sur le bus One Wire
Serial.println("*** Liste des elements presents sur le bus 1-wire *** ");

while (capteur.search(adresse)== true) { // tant qu'un nouveau capteur est détecté
// la fonction search renvoie la valeur VRAI si un élément 1-wire est trouvé. Stocke son adresse dans le tableau adresse
// adresse correspond à l'adresse de début du tableau adresse[8] déclaré ...

// ce code est exécuté pour chaque capteur détecté
compt=compt+1; // incrémente la variable de comptage du nombre de compteurs
Serial.print ("Numero ");
Serial.print (compt);
Serial.print (" : 1 capteur 1-wire present avec code adresse 64 bits : "); // la fonction search renvoie la valeur VRAI si un élément 1-wire est trouvé.
// Stocke son adresse dans le tableau adresse
// adresse correspond à l'adresse de début du tableau adresse[8] déclaré ...

//--- affichage des 64 bits d'adresse au format hexadécimal
for(int i = 0; i < 8; i++) { // l'adresse renvoyée par la fonction search est stockée sur 8 octets

if (adresse[i]<16) Serial.print('0'); // pour affichage des 0 poids fort au format hexadécimal
Serial.print(adresse[i], HEX); // affiche 1 à 1 les 8 octets du tableau adresse au format hexadécimal
Serial.print(" ");

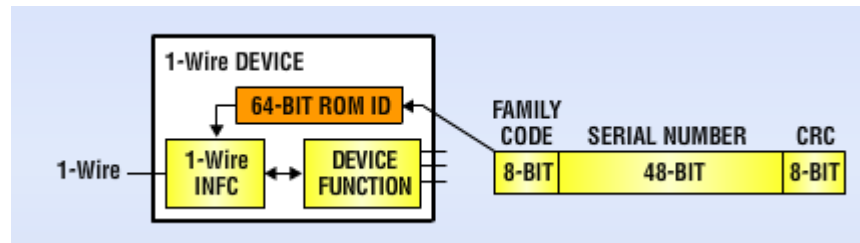
} // fin for

Serial.println();
```

Fonction **setup()** (suite)

Vérification du type du capteur

- Comme nous l'avons dit précédemment, l'adresse 64 bits est constituée de la façon suivante :



- Le premier octet correspond au type du capteur et vaut 0x28 pour un capteur DS18B20, 0x10 pour un DS18S20, 0x22 pour un DS1820. Ici, on se contente de vérifier que nous avons bien à faire à un 18B20 (code 0x28) :

Note : un nombre sous la forme **0x123** correspond au nombre hexadécimal **123**

Vérification du code de contrôle

- La bibliothèque OneWire intègre une fonction, la fonction **crc8()** qui permet de contrôler la validité de toute donnée reçue en provenance du capteur. Il suffit de vérifier la concordance entre le code CRC reçu et le code fourni par la fonction **crc8** pour être sûr que la transmission des données reçues est valide. Ici, on teste tout simplement la validité du code CRC de l'adresse reçue :

```
//---- test du type de capteur ----
// le type du capteur est donné par le 1er octet du code adresse 64 bits
// Valeur 0x28 pour capteur type DS18B20, 0x10 pour type DS18S20, 0x22 pour type DS1820
if (adresse[0]==0x28) Serial.println ("Type : Capteur temperature DS18B20.");

//----- contrôle du code CRC ----
// le dernier octet de l'adresse 64bits est un code de contrôle CRC
// à l'aide de la fonction crc8 on peut vérifier si ce code est valide
if (capteur.crc8( adresse, 7) == adresse[7]) // vérification validité code CRC de l'adresse 64 bits
// le code CRC de l'adresse 64 bits est le 8ème octet de l'adresse (index 7 du tableau)
{
    Serial.println ("Verification du code CRC de l'adresse 64 bits de ce capteur : VALIDE !");
}
else
{
    Serial.println ("Verification du code CRC de l'adresse 64 bits de ce capteur : NON VALIDE !");
}

Serial.println("-----");
} // fin boucle while de test présence capteur
```

Pour plus de détails, voir : http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.LibrairieOneWirecrc8

Message de synthèse

- Ensuite, on affiche le résultat des courses en se basant sur la valeur de la variable de comptage du nombre de capteurs trouvés, ce qui donne :

```
if (compt==0) // si aucun capteur n'a été détecté
{
  Serial.println("Aucun capteur present sur le bus 1-wire"); // affiche message + saut de ligne
}
else // si au moins 1 capteur a été détecté
{
  Serial.print(compt); // affiche nombre de capteurs détectés
  Serial.println (" capteur(s) detecte(s) sur ce bus 1-wire"); // affiche message + saut de ligne
  Serial.println ("**** Recherche terminee - fin de liste **** "); // affiche message + saut de ligne
} // fin else

} // fin de la fonction setup()
```

Fonction **loop()**

- Ici, la fonction loop() est laissée vide

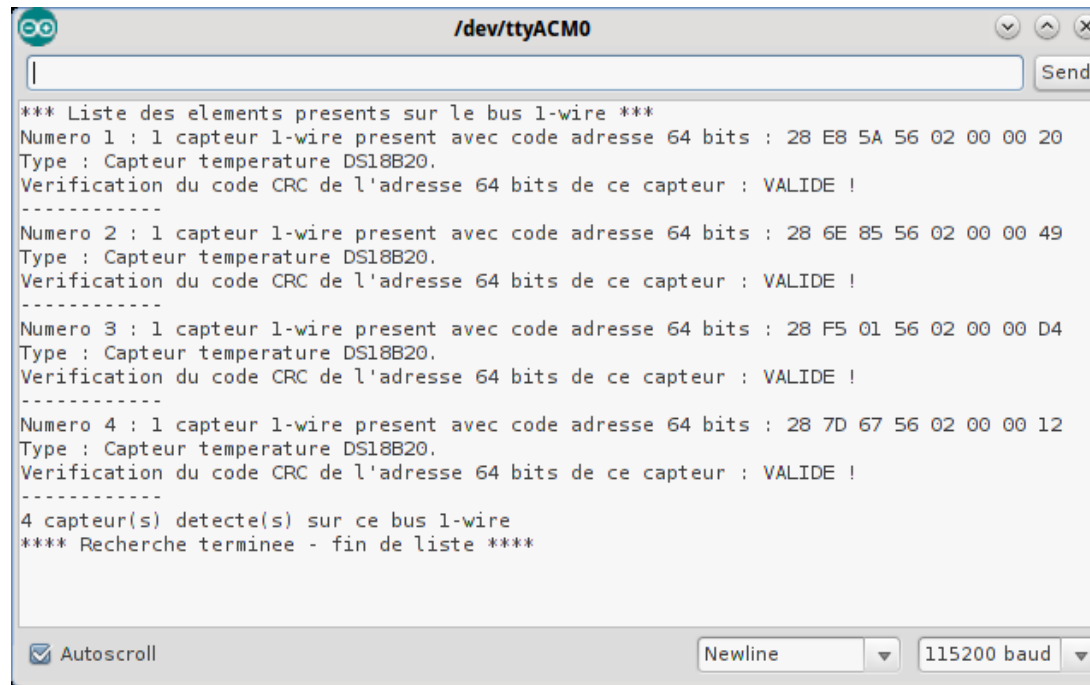
```
void loop(){ // debut de la fonction loop()

// --- ici instructions à exécuter par le programme principal ---

} // fin de la fonction loop() - le programme recommence au début de la fonction loop sans fin
```

Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne ...



```
*** Liste des elements presents sur le bus 1-wire ***
Numero 1 : 1 capteur 1-wire present avec code adresse 64 bits : 28 E8 5A 56 02 00 00 20
Type : Capteur temperature DS18B20.
Verification du code CRC de l'adresse 64 bits de ce capteur : VALIDE !
-----
Numero 2 : 1 capteur 1-wire present avec code adresse 64 bits : 28 6E 85 56 02 00 00 49
Type : Capteur temperature DS18B20.
Verification du code CRC de l'adresse 64 bits de ce capteur : VALIDE !
-----
Numero 3 : 1 capteur 1-wire present avec code adresse 64 bits : 28 F5 01 56 02 00 00 D4
Type : Capteur temperature DS18B20.
Verification du code CRC de l'adresse 64 bits de ce capteur : VALIDE !
-----
Numero 4 : 1 capteur 1-wire present avec code adresse 64 bits : 28 7D 67 56 02 00 00 12
Type : Capteur temperature DS18B20.
Verification du code CRC de l'adresse 64 bits de ce capteur : VALIDE !
-----
4 capteur(s) detecte(s) sur ce bus 1-wire
**** Recherche terminee - fin de liste ****
```

Pour chaque capteur 1-wire : récupération de l'adresse 64 bits du capteur, test du type du capteur, vérification de la validité du code de contrôle CRC...
Et tout ça sur 1 seule broche Arduino !! Cool non ? Notre petit code ne fait même pas 5Ko en plus...



17. Exemple 1-wire : DS18B20 : Mesure de la température à l'aide de capteurs multiples : le programme

Ce qu'on va faire ici...

- Partis sur notre lancée, on va pouvoir maintenant réaliser une mesure sur les différents capteurs. Par contre, ici pour identifier correctement les capteurs, on va **mémoriser les adresses des capteurs détectés dans un tableau à 2 dimensions de 9 octets x n capteurs**. Mais bon, ça, c'est rien pour vous arrivés à ce stade..

Entête déclarative

Inclusion des librairies utiles

- On commence par inclure les librairies
 - on inclut la librairie **OneWire** qui permet d'utiliser un dispositif 1-wire. Cette librairie doit avoir été installée dans le répertoire <Libraries>

Déclaration broche utilisée

- On déclare la broche utilisée avec le capteur. **Une même broche pourra être utilisée pour autant de capteurs qu'on le souhaite !**

Variables utiles

- On déclare un tableau 2D de 8 octets x n capteurs qui va servir à stocker l'adresse 64 bits du capteur (8 octets x 8 bits = 64 bits) ainsi qu'un tableau de 8 octets pour stocker l'adresse courante.
- On déclare également une variable de comptage des capteurs présents sur le bus One-Wire.
- On déclare un tableau de 12 octets pour accéder aux octets de la RAM du capteur, et des variables pour le calcul de la température
- On déclare les codes d'instructions utilisés

Objet utile

- On déclare un objet OneWire qui représente le capteur 1-wire utilisé, ici le DS18B20, en précisant la broche utilisée.

```
// --- Inclusion des librairies utilisées ---
#include <OneWire.h> // librairie pour capteur OneWire

// --- constantes des broches ---
const int broche_OneWire=2; //déclaration constante de broche

// --- Déclaration des variables globales ---
byte adresse[8][10]; // Tableau de 8 octets pour stockage du code d'adresse 64 bits du composant One Wire - jusqu'à 10 capteurs ici
byte adresseCourante[8]; // Tableau de 8 octets pour stockage du code d'adresse courante
int compt=0; // variable de comptage du nombre de capteurs sur le bus One Wire

byte data[12]; // Tableau de 12 octets pour lecture des 9 registres de RAM et des 3 registres d'EEPROM du capteur One Wire
int temperature=0; // variable entiere pour stocker la température brute
float temperaturef=0.0; // variable décimale pour stocker la température réelle

//---- code d'instruction du capteur
const int modeLecture=0xBE;
const int lancerMesure=0x44;

// --- Déclaration des objets utiles pour les fonctionnalités utilisées ---
OneWire capteur(broche_OneWire); // crée un objet One Wire sur la broche voulue
```

Fonction **setup()**

Initialisation série

- On initialise le port série à 115200 Bauds

Détection du capteur

- Un capteur est détecté à l'aide de la fonction **search()** qui renvoie **true** si un capteur est présent, **False** sinon.
- A la différence du programme n'utilisant qu'un capteur, ici, on exécute le même code tant qu'un capteur est détecté en plaçant la fonction **search** au sein d'une boucle **while()** avec à chaque fois qu'un capteur est présent/détecté, l'affichage d'un message et de son adresse 64 bits

Affichage de l'adresse 64 bits au format hexadécimal

- La fonction **search()** précédemment appelée a reçu en paramètre le tableau de 8 octets déclaré dans l'entête : l'adresse 64 bits va automatiquement être stockée dans ce tableau.
- Ensuite, à l'aide d'une simple boucle **for**, on défile les octets du tableau et on les affiche au format hexadécimal : le point clé ici : on stocke l'adresse courante du capteur dans le tableau 2D précédemment déclaré :

```
void setup() { // debut de la fonction setup()

Serial.begin(115200); // initialise connexion série à 115200 bauds
// IMPORTANT : régler le terminal côté PC avec la même valeur de transmission

//---- détection des capteurs présents sur le bus One Wire
Serial.println("*** Liste des elements presents sur le bus 1-wire *** ");

while (capteur.search(adresseCourante)== true) { // tant qu'un nouveau capteur est détecté
// la fonction search renvoie la valeur VRAI si un élément 1-wire est trouvé. Stocke son adresse dans le tableau adresse
// adresse correspond à l'adresse de début du tableau adresse[8] déclaré ...

// ce code est exécuté pour chaque capteur détecté
compt=compt+1; // incrémente la variable de comptage du nombre de compteurs
Serial.print ("Numero ");
Serial.print (compt);
Serial.print (" : 1 capteur 1-wire present avec code adresse 64 bits : "); // la fonction search renvoie la valeur VRAI si un élément 1-wire est trouvé. Stocke son
adresse dans le tableau adresse
// adresse correspond à l'adresse de début du tableau adresse[8] déclaré ...

//--- affichage des 64 bits d'adresse au format hexadécimal
for(int i = 0; i < 8; i++) { // l'adresse renvoyée par la fonction search est stockée sur 8 octets

if (adresseCourante[i]<16) Serial.print('0'); // pour affichage des 0 poids fort au format hexadécimal
Serial.print(adresseCourante[i], HEX); // affiche 1 à 1 les 8 octets du tableau adresse au format hexadécimal
Serial.print(" ");

adresse[i][compt]=adresseCourante[i]; // mémorise dans le tableau 2D l'adresse courante

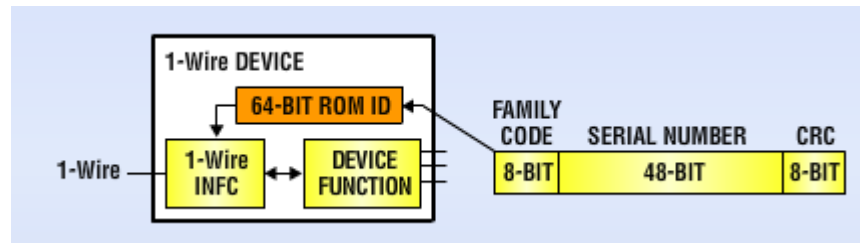
} // fin for

Serial.println();
```


Fonction **setup()** (suite)

Vérification du type du capteur

- Comme nous l'avons dit précédemment, l'adresse 64 bits est constituée de la façon suivante :



- Le premier octet correspond au type du capteur et vaut 0x28 pour un capteur DS18B20, 0x10 pour un DS18S20, 0x22 pour un DS1820. Ici, on se contente de vérifier que nous avons bien à faire à un 18B20 (code 0x28) :

Note : un nombre sous la forme **0x123** correspond au nombre hexadécimal **123**

Vérification du code de contrôle

- La librairie OneWire intègre une fonction, la fonction **crc8()** qui permet de contrôler la validité de toute donnée reçue en provenance du capteur. Il suffit de vérifier la concordance entre le code CRC reçu et le code fourni par la fonction **crc8** pour être sûr que la transmission des données reçues est valide. Ici, on teste tout simplement la validité du code CRC de l'adresse reçue :

```
//---- test du type de capteur ----
// le type du capteur est donné par le 1er octet du code adresse 64 bits
// Valeur 0x28 pour capteur type DS18B20, 0x10 pour type DS18S20, 0x22 pour type DS1820
if (adresseCourante[0]==0x28) Serial.println ("Type : Capteur temperature DS18B20.");

//----- contrôle du code CRC ----
// le dernier octet de l'adresse 64bits est un code de contrôle CRC
// à l'aide de la fonction crc8 on peut vérifier si ce code est valide
if (capteur.crc8( adresseCourante, 7) == adresseCourante[7]) // vérification validité code CRC de l'adresse 64 bits
// le code CRC de l'adresse 64 bits est le 8ème octet de l'adresse (index 7 du tableau)
{
    Serial.println ("Verification du code CRC de l'adresse 64 bits de ce capteur : VALIDE !");
}
else
{
    Serial.println ("Verification du code CRC de l'adresse 64 bits de ce capteur : NON VALIDE !");
}

Serial.println("-----");
} // fin boucle while de test présence capteur
```

Pour plus de détails, voir : http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.LibrairieOneWirecrc8

Message de synthèse

- Ensuite, on affiche le résultat des courses en se basant sur la valeur de la variable de comptage du nombre de capteurs trouvés, ce qui donne :

```
if (compt==0) // si aucun capteur n'a été détecté
{
  Serial.println("Aucun capteur present sur le bus 1-wire"); // affiche message + saut de ligne
}
else // si au moins 1 capteur a été détecté
{
  Serial.print(compt); // affiche nombre de capteurs détectés
  Serial.println (" capteur(s) detecte(s) sur ce bus 1-wire"); // affiche message + saut de ligne
  Serial.println ("**** Recherche terminee - fin de liste **** "); // affiche message + saut de ligne
} // fin else

} // fin de la fonction setup()
```

Fonction `loop()`

- A la différence des codes utilisant un seul capteur, ici nous allons utiliser une boucle pour défiler les capteurs détectés et nous récupérerons les données d'adresse du capteur voulu dans le tableau 2D en le copiant dans l'octet de stockage de l'adresse courante. Ensuite, le code est identique à ce que nous avons fait pour un capteur.
- Puis on lance une lecture des registres en envoyant l'instruction de lecture vers le capteur avec la séquence `reset()` - `select()` - `write()` utilisant les fonctions de la librairie `OneWire()`
- Une fois fait, on lit les données reçues avec l'instruction `read()`
- Ensuite, on réalise le test de validité des données à l'aide de la fonction `crc8()` et du code CRC reçu (octet 8)

```
void loop(){ // debut de la fonction loop()

  for (int index=1; index<=compt; index++) { // défile les 4 capteurs - 1er capteur = index 1 !

    // récupère adresse du capteur courant dans tableau 2D
    for (int j=0; j<8; j++) {adresseCourante[j]=adresse[j][index];}

    //----- lancer une mesure -----
    capteur.reset(); // initialise le bus 1-wire avant la communication avec un capteur donné
    capteur.select(adresseCourante); // sélectionne le capteur ayant l'adresse 64 bits contenue dans le tableau envoyé à la fonction
    capteur.write(lancerMesure,1); // lance la mesure et alimente le capteur par la broche de donnée

    //----- pause d'une seconde -----
    delay(1000); // au moins 750 ms
    // il faudrait mettre une instruction capteur.depower ici, mais le reset va le faire

    //----- lancer une LECTURE des Registres -----
    capteur.reset(); // initialise le bus 1-wire avant la communication avec un capteur donné
    capteur.select(adresseCourante); // sélectionne le capteur ayant l'adresse 64 bits contenue dans le tableau envoyé à la fonction
    capteur.write(modeLecture,1); // passe en mode lecture de la RAM du capteur

    // ----- lire les 9 octets de la RAM (appelé Scratchpad) ----

    for ( int i = 0; i < 9; i++) { // 9 octets de RAM stockés dans 9 octets
      data[i] = capteur.read(); // lecture de l'octet de rang i stocké dans tableau data
    }

    //----- test de validité des valeurs reçues par contrôle du code CRC ----

    Serial.println("");
    Serial.println("---- Resultat de mesure de la temperature ---- ");

    // le dernier (9ème) octet de la RAM est un code de contrôle CRC
    // à l'aide de la fonction crc8 on peut vérifier si ce code est valide
    if (capteur.crc8( data, 8) == data[8]) // vérification validité code CRC des valeurs reçues
    {
      Serial.println ("Verification du code CRC des Registres : VALIDE !");
    }
    else
    {
      Serial.println ("Verification du code CRC des Registres : NON VALIDE !");
    }
  }
}
```

Fonction `loop()` (suite)

- A présent, nous allons lire les 2 octets contenant la valeur de la mesure. Je rappelle que les 2 octets de la mesure ont le format suivant :

	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

- et par un jeu de décalage des bits (on ne garde que les bits utiles de poids fort, on décale vers 8 bits à gauche et on ajoute les bits de poids faible) : on obtient la valeur finale entière correspondante qui est stockée dans une seule variable int.
- Puis, on réalise la conversion de la valeur pour obtenir la valeur décimale en degrés Celcius,
- ce qui donne :

```
//----- réalisation d'une mesure de température---
data[1]=data[1] & B10000111; // met à 0 les bits de signes inutiles
temperature=data[1]; // bits de poids fort
temperature=temperature<<8; // décalage de 8 bits vers la gauche = vers les bits de poids fort
temperature=temperature+data[0]; // bits de poids faible ici mis dans les 8 premiers bits

Serial.print ("Mesure brute =");
Serial.println (temperature);

// --- en mode 12 bits, la résolution est de 0.0625°C - cf datasheet DS18B20
temperaturef=float(temperature)*6.25;
temperaturef=temperaturef/100.0;

Serial.print ("Mesure Finale =");
Serial.print (temperaturef,2);
Serial.println (" Degres Celsius. ");

} // fin for index

while(1); // stop loop

} // fin de la fonction loop()
```

Remarque :

Ici, les températures ne sont pas mémorisées dans un tableau, mais il pourrait être judicieux de le faire au besoin.

Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne ...

```
*** Liste des elements presents sur le bus 1-wire ***
Numero 1 : 1 capteur 1-wire present avec code adresse 64 bits : 28 E8 5A 56 02 00 00 20
Type : Capteur temperature DS18B20.
Verification du code CRC de l'adresse 64 bits de ce capteur : VALIDE !
-----
Numero 2 : 1 capteur 1-wire present avec code adresse 64 bits : 28 6E 85 56 02 00 00 49
Type : Capteur temperature DS18B20.
Verification du code CRC de l'adresse 64 bits de ce capteur : VALIDE !
-----
Numero 3 : 1 capteur 1-wire present avec code adresse 64 bits : 28 F5 01 56 02 00 00 D4
Type : Capteur temperature DS18B20.
Verification du code CRC de l'adresse 64 bits de ce capteur : VALIDE !
-----
Numero 4 : 1 capteur 1-wire present avec code adresse 64 bits : 28 7D 67 56 02 00 00 12
Type : Capteur temperature DS18B20.
Verification du code CRC de l'adresse 64 bits de ce capteur : VALIDE !
-----
4 capteur(s) detecte(s) sur ce bus 1-wire
**** Recherche terminee - fin de liste ****

---- Resultat de mesure de la temperature ----
Verification du code CRC des Registres : VALIDE !
Mesure brute =286
Mesure Finale =17.87 Degres Celsius.

---- Resultat de mesure de la temperature ----
Verification du code CRC des Registres : VALIDE !
Mesure brute =287
Mesure Finale =17.94 Degres Celsius.

---- Resultat de mesure de la temperature ----
Verification du code CRC des Registres : VALIDE !
Mesure brute =290
Mesure Finale =18.12 Degres Celsius.

---- Resultat de mesure de la temperature ----
Verification du code CRC des Registres : VALIDE !
Mesure brute =288
Mesure Finale =18.00 Degres Celsius.
```

Pour chaque capteur 1-wire : récupération de l'adresse 64 bits du capteur, test du type du capteur, vérification de la validité du code de contrôle CRC...
puis mesure de chacun des capteurs à partir des adresses stockées dans le tableau 2D....

Noter que le programme est identique, même si vous utilisez 10 capteurs ou plus ! (adapter la taille du tableau 2D en conséquence)



18. Annexe : 1-wire : DS18B20 : Utilisation d'une librairie Arduino dédiée : le programme

Ce qu'on va faire ici...

- Pour que votre information soit complète au sujet de ce capteur, je vous informe de l'existence d'une librairie Arduino qui peut s'utiliser en complément de la librairie **OneWire**. Cette librairie est une « surcouche » sur la librairie **OneWire** et offre des fonctions dédiées potentiellement utiles.
- La librairie s'appelle **DallasTemperature** et est disponible ici : <https://github.com/milesburton/Arduino-Temperature-Control-Library>

Personnellement, je ne suis pas sûr que cette librairie simplifie tant que ça l'utilisation du capteur... à vous de voir.

- Les classes disponibles avec cette librairie sont :
 - **DallasTemperature**
 - **OneWire**
 - **AlarmHandler**
 - **DeviceAddress**
- Les fonctions fournies par cette librairie sont :

◦ setResolution	◦ validAddress
◦ getResolution	◦ isConnected
◦ getTempC	◦ readScratchPad
◦ toFahrenheit	◦ writeScratchPad
◦ getTempF	◦ readPowerSupply
◦ getTempCByIndex	◦ setHighAlarmTemp
◦ getTempFByIndex	◦ setLowAlarmTemp
◦ setWaitForConversion	◦ getHighAlarmTemp
◦ getWaitForConversion	◦ getLowAlarmTemp
◦ requestTemperatures	◦ resetAlarmSearch
◦ requestTemperaturesByAddress	◦ alarmSearch
◦ requestTemperaturesByIndex	◦ hasAlarm
◦ isParasitePowerMode	◦ toCelsius
◦ begin	◦ processAlarmss
◦ getDeviceCount	◦ setAlarmHandlers
◦ getAddress	◦ defaultAlarmHandler
	◦ calculateTemperature

Je vous propose ici à titre d'illustration la version francisée d'un exemple fourni avec la librairie.

Entête déclarative

- Se reporter aux commentaires pour les détails :

```
#include <OneWire.h>
#include <DallasTemperature.h>

// le capteur est connecté sur la broche 2 - précision 9 bits
#define ONE_WIRE_BUS 2
#define TEMPERATURE_PRECISION 9

// Initialise le bus OneWire pour utiliser un capteur 1-wire quelconque
OneWire oneWire(ONE_WIRE_BUS);

//Transmet le pointeur de l'objet OneWire à la librairie Dallas Temperature.
DallasTemperature sensors(&oneWire);

int numberOfDevices; //variable pour nombre de capteurs trouvés

DeviceAddress tempDeviceAddress; // variable de stockage d'une adresse de capteur trouvé
```

Fonction **setup()**

- Se reporter aux commentaires pour les détails :

```
void setup(void)
{
  Serial.begin(115200);
  Serial.println("Dallas Temperature IC Control Library Demo");

  // Démarre la librairie DallasTempérature
  sensors.begin();

  // Comptage des capteurs
  numberOfDevices = sensors.getDeviceCount();

  // localise les capteurs sur le bus
  Serial.print("Recherche des capteurs...");

  Serial.print("Trouve ");
  Serial.print(numberOfDevices, DEC);
  Serial.println(" capteurs.");

  // message sur l'etat de l'alimentation
  Serial.print("Parasite power est: ");
  if (sensors.isParasitePowerMode()) Serial.println("ON");
  else Serial.println("OFF");

  // Défile les capteurs trouvés et affiche adresse
  for(int i=0;i<numberOfDevices; i++)
  {
    // cherche le capteur pour l'adresse
    if(sensors.getAddress(tempDeviceAddress, i))
    {
      Serial.print("Trouve capteur ");
      Serial.print(i, DEC);
      Serial.print(" avec adresse: ");
      printAddress(tempDeviceAddress);
      Serial.println();

      Serial.print("Fixe la resolution a : ");
      Serial.println(TEMPERATURE_PRECISION, DEC);

      // fixe la résolution à utiliser - jusqu'à 12 bits
      sensors.setResolution(tempDeviceAddress, TEMPERATURE_PRECISION);

      Serial.print("Résolution actuellement a : ");
      Serial.print(sensors.getResolution(tempDeviceAddress), DEC);
      Serial.println();
    }else{
      Serial.print("Trouve capteur fantome a ");
      Serial.print(i, DEC);
      Serial.print(" mais ne peut pas obtenir l'adresse. Verifier cablage et connexion capteur.");
    }
  }
}
```


Fonction **printTemperature()**

- Se reporter aux commentaires pour les détails :

```
// fonction pour afficher la température d'un capteur
void printTemperature(DeviceAddress deviceAddress)
{
  // method 1 - slower
  //Serial.print("Temp C: ");
  //Serial.print(sensors.getTempC(deviceAddress));
  //Serial.print(" Temp F: ");
  //Serial.print(sensors.getTempF(deviceAddress)); // Makes a second call to getTempC and then converts to Fahrenheit

  // method 2 - faster
  float tempC = sensors.getTempC(deviceAddress);
  Serial.print("Temp C: ");
  Serial.print(tempC);
  Serial.print(" Temp F: ");
  Serial.println(DallasTemperature::toFahrenheit(tempC)); // Converts tempC to Fahrenheit
}
```

Fonction **loop()**

- Se reporter aux commentaires pour les détails :

```
void loop(void)
{
    // requete de température pour tous les capteurs du bus
    Serial.print("Lance mesure des temperatures...");
    sensors.requestTemperatures(); // envoi la commande pour obtenir la température
    Serial.println("DONE");

    // défile les capteurs et affiche la température
    for(int i=0;i<numberOfDevices; i++)
    {
        // Search the wire for address
        if(sensors.getAddress(tempDeviceAddress, i))
        {
            // Affiche l'index du capteur
            Serial.print("Temperature du capteur : ");
            Serial.println(i,DEC);

            printTemperature(tempDeviceAddress); // Appelle la fonction pour afficher les données
        }
    }
}
```

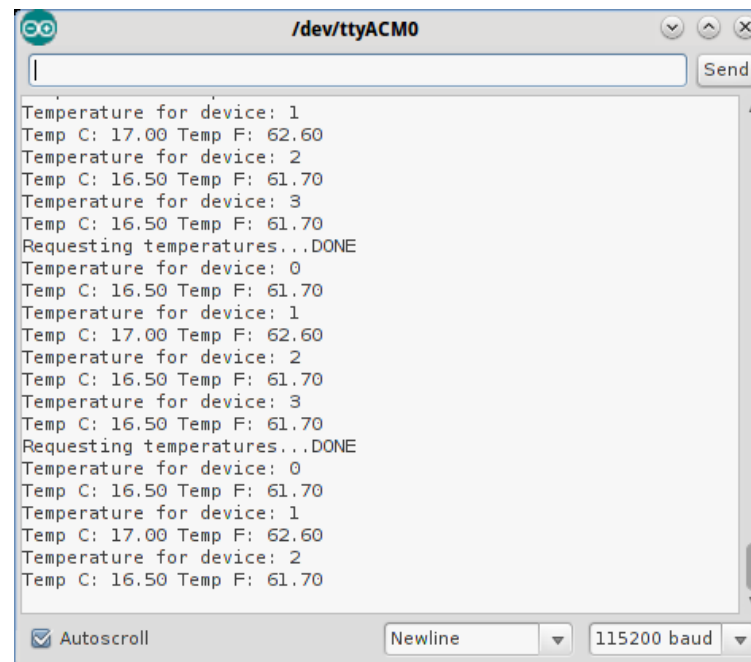
Fonction **printAddress()**

- Se reporter aux commentaires pour les détails :

```
// Fonction pour afficher l'adresse d'un capteur
void printAddress(DeviceAddress deviceAddress)
{
  for (uint8_t i = 0; i < 8; i++)
  {
    if (deviceAddress[i] < 16) Serial.print("0");
    Serial.print(deviceAddress[i], HEX);
  }
}
```

Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne ...



L'impression que j'en ai, c'est que cette librairie fait un peu doublon avec la librairie OneWire... Je vous laisse juge.

19. Les éléments du langage Arduino étudiés dans cet atelier

Librairie OneWire :

Fonctions d'initialisation

- `myWire.search(addrArray)`
- `myWire.reset_search()`
- `myWire.skip()`

Fonctions de communication

- `myWire.reset()`
- `myWire.select(addrArray)`
- `myWire.write(num)`
- `myWire.write(num, 1)`
- `myWire.read()`

Fonction de contrôle des données

- `myWire.crc8(dataArray, length)`

Fonction de contrôle de l'alimentation

- `myWire.depower()`

La documentation complète du langage Arduino en français est disponible ici :
http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.ReferenceMaxi

20. A présent, vous devriez être capable :

- de comprendre et d'expliquer la communication série 1-wire
- d'utiliser la librairie OneWire
- d'interfacer votre carte Arduino avec un capteur de température 1-wire très pratique, le DS18B20.

Table des matières

Utiliser la communication série 1-Wire avec Arduino : l'exemple du capteur de température DS18B20.

Intro |

Matériel nécessaire pour les ateliers Arduino |

Matériel spécifique nécessaire pour cet atelier |

Technique : la communication 1-Wire : principe général |

Technique : 1-Wire : Séquence de transmission des données |

Info : Communication 1-Wire : Principe de l'alimentation par 1 seul fil de masse |

Pour info : Communication 1-Wire : Notion de contrôle par « Calcul de Redondance Cyclique » |

Exemple de dispositif 1-wire : le capteur de température DS18B20 |

Les codes d'instructions du DS18B20 |

Utiliser un DS18B20 (capteur 1-wire) avec la librairie Arduino OneWire |

Exemple 1-wire : DS18B20 : Détecter le capteur et obtenir son adresse : le montage |

Exemple 1-wire : DS18B20 : Détecter le capteur et obtenir son adresse : le programme |

Exemple 1-wire : DS18B20 : Afficher le contenu des registres de données : le programme |

Exemple 1-wire : DS18B20 : Lancer une mesure, lire et contrôler et afficher la mesure obtenue. |

Exemple 1-wire : DS18B20 : Détecter des capteurs multiples et obtenir leurs adresses : le montage |

Exemple 1-wire : DS18B20 : Détecter des capteurs multiples et obtenir leurs adresses : le programme |

Exemple 1-wire : DS18B20 : Mesure de la température à l'aide de capteurs multiples : le programme |

Annexe : 1-wire : DS18B20 : Utilisation d'une librairie Arduino dédiée : le programme |

Les éléments du langage Arduino étudiés dans cet atelier |

A présent, vous devriez être capable : |

Bravo !
vous avez terminé cet atelier Arduino !



Prêt pour la suite ? Retrouvez de nombreux autres thèmes d'ateliers Arduino ici :

http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ATELIERS